

z/VM
7.3

*RSCS Networking
Diagnosis*



Note:

Before you use this information and the product it supports, read the information in [“Notices” on page 267.](#)

This edition applies to version 7, release 3 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-11-22

© **Copyright International Business Machines Corporation 1990, 2023.**

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

Contents

Figures.....	xiii
Tables.....	xvii
About This Document.....	xix
Intended Audience.....	xix
Syntax, Message, and Response Conventions.....	xix
Where to Find More Information.....	xxii
Links to Other Documents and Websites.....	xxii
How to provide feedback to IBM.....	xxiii
Summary of Changes for z/VM: RSCS Networking Diagnosis.....	xxv
GC24-6316-73, z/VM 7.3 (December 2023).....	xxv
GC24-6316-73, z/VM 7.3 (September 2022).....	xxv
GC24-6316-01, z/VM 7.2 (September 2020).....	xxv
GC24-6316-00, z/VM 7.1 (September 2018).....	xxv
Part 1. Functional Overview.....	1
Chapter 1. Introduction to RSCS.....	3
Operating Requirements.....	3
Task Overview.....	3
System Tasks.....	3
Link Driver Tasks.....	5
Auto-Answer Task.....	6
Task Interaction.....	6
Processing Files.....	7
Establishing SNA Connections.....	8
Establishing Auto-Dial and Auto-Answer Links.....	8
Processing Commands.....	9
Processing Messages.....	10
Chapter 2. RSCS Structure.....	11
Communication Between Tasks.....	11
Direct Task Interfaces.....	13
Queued Command Interfaces.....	14
Data Structures.....	15
Primary Data Areas.....	16
Defining Network Structure.....	17
Processing Files.....	21
Sharing RSCS Resources.....	25
Disk File Interface.....	26
Defining Tasks.....	27
Managing Unit Record Devices.....	28
Exit Facility.....	29
Message Subscriptions.....	30
Message Request and Work Areas.....	30
Event Scheduler.....	31

Command Authorization.....	31
Printer Related Areas.....	32
Chapter 3. Task Management.....	33
Using GCS Facilities.....	33
Task Management Facilities.....	33
GCS Macros.....	34
Attaching System Tasks.....	35
Starting the Communications Task.....	35
Starting System Tasks.....	35
Starting the SNA Control Task and Auto-Answer Tasks.....	36
Starting Link Driver Tasks.....	36
Link Driver Tasks.....	37
Session Driver as Primary LU.....	37
Session Driver as Secondary LU.....	37
Auto-Start Links.....	37
Auto-Answer Links.....	37
Capturing Task Abends.....	38
End of Task Exit Processing.....	38
Chapter 4. Inter-Task Communication.....	39
Using GCS Services.....	39
Task Synchronization.....	39
Task Queues.....	39
Event Control Blocks.....	40
POST and WAIT Macros.....	40
DMTCOMNQ and DMTCOMDQ.....	41
Task Serialization.....	41
DMTRES.....	42
Disabling Interrupts.....	42
ENQ and DEQ Macros.....	42
Chapter 5. System Tasks.....	43
Communications Task.....	43
Initialization.....	43
Creating System Tasks.....	44
Trapping Special Messages.....	44
Processing Commands.....	45
Shutting Down RSCS.....	46
Spool Manager Task.....	46
Initialization.....	47
Processing Reader Interrupts.....	47
Processing Spool File Commands.....	48
Managing File Routines for Link Drivers.....	50
Serializing Resources.....	51
Managing the Unit Record Device Pool.....	51
Auto-Start Task.....	52
Initialization.....	52
Dynamic Port Allocation.....	52
Processing Commands.....	52
Free Ports.....	53
Timer ECB.....	53
Termination.....	53
Event Manager Task.....	53
Initialization.....	53
Allocating Task IDs.....	53
Event Types.....	54
Timer Management.....	54

Processing SCHEDULE Commands.....	54
EXEC Processor Task.....	55
Initialization.....	55
Processing Exec Queues.....	55
SNA Control Task.....	55
Establishing a Session.....	55
Initializing the SNA Control Task.....	57
VTAM Exit Routines.....	57
Maintaining the RSCS/VTAM Interface.....	59
Port Redirector Task.....	60
Initialization.....	60
Processing LISTEN Requests.....	61
Termination.....	62
Auto-Answer Tasks.....	62
Initialization.....	62
Identifying Callers.....	62
Processing Sign-On Records.....	63
Invoking Links.....	63
Error Processing.....	63
Calling Exit Points.....	63
Chapter 6. Networking Link Drivers.....	65
Common Networking Structures.....	65
Data Areas.....	65
Building NJE Headers.....	65
Receiving NJE Headers.....	66
Receiving and Transmitting NMRs.....	67
General Purpose Routines.....	67
Initializing Storage.....	68
Processing Sign-on Records.....	68
Processing Commands.....	68
Accounting.....	68
Transmitting Buffers.....	69
Receiving Buffers.....	69
SNA LU_TO NJE Session Driver.....	70
Initialization.....	70
Terminating the Link.....	71
BSC and CTC Link Driver.....	71
Initialization.....	71
Terminating the Link.....	72
TCPNJE Link Driver.....	72
Initialization.....	72
Terminating the Link.....	73
GATEWAY Link Driver.....	73
Initialization.....	73
Gateway Service Macros.....	74
Terminating the Link.....	75
Chapter 7. Printer Link Drivers.....	77
3270P Printer Link Driver.....	77
Initialization.....	77
Receiving and Sending Data.....	77
Terminating the Link.....	78
TN3270E Printer Link Driver.....	78
Initialization.....	79
Receiving and Sending Data.....	79
Terminating the Link.....	80
SNA 3270 Printer Session Driver.....	80

Initialization.....	80
Receiving and Sending Data.....	81
Terminating the Link.....	82
ASCII Printer and Plotter Link Driver.....	82
Initialization.....	83
Receiving and Sending Data.....	83
Command Processing.....	84
Processing CP File Characteristics.....	84
Building Data Streams.....	84
I/O Processing.....	84
Terminating the Link.....	85
TCPASCII Printer and Plotter Link Driver.....	85
Initialization.....	86
Receiving and Sending Data.....	86
Command Processing.....	87
Processing CP File Characteristics.....	87
Building Data Streams.....	87
Socket Processing.....	88
Terminating the Link.....	88
Line Printer Daemon (LPD) Link Driver.....	88
Initialization.....	89
Sending and Receiving Data.....	89
Terminating the Link.....	90
Line Printer Remote (LPR) Link Driver.....	90
Initialization.....	90
Sending and Receiving Data.....	91
Terminating the Link.....	92
Chapter 8. Workstation Link Drivers.....	93
RJE Workstation Link Driver.....	93
Initialization.....	93
Receiving and Sending Data.....	93
Terminating the Link.....	94
MRJE Workstation Link Driver.....	94
Initialization.....	95
Receiving and Sending Data.....	95
Terminating the Link.....	96
SNARJE Workstation Session Driver.....	96
Initialization.....	96
Receiving and Sending Data.....	97
Terminating the Link.....	98
Chapter 9. Special Purpose Link Drivers.....	99
List Processor.....	99
Initialization.....	99
Receiving and Sending Files.....	99
Terminating the Link.....	101
NOTIFY Link Driver.....	101
Initialization.....	101
Generating a Note.....	101
Building Note Records.....	102
Purging Files.....	102
Unsolicited File Transfer (UFT) Driver.....	103
Initialization.....	103
Sending and Receiving Data.....	104
Terminating the Link.....	105
Unsolicited File Transfer Daemon (UFTD) Driver.....	105
Initialization.....	106

Sending and Receiving Data.....	106
Terminating the Link.....	107
Chapter 10. Utility Routines.....	109
General Purpose Routines.....	109
Table Search Routines.....	109
Disk File Interface Routine.....	109
Time-of-Day Conversion Routines.....	110
Number/Data Conversion Routines.....	110
Specialized Routines.....	111
Hashed Indexing Routines.....	112
Storage Management Routines.....	113
I/O Interface Routines.....	114
Spool Interface Routines.....	114
Input Spool Routines.....	114
Output Spool Routines.....	115
NETDATA Conversion Routine.....	115
General Parsing Routines.....	116
Task Table Service Routines.....	117
Chapter 11. Parsing Commands and Statements.....	119
Defining Syntax.....	119
CDEF Macro.....	119
LDEF Macro.....	120
PDEF Macro.....	120
DDEF Macro.....	123
Finding Command and Statement Definitions.....	123
Parsing Commands and Statements.....	124
QUERY Command Processing.....	125
Chapter 12. Message Processing.....	127
Message Structure.....	127
Text Messages.....	128
Columnar Messages.....	128
EMSG Settings.....	128
National Language Support.....	128
Command Response Interface.....	128
Message Subscriptions.....	129
Processing Messages.....	129
Preparing to Issue Messages.....	130
Issuing Messages to All Destinations.....	131
Formatting Messages.....	132
Processing Substitution Values.....	133
Message Repositories.....	134
Conversion Repository.....	135
Translation Repository.....	136
Message Compilers.....	138
Part 2. Diagnostic Aids.....	139
Chapter 13. Debugging Considerations.....	141
Abend Processing.....	141
Console Abend Messages.....	141
Abend Dumps.....	142
Reading Dumps.....	142
System Abend Considerations.....	142
Finding RSCS Data Areas.....	142

GCS Considerations.....	143
Active Tasks.....	143
Tracing State Blocks.....	143
Trace Data Format.....	143
Sample CTC Trace (RECORDS Option).....	143
Sample CTC Trace (ALL Option).....	145
Sample SNANJE Trace.....	148
Sample TCPNJE Trace.....	149
Chapter 14. Examining Dumps.....	157
Getting Dump Information.....	157
Checking for a Compressed Load Map.....	157
Using RSCS-Supplied Subcommands.....	157
CVT.....	158
DWA.....	158
IOTABLE.....	159
ITRACE.....	161
LINKS.....	165
NDWA.....	165
RIB.....	166
ROUTES.....	167
TAGQUE.....	168
TIB.....	169
Chapter 15. Solving Problems in RSCS Interchange.....	171
Using REXX Traces.....	171
Using a Log File.....	171
Using Incoming and Outgoing Mail Files.....	172
Using RSCS Diagnosis Commands.....	172
Part 3. Reference Directories.....	173
Chapter 16. Module Directory.....	175
RSCS Modules.....	175
DMTAPT.....	175
DMTAST.....	175
DMTAXA.....	175
DMTAXM.....	175
DMTBOX.....	176
DMTBPL.....	176
DMTCMA.....	177
DMTCMB.....	177
DMTCMQ.....	177
DMTCMX.....	178
DMTCMY.....	178
DMTCMZ.....	179
DMTCOM.....	179
DMTCQC.....	180
DMTCQX.....	181
DMTCQY.....	181
DMTCQZ.....	182
DMTCVT.....	182
DMTDDL.....	182
DMTDUP.....	182
DMTEND.....	182
DMTEQU.....	182
DMTEVE.....	183

DMTEXE.....	183
DMTGPI.....	183
DMTHAS.....	184
DMTIOT.....	184
DMTITR.....	184
DMTIRW.....	185
DMTIRX.....	185
DMTLAX.....	185
DMTLCR.....	185
DMTLIS.....	185
DMTLOG.....	185
DMTLPD.....	186
DMTLPR.....	186
DMTMAN.....	186
DMTMGF.....	186
DMTMGI.....	187
DMTMGS.....	187
DMTMGX.....	187
DMTMPT.....	187
DMTNCR.....	188
DMTNET.....	188
DMTNHD.....	188
DMTNHE.....	189
DMTNOT.....	189
DMTNPT.....	190
DMTNRV.....	190
DMTNTR.....	190
DMTNUS.....	190
DMTPAF.....	191
DMTPAR.....	191
DMTPCR.....	191
DMTPRD.....	191
DMTQSA.....	192
DMTRDR.....	192
DMTRER.....	192
DMTRES.....	192
DMTRES.....	192
DMTREG.....	193
DMTRGX.....	193
DMTRPT.....	193
DMTSCT.....	194
DMTSEP.....	194
DMTSJE.....	194
DMTSLO.....	194
DMTSML.....	195
DMTSNE.....	195
DMTSOK.....	195
DMTSPT.....	195
DMTTAP.....	196
DMTTAS.....	196
DMTTNE.....	196
DMTTPT.....	196
DMTUFD.....	197
DMTUFT.....	197
DMTURO.....	197
DMTVXT.....	197
Exit Points.....	198
Dump Formatting Routines.....	199
DMTYCV.....	199

DMTYDS.....	200
DMTYEX.....	200
DMTYIO.....	200
DMTYIT.....	200
DMTYLI.....	200
DMTYND.....	200
DMTYRI.....	200
DMTYRO.....	200
DMTYTG.....	200
DMTYTI.....	200
Chapter 17. Control Blocks.....	201
Primary Data Areas.....	201
CRV.....	201
CVT.....	202
SYSIDENT.....	204
Network and Task Structure.....	204
DEST.....	204
EQUATE.....	205
LINKTABL.....	205
PORT.....	210
REROUTE.....	210
ROUTEGRP.....	211
TASKBLOK.....	211
Accounting Structures.....	212
ACNTBUFF.....	212
AUTHBLOK.....	212
Printer-Related Structures.....	213
FORM.....	213
RFCBTAB.....	213
SEPBLOK.....	213
File Queueing Structures.....	213
SAFTAG.....	214
TAG.....	214
TAGAREA.....	216
TASHADOW.....	217
TASTORAG.....	217
TCP/IP-Related Structures.....	218
PRDBLOK.....	218
SOCKBLOK.....	218
SOCKCBLK.....	219
SOCKET.....	220
Tracing Structures.....	224
ITRACFRM.....	224
ITRACHDR.....	224
ITRACREC.....	225
Miscellaneous Structures.....	225
EVEBLOK.....	225
HASHBLOK.....	227
IOTABLE.....	227
MONITENT.....	228
SAVEAREA.....	228
Chapter 18. Command and Request Elements.....	231
CMNDAREA.....	231
Basic Structure.....	231
Type A0 (REORDER).....	232
Type A1 (CLOSE, ORDER, PURGE).....	232

Type A1 (TRANSFER).....	233
Type A2 (CHANGE).....	234
Type C0 (FORCE).....	235
Type C1 (ITO).....	235
Type C2 (RETRY).....	235
Type E0 (Execs).....	236
Type L0 (DRAIN, FREE, HOLD, READY, START, and TRACE).....	236
Type L1 (BACKSPACE, FWDSPACE).....	237
Type L2 (FLUSH).....	238
Type L3 (Commands, Messages).....	238
Type V1 (START).....	239
Type V2 (STOP).....	239
MSGBLOK.....	240
RDEVBLOK.....	242
 Chapter 19. Networking Data Areas and Record Formats.....	245
Data Areas and Equates.....	245
BUFFER.....	245
HDRTRL.....	245
NCC.....	246
NJEEQU.....	246
NMR.....	248
RIB.....	249
TANK.....	251
TIB.....	252
XABHDR.....	256
NJE Header Formats.....	257
Job Header Format.....	257
Job Trailer Format.....	259
Data Set Header Format.....	259
Record Formats.....	262
Coded NOP Records.....	262
Segmented Header Formats.....	263
Spanned Record Format.....	263
TCPNJE Record Formats.....	264
Control Record Format.....	264
Data Block Header (TTB).....	265
Data Block Record Header (TTR).....	265
 Notices.....	267
Programming Interface Information.....	268
Trademarks.....	268
Terms and Conditions for Product Documentation.....	269
IBM Online Privacy Statement.....	269
 Bibliography.....	271
Where to Get z/VM Information.....	271
z/VM Base Library.....	271
z/VM Facilities and Features.....	272
Prerequisite Products.....	274
Related Products.....	274
Additional Publications.....	274
 Index.....	275

Figures

1. RSCS Operational Environment.....	3
2. Overview of RSCS Initialization and Task Creation.....	5
3. Sending a File to a Remote Node.....	7
4. Receiving a File from a Remote Node.....	7
5. Processing a Store-and-Forward File.....	8
6. Processing a Store-and-Forward Command.....	9
7. Register Savearea Convention.....	12
8. Direct Interface Between Tasks.....	13
9. Command Processing Interface.....	13
10. Stop Command ECB Interface.....	13
11. VTAM Event Interface.....	13
12. Queued Task Interface.....	14
13. REX Task Command Queue.....	14
14. EXE Task Command Queue.....	15
15. EVE Task Command Queue.....	15
16. Command and Message Element Queue to Link Drivers.....	15
17. CRV Anchored in CVT.....	17
18. Sample ROUTEGRP Hierarchy.....	19
19. Close-up of COUNTRY ROUTEGRP Entry.....	19
20. NODE Entries Chained to the Owing ROUTEGRP.....	20
21. Overview of TAGAREA.....	22
22. Overview of TASTORAG Allocation.....	22
23. Overview of a Page of Storage.....	23

24. Relationship of TAG, TASHADOW, and LINKTABL Elements.....	24
25. TASHADOW Queue Pointers (900 TASHADOWS on Link).....	24
26. Example of Resource Ownership.....	26
27. Data Structures as -Line A3 2- is Read.....	26
28. Structure of the RSCS Task Table.....	28
29. Channels Table Setup with -CHANNELS E F- Specified.....	29
30. Sample EXITBLOK Structure.....	29
31. Overview of MONITENT Structure.....	30
32. Overview of Message Builder Data Areas.....	31
33. Structure of the FCB Table.....	32
34. Processing the INIT Command.....	35
35. Attaching the Communications Task.....	35
36. Attaching Mandatory System Tasks.....	36
37. Processing the Configuration File.....	43
38. A Successful BIND Request.....	56
39. An Unsuccessful BIND Request.....	56
40. Attaching the Dial-Up Task.....	62
41. Structure of GATEWAY Link Driver.....	73
42. Structure of an ASCII-Type Link Driver.....	82
43. Structure of a TCPASCII-Type Link Driver.....	86
44. Initializing the RJE Link Driver Task.....	93
45. Initializing the MRJE Link Driver Task.....	95
46. Initializing the List Processor Task.....	99
47. Simple Distribution List Processed by DMTNTRSB.....	100
48. Distribution List with Private Sections.....	100

49. Initializing the NOTIFY Link Driver.....	101
50. Sample Keyword Table.....	117
51. Sample Entry in the RSCSSTMT Macro.....	119
52. Syntax Structure of TRANSFER Command.....	120
53. Syntax Definition of the DISCONNECT Command.....	121
54. Syntax Definition for NETWORK Command.....	122
55. Syntax Definition for CPQUERY Command.....	122
56. Sample QUERY Command.....	123
57. Overview of Message Processing.....	130
58. Compiling a Conversion Repository.....	135
59. A Message in the Conversion Repository.....	135
60. Compiling a Translation Repository.....	136
61. Structure of a Text Message.....	136
62. Building the Message Text.....	137
63. Building a Columnar Message.....	137
64. Bottom Level Column Headers.....	137
65. Sample Console Abend Message.....	141
66. Output of the CVT Subcommand.....	158
67. Output of DWA Subcommand.....	159
68. Output of the IOTABLE Subcommand.....	160
69. Output of the LINKS Subcommand.....	165
70. Output of RIB Subcommand.....	167
71. Output of the ROUTES Subcommand.....	167
72. Output of the TAGQUE Subcommand.....	169
73. Output of TIB Subcommand.....	170

74. Sample DEBUG Output.....	171
75. Sample RSCS Interchange Server Log File.....	172

Tables

1. Examples of Syntax Diagram Conventions.....xix

2. RSCS System Tasks and Functions..... 4

3. Link Driver Tasks and Associated Nodes..... 5

4. Session Driver Tasks and Associated Nodes..... 6

5. Communications Vector Table..... 16

6. GCS Macros Issued by RSCS Tasks.....34

About This Document

This document describes the debug facilities of IBM® Remote Spooling Communications Subsystem (RSCS) Networking for z/VM. It also describes RSCS diagnostic aids and facilities. It is intended to help you isolate and diagnose any problems that might occur in RSCS. It contains the following information:

- Overviews of RSCS functions
- Techniques and facilities for collecting and processing diagnostic information about RSCS
- Reference information about RSCS modules, entry points, and various data areas that are used for diagnostic purposes

Intended Audience

This information is for anyone who needs to diagnose problems in the RSCS virtual machine.

You should be familiar with assembler language programming techniques and the operating procedures for RSCS. Knowledge of the Group Control System (GCS) and Control Program (CP) components of z/VM® is required. You should also be familiar with TCP/IP for z/VM and the Advanced Communications Function/Virtual Telecommunications Access Method (ACF/VTAM) licensed program product (referred to as VTAM® in this document).





Syntax, Message, and Response Conventions

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The  symbol indicates the beginning of the syntax diagram.
- The  symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.
- The  symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.
- The  symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in [Table 1 on page xix](#).



Table 1. Examples of Syntax Diagram Conventions	
Syntax Diagram Convention	Example
Keywords and Constants A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown. In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.	 KEYWORD 

Table 1. Examples of Syntax Diagram Conventions (continued)

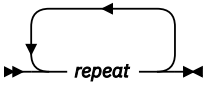
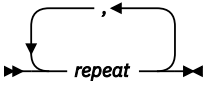
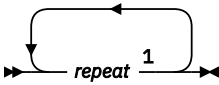
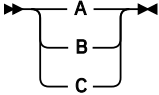
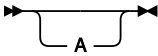
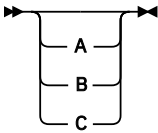
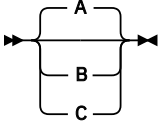
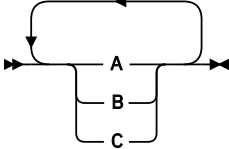
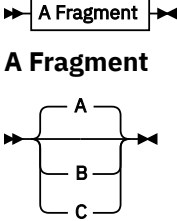
Syntax Diagram Convention	Example
Abbreviations <p>Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated.</p> <p>In this example, you can specify KEYWO, KEYWOR, or KEYWORD.</p>	<p>►► KEYWOrd ◄◄</p>
Symbols <p>You must specify these symbols exactly as they appear in the syntax diagram.</p>	<p>* Asterisk</p> <p>: Colon</p> <p>, Comma</p> <p>= Equal Sign</p> <p>- Hyphen</p> <p>() Parentheses</p> <p>. Period</p>
Variables <p>A variable appears in highlighted lowercase, usually italics.</p> <p>In this example, <i>var_name</i> represents a variable that you must specify following KEYWORD.</p>	<p>►► KEYWOrd — <i>var_name</i> ◄◄</p>
Repetitions <p>An arrow returning to the left means that the item can be repeated.</p> <p>A character within the arrow means that you must separate each repetition of the item with that character.</p> <p>A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated.</p> <p>Syntax notes may also be used to explain other special aspects of the syntax.</p>	<p>  </p> <p>  </p> <p>  </p> <p>Notes: ¹ Specify <i>repeat</i> up to 5 times.</p>
Required Item or Choice <p>When an item is on the line, it is required. In this example, you must specify A.</p> <p>When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.</p>	<p>►► A ◄◄</p> <p>  </p>

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	Example
<p>Optional Item or Choice</p> <p>When an item is below the line, it is optional. In this example, you can choose A or nothing at all.</p> <p>When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.</p>	 
<p>Defaults</p> <p>When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line.</p> <p>In this example, A is the default. You can override A by choosing B or C.</p>	
<p>Repeatable Choice</p> <p>A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item.</p> <p>In this example, you can choose any combination of A, B, or C.</p>	
<p>Syntax Fragment</p> <p>Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name.</p> <p>In this example, the fragment is named "A Fragment."</p>	

Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

- xxx**
Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.
- []**
Brackets enclose optional text that might be displayed.
- { }**
Braces enclose alternative versions of text, one of which will be displayed.
- |**
The vertical bar separates items within brackets or braces.
- ...**
The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

Where to Find More Information

For additional information about RSCS and z/VM, see [“Bibliography” on page 271](#).

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See [How to send feedback to IBM](#) for additional information.

Summary of Changes for z/VM: RSCS Networking Diagnosis

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line (|) to the left of the change.

GC24-6316-73, z/VM 7.3 (December 2023)

This edition includes terminology, maintenance, and editorial changes.

GC24-6316-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

GC24-6316-01, z/VM 7.2 (September 2020)

This edition supports the general availability of z/VM 7.2.

GC24-6316-00, z/VM 7.1 (September 2018)

This edition supports the general availability of z/VM 7.1.

Part 1. Functional Overview

This part describes the functions, tasks, and data areas that enable RSCS to process files, messages, and commands in the network. This information is presented for diagnostic purposes only.

Chapter 1. Introduction to RSCS

This chapter introduces the structure of RSCS, its operational environment, and functions.

Operating Requirements

RSCS Networking for z/VM (RSCS) is an optional feature of z/VM. With RSCS, you can send and receive files, messages, and commands to other systems and devices (called nodes) within RSCS and TCP/IP networks.

As [Figure 1 on page 3](#) shows, RSCS runs as an application of the Group Control System (GCS) component of z/VM. RSCS uses GCS for task management and I/O operations; RSCS also uses GCS to communicate with CP and other virtual machines. RSCS must run in an ESA-mode virtual machine.

To communicate within a System Network Architecture (SNA) network, RSCS requires VTAM.

RSCS and VTAM share the same GCS virtual machine group. Under GCS, VTAM creates an interface that RSCS uses to exchange information with other VTAM applications in the SNA network. VTAM selects the path that RSCS uses to communicate within the SNA network. It also allows RSCS to send and receive data across SNA-type links. In addition, RSCS uses the TCP/IP feature of z/VM to communicate within an IP network.

For information about specific operating requirements, see [z/VM: General Information](#).

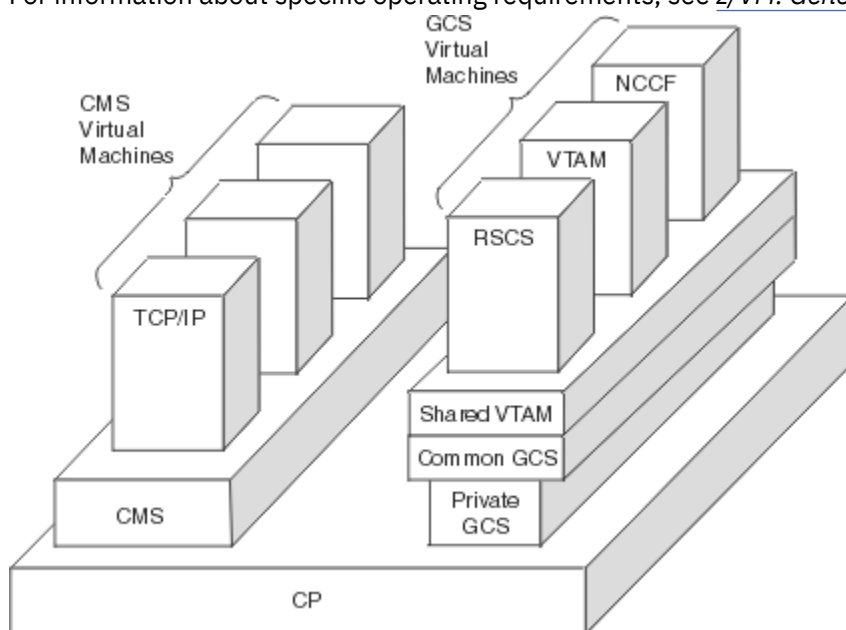


Figure 1. RSCS Operational Environment

Task Overview

RSCS's major functions are performed by various tasks. This section provides an overview of the RSCS system, link driver, and auto-answer tasks.

System Tasks

Most system tasks must be present for RSCS to function properly. The exceptions are the SNA control task, which may be absent if the RSCS/VTAM interface is not running, and the TCP port redirector task, which is not required if there are no TCPNJE-type links defined. The system tasks attach the link driver

and auto-answer tasks as they are needed. See [Chapter 5, “System Tasks,”](#) on page 43 for more information.

System tasks are referred to by a task name or an abbreviation of the name of the primary module that comprises the task function. For example as [Table 2 on page 4](#) shows, the communications task, whose primary module is DMTREX, is also called the REX task. In this document, the task name and the module abbreviation describes a task.

Table 2. RSCS System Tasks and Functions

Task Name	Primary Module	Function
Communications task	DMTREX	Initializes RSCS, creates other system tasks, and processes commands.
Spool manager task	DMTAXM	Manages the RSCS virtual reader, maintains data structures that describe the file queues, and tells link drivers about files.
Exec processing task	DMTEXE	Runs GCS execs called by the RSCS EXEC command.
Auto-start task	DMTAST	Manages enabled auto-dial ports for out-going calls, auto-start links, and the Inactivity Time Out (ITO) and RETRY functions. This task is also called the auto-dial task.
Event scheduling task	DMTEVE	Runs specified events or commands at requested times.
SNA control task	DMTSCT	Receives SNA requests and manages the RSCS/VTAM interface.
TCP port redirector task	DMTPRD	Routes TCP/IP connection requests from a host system to a specific TCPNJE-type link driver task.

Figure 2 on [page 5](#) shows an overview of the RSCS initialization process. This process starts when GCS passes an INIT command to the RSCS console input routine (DMTMANEP). If there are no syntax errors in the command, the RSCS communications task (REX) is attached.

The communications task then calls DMTIRX to read the RSCS configuration file and calls Exit 0. If there are no errors in the configuration file and or in any Exit 0 routines, the REX task attaches the spool manager task (AXM), EXEC processing task (EXE), auto-start task (AST), the event scheduling task (EVE), and the TCP port redirector task (PRD). After each system task has initialized successfully, the RSCS virtual machine comes up. The optional SNA control task (SCT) is attached when a NETWORK START command is issued. However, SNA-type links cannot be started before the SCT task completes the initialization of the RSCS/VTAM interface.

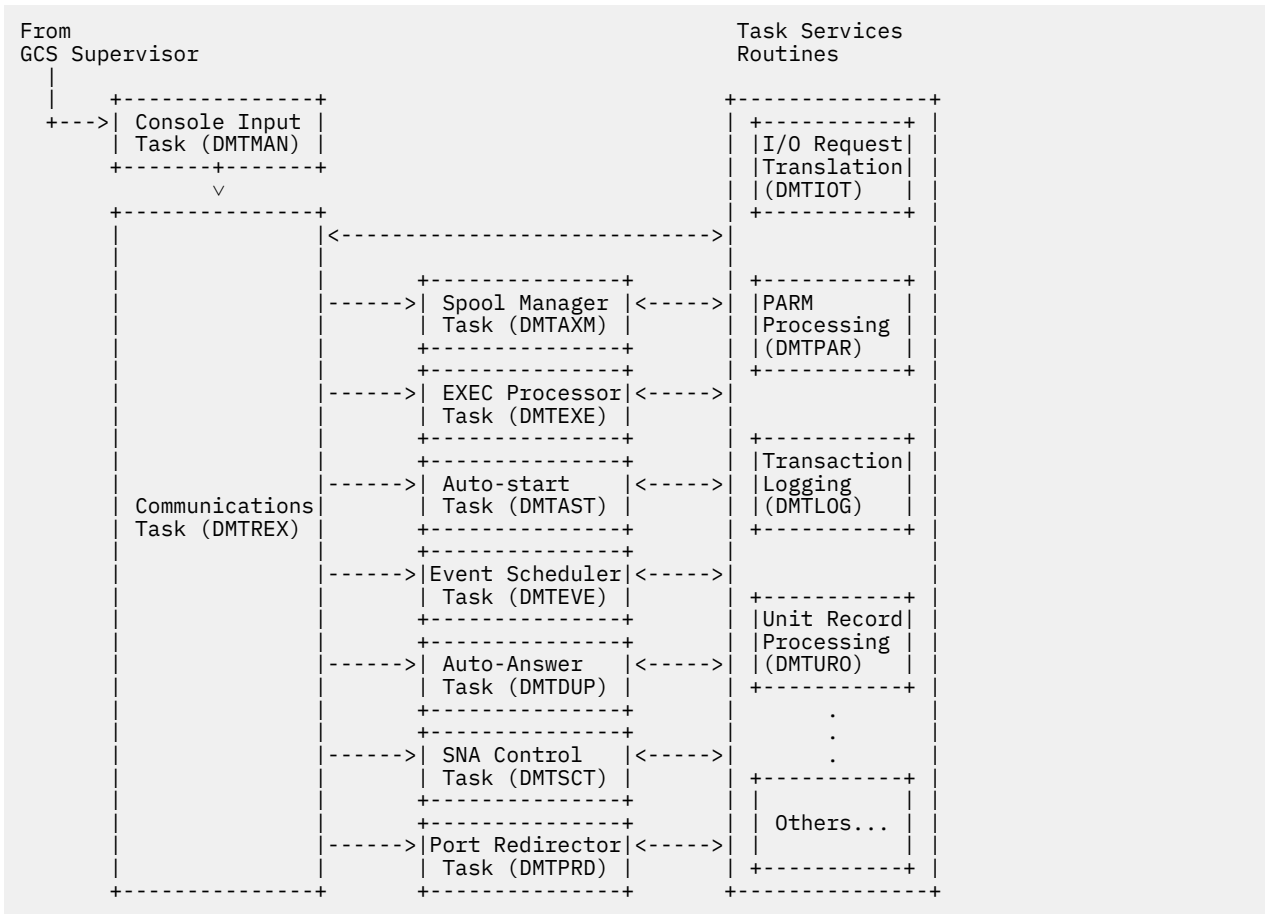


Figure 2. Overview of RSCS Initialization and Task Creation

Link Driver Tasks

The term *link driver* describes all tasks that establish a link between the local RSCS virtual machine and another node, printer, or service. There are two types of link driver tasks: network link driver tasks and session (or SNA) driver tasks.

The LINKDEFINE statement defines links to the RSCS virtual machine. See [z/VM: RSCS Networking Planning and Configuration](#) for information about defining links and using configuration file statements.

Link Driver Tasks

The term *link driver* describes any non-SNA links in RSCS. Table 3 on page 5 describes the following information for RSCS link driver tasks: link type, primary module, and associated node.

Table 3. Link Driver Tasks and Associated Nodes

Link Type	Primary Module	Associated Node
ASCII	DMTAPT	Local or remote ASCII printer or plotter
GATEWAY	DMTGPI	Any peer node (for example, a CMS server or remote system)
LISTPROC	DMTLIS	None
LPD	DMTLPD	Remote LPR client in a TCP/IP network
LPR	DMTLPR	A line printer daemon within a TCP/IP network
MRJE	DMTSML	Remote multi-leaving workstations

Table 3. Link Driver Tasks and Associated Nodes (continued)

Link Type	Primary Module	Associated Node
NJE	DMTNET	Remote NJE peer systems
NOTIFY	DMTNOT	None
RJE	DMTNPT	Remote workstations
TCPASCII	DMTTAP	ASCII printer or plotter within a TCP/IP network
TCPNJE	DMTTNE	Remote NJE peer system within a TCP/IP network
TN3270E	DMTTPT	TCP/IP attached 3270 printers
UFT	DMTUFT	An unsolicited File Transfer daemon in a TCP/IP network
UFTD	DMTUFD	A UFT client in a TCP/IP network
3270P	DMTRPT	Local or remote 3270 printers

Link driver tasks are attached when a START command is issued or when RSCS receives a call on an enabled auto-answer port. NOTIFY-type and LISTPROC-type links, which do not need dedicated line addresses, are only attached when a START command is issued for the link.

Session Driver Tasks

In RSCS, the term *session driver* describes all SNA-type links (see Table 4 on page 6). Session driver tasks are attached when a START command is issued for an SNA-type link. They are also attached when RSCS receives an SNA request from a remote node for a logical unit (LU) driven by an RSCS session driver.

Table 4. Session Driver Tasks and Associated Nodes

Driver Name	Primary Module	Associated Node
SNANJE	DMTSNE	Remote peer systems
SNARJE	DMTSJE	Remote RJE workstation
SNA3270P	DMTSPT	Remote or local 3270 printers

Auto-Answer Task

The RSCS auto-answer task monitors enabled auto-answer ports. DMTDUP is the primary module for the task. An auto-answer task (DUP) is attached each time the RSCS ENABLE command is issued for an auto-answer port. Several auto-answer tasks may be present in the RSCS virtual machine at the same time.

When an auto-answer port receives a call from a remote system, the auto-answer task on the local node establishes a connection and then analyzes the sign-on record it receives from the caller. The sign-on card identifies the calling system and identifies the type of link driver (NJE, MRJE, or RJE) requested for the connection. If a matching link has been defined on the local node, the auto-answer task transforms itself into the link driver task for the specified link.

Task Interaction

This section describes an overview of how RSCS tasks work together in the RSCS virtual machine to ensure the receipt and delivery of files, commands, and messages in the network.

Processing Files

RSCS's main responsibility is to receive and deliver files to the destination specified by the file originator. In the RSCS virtual machine, these responsibilities include:

- Sending files from local users to remote users
- Receiving files from remote users to local users
- Sending files from remote users to other remote users (store-and-forward handling)

To RSCS, a *remote user* is any user, workstation, or printer to which RSCS can communicate through a link.

Sending Files to Remote Users

When you send a file to a user at a remote node, you create a spool file that is spooled to the RSCS virtual reader. CP generates an interrupt to tell the RSCS virtual machine when the file arrives. GCS, which provides interrupt handling facilities for RSCS, receives this interrupt and notifies the spool manager task (AXM) about the file.

The AXM task queues the file on any link that is eligible to send it. If an eligible link is idle, the AXM task notifies that link driver about the file, and the link driver task takes control of the file.

The link driver task selects the file to be transmitted, opens it, and builds transmission buffers according to the file's contents and link protocol. When the end of the file is reached as it is read from spool, the link driver will wait for the remote node to acknowledge receipt of the file. When it receives the acknowledgement, the link driver then closes and purges the file from the local node.

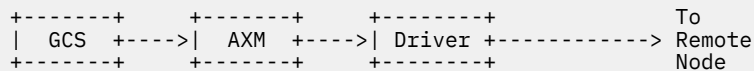


Figure 3. Sending a File to a Remote Node

Receiving Files from Remote Users

When RSCS receives a file from a remote node, a link driver task is informed of the arrival of teleprocessing (TP) buffers. Non-SNA links are notified by I/O interrupts on the devices they own. SNA-type links are notified when a VTAM RECEIVE request has completed. TCP/IP-type links are notified when a Socket RECV request has completed.

When it detects the beginning of a file, the link driver task calls the AXM task, which defines an output unit record (UR) device. The file data is written to that UR device by write CCWs. When the end of the file is reached, the UR device is spooled to the destination user and the device is closed. If necessary, the link driver sends an acknowledgement to the remote node.

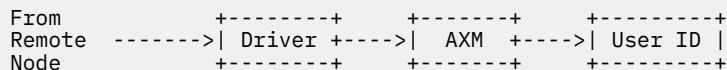


Figure 4. Receiving a File from a Remote Node

Handling Store-and-Forward Files

Files that originate on a remote node and are destined to another remote node are called *store-and-forward files* on an intermediate (local) node. The intermediate node first receives the file from a remote node and then sends it to the next remote node.

When it receives a file, a link driver on the intermediate node opens an output UR device and writes the file to that device. When the end-of-file marker is received, the output device is spooled to the virtual reader of the local RSCS virtual machine. When the device is closed, the file appears in RSCS's virtual reader. The link driver then sends an acknowledgement to the node that sent the file, if necessary.

The spool manager task is notified when the file arrives in the RSCS virtual reader. The AXM task queues the file on a link that is eligible to send it to the next node.

The link driver task selects the file for transmission, opens it, and builds transmission buffers based on the file's contents and link protocol. When it reaches the end of the file, the link driver will wait for the remote node to acknowledge receipt of the file. When it receives this acknowledgement, the link driver closes the file and purges it from the local node.

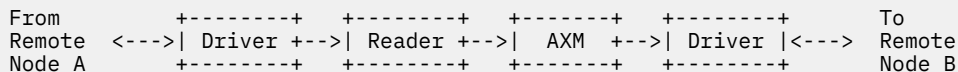


Figure 5. Processing a Store-and-Forward File

Establishing SNA Connections

The RSCS/VTAM interface allows RSCS to establish SNA connections within the network. When a NETWORK START command is run, the REX task attaches the SNA control task (SCT). After it initializes, the SCT task monitors the initialization of the RSCS/VTAM interface. When this process completes, session drivers can be attached.

When a START command is issued for an SNA-type link, the REX task passes the request to the SCT task. The SCT task then runs the VTAM SIMLOGON macro, which causes VTAM to simulate a logon request from the remote node. The local node becomes the primary LU for the SNA session.

When VTAM establishes contact with a remote node, it drives the remote system's logon exit. If the remote system is another RSCS, a session driver task will be attached on that remote RSCS. The session driver task then issues a VTAM OPNDST request. This request establishes the RSCS connection allowing data to be processed over the link. SNANJE-type links must also exchange NJE sign-on records. Printer and workstation links (SNA3270P-type and SNARJE-type) can send data when the SNA session is established.

If no problems are found, the SNA control task becomes idle. If a SNA 3270 printer is shared by multiple systems, VTAM can issue a release request (RELREQ). This request tells RSCS that another SNA application needs the printer. However, if files are enqueued on the session driver, RSCS does not release the printer. If the printer is no longer needed on the local node, the SCT task deactivates the printer session; the session driver task remains active. If the printer is needed later, the session driver task runs the VTAM SIMLOGON macro to ask VTAM to reacquire the printer for the local RSCS virtual machine.

For SNANJE-type links, a START command does not have to be issued on both ends of the session. A VTAM operator can enter a VARY LOGON command to begin a session. If an operator on a remote node enters a START command for an SNANJE-type link on the local node, VTAM drives the SCIP exit in the local RSCS virtual machine. The session driver task on the local node becomes the SNA secondary LU (SLU). It then issues the VTAM OPNSEC macro to establish the session. Data can flow on the SNA session once sign-on records are exchanged. RSCS can only act as the SLU for SNANJE-type links; workstation and printer links must be the primary LU in an SNA session. See [“SNA Control Task” on page 55](#) for more information.

Establishing Auto-Dial and Auto-Answer Links

Auto-dial and auto-answer links can be established between RSCS virtual machines that do not communicate often with each other. Auto-dial and auto-answer ports must be defined and enabled on each RSCS virtual machine. The RSCS virtual machines must also have a link, with auto-start options, defined between them.

When a file is queued on the link between the RSCS virtual machines, the spool manager task on the local RSCS virtual machine tells the auto-dial task (DUP) that an auto-dial port is needed. The auto-dial task then assigns an available port address to the link and attaches the link driver task. The link driver task initializes and dials the phone number specified on its PARM statement or on the DEFINE or START commands.

When an enabled auto-answer port on the remote node receives the call, the DUP task establishes the connection. The DUP task examines the sign-on record from the calling node. If it finds a correct link definition, the DUP task runs the GCS LINK macro to transform itself into the type of link driver specified by the caller.

After the file is sent and the links become idle, Inactivity Time Out (ITO) definitions for each link determine how long the connection remains up. See [z/VM: RSCS Networking Planning and Configuration](#) for information about ITO definitions.

Processing Commands

RSCS also processes commands that originate from local and remote users. The commands may be run on the local node or sent to another node in the network.

Commands Issued By a Local User for the Local Node

When RSCS initializes, the REX task establishes a connection to the IUCV *MSG service. All SMSG commands issued at the local node are placed on DMTRGX's command queue by an interrupt handler.

If the command is for the local node, the processing routine for the command is called. Command responses are then returned to the command originator. If the command requires another RSCS task to complete work, the command processing routine passes a command element to the appropriate task. The CMNDAREA maps each type of command element (see "CMNDAREA" on page 231). Commands to manipulate files or queues (CHANGE, TRANSFER, REORDER) are passed to the AXM task. Commands, such as FLUSH, TRACE, and DRAIN, are passed to a link driver. The SCHEDULE command is passed to the EVE task; the EXEC command is passed to the EXE task. The TCP command is passed to the PRD task. The network command is passed to the SCT task.

Some commands cause RSCS to attach or delete a task. For example, a START command creates a link driver task and the ENABLE command creates an auto-answer task. The NETWORK START command causes the SNA control task to be attached. The TCP START command causes the TCP/IP Port Redirector to be attached.

Commands Issued to Remote Nodes

To send a command to a remote node, users on the local node must enter the CMD command. The REX task receives the command element from the RSCS interrupt handler and passes it to a command processing routine for processing. This processing routine builds a command element and queues it on the appropriate link driver. The link driver task dequeues the element and sends it to the remote node.

Commands Issued to the Local Node

When a user on a remote node enters a command for the local RSCS virtual machine, a link driver task is informed of the arrival of a TP buffer. When it finds the command element in the TP buffer, the link driver task calls DMTRGX to determine how the command should be processed. DMTRGX then notifies the REX task about the command and places the command element on its command queue.

Processing continues as if the command originated from a local user. However, RSCS returns any command responses to the command originator over an appropriate link to the node.

Commands Issued by a Remote User for a Remote Node

Like files, commands that originate from one remote node can be destined to another remote node. The local (intermediate) node does not process the command element. After a link driver task receives the command element, it calls DMTRGX. As Figure 6 on page 9 shows, DMTRGX then queues the element onto the next node's link driver task. This link driver then removes the command element from its queue, places it in a TP buffer, and sends the element to the next remote node.

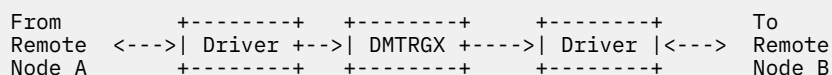


Figure 6. Processing a Store-and-Forward Command

Processing Messages

In RSCS, message processing is similar to command processing. However, when a remote user sends a message to a local user, the link driver task that receives the message element also delivers the message to the user. The REX task does not process the message. See [Chapter 12, “Message Processing,” on page 127](#) for more information.

Chapter 2. RSCS Structure

This chapter describes the overall structure of RSCS communication and data flow. The first section describes how RSCS tasks communicate and share information. The second part describes some of the data areas that store the information RSCS needs to successfully complete a task.

Communication Between Tasks

This section provides an overview of task communication within RSCS. Because few RSCS functions are completed by a single task, inter-task communications are an important part of RSCS processing. For more information, see [Chapter 4, “Inter-Task Communication,”](#) on page 39.

The register savearea convention that RSCS follows is shown in the following figure.

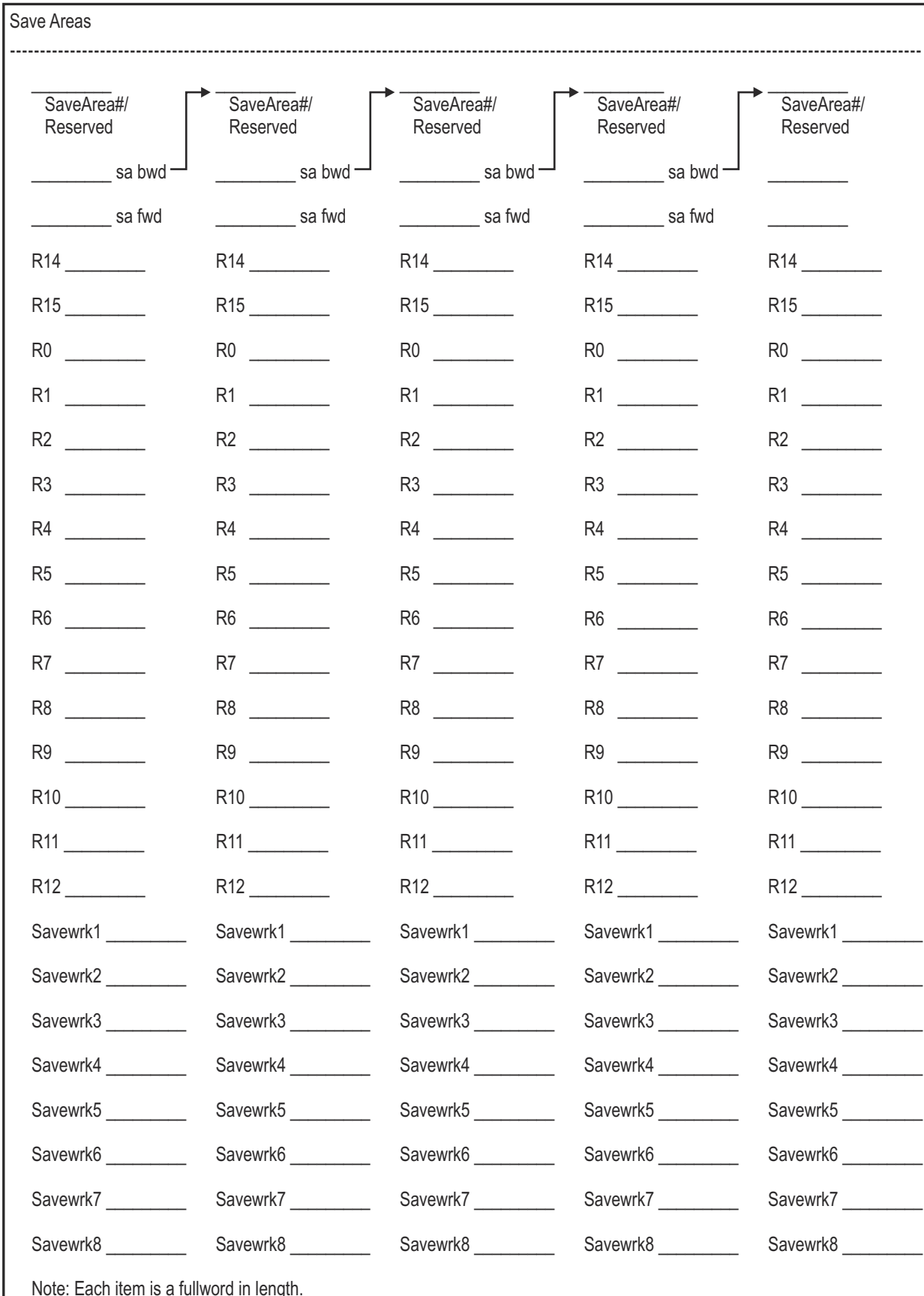


Figure 7. Register Savearea Convention

Direct Task Interfaces

Figure 8 on page 13 shows a direct interface between two synchronized tasks. Task 1 sends a request to Task 2, which then acts on that request. Task 1 cannot make other requests until Task 2 completes the first request.

```

+-----+      +-----+
| Task 1 +----->| Task 2 |
+-----+      +-----+

```

Figure 8. Direct Interface Between Tasks

REX Task Command Interface

When a non-GCS command is entered at the GCS console, GCS uses information from the GCS LOADCMD command to determine which application should process the command. GCS determines RSCS should process the task due to the use of the "RSCS" command prefix, passing the command to the RSCS console task, DMTMANEP.

As Figure 9 on page 13 shows, DMTMANEP then passes the command to the communications task, which is responsible for processing the command. The REX task notifies DMTMAN when the processing completes. DMTMAN then passes a return code from the command back to GCS.

```

+-----+
| Command |
+-----+
| DMTMANEP | <-----> | REX |
+-----+      +-----+

```

Figure 9. Command Processing Interface

STOP Command Interface to Link Drivers

The LTERECB field in a link's LINKTABL indicates if the link driver task should end. Link driver tasks check this field each time a GCA WAIT macro call has been satisfied. The REX task and VTAM TPEND exit usually post the LTERECB for link and session drivers, respectively. Other system tasks can also post this ECB. When its LTERECB ECB is posted, the link driver task stops processing as soon as possible and terminates, returning control to GCS.

```

+-----+      +-----+
| Tasks +----->| Drivers |
+-----+      +-----+

```

Figure 10. Stop Command ECB Interface

VTAM Event Interface to Session Drivers

The SNA control task (SCT) detects several VTAM events (for example, SEND and RECEIVE requests) that affect RSCS-SNA sessions. VTAM informs the SCT task of these events by posting an ECB or by scheduling a VTAM exit routine in DMTVXT.

The SCT task then posts an ECB in the session driver task's LINKTABL to notify the session driver of the VTAM event. If VTAM passes new BIND information to RSCS, the SCT task also passes this information to the session driver.

```

+-----+      +-----+
| Tasks +----->| Drivers |
+-----+      +-----+

```

Figure 11. VTAM Event Interface

Queued Command Interfaces

Figure 12 on page 14 shows a queued interface between tasks that are not synchronized. For this type of communication, a task places its requests on another task's input queue. Each request is represented by a queue element, which can be up to 256 bytes long.

In this example, Task 2 reads requests from its input queue and acts on them after Task 1 has made its request. Other tasks may place requests in the queue before Task 2 has acted on the first request. These additional requests are held in the input queue until Task 2 completes the first request.

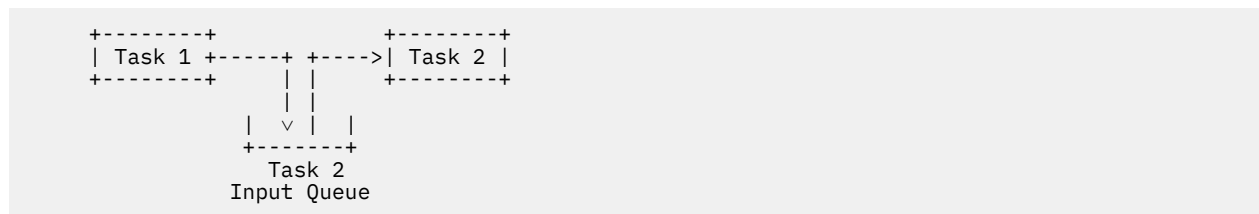


Figure 12. Queued Task Interface

The RSCS routines, DMTCOMNQ and DMTCOMDQ, can access all of the queue interfaces within RSCS. Tasks call DMTCOMNQ to place elements on other task queues; they call DMTCOMDQ to remove elements from their own input queue.

The task that called DMTCOMNQ then posts the ECB associated with the specific queue for the target task. When the target task determines that its ECB is posted, it calls DMTCOMDQ to receive the element. This process continues until DMTCOMDQ indicates that no elements remain on the task's input queue; the target task then clears its queue ECB. For more information, see [“DMTCOMNQ and DMTCOMDQ” on page 41](#).

REX Task IUCV and Command Queue

The REX task receives commands from several sources. The command elements are in the L3 format of CMNDAREA. The REX task then passes them to the main command processor module, DMTCMX.

DMTAXMEP

START commands when links are to be auto-started.

DMTREXIU

Commands when data from SMSG commands is received on the IUCV session connected to the *MSG system service.

DMTMANEX

ENABLE commands when auto-answer ports must be re-enabled.

Link Drivers

Commands from workstations or command nodal message records (NMRs) from remote nodes.

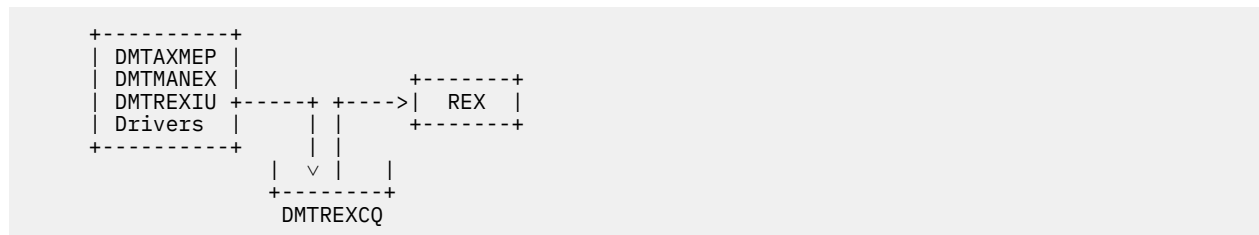


Figure 13. REX Task Command Queue

System Task Command Queue

Each RSCS system task has one or more input command queues. Most of the input to the command queues comes from other system tasks.

EXE Command Queue

Command elements are enqueued to the EXEC processor task (EXE) when a link must be restarted or when an exec is issued. As [Figure 14 on page 15](#) shows, the EXE task receives the command elements from DMTCMZ and the end of task routine, DMTMANEX.

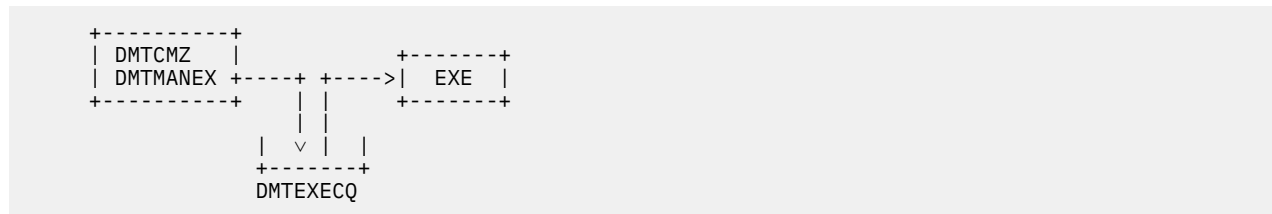


Figure 14. EXE Task Command Queue

EVE Task Command Queue

The event scheduler task, EVE, receives the processed form of SCHEDULE commands from DMTCMB. Exit routines, running in any RSCS task, can also make requests for schedule events.

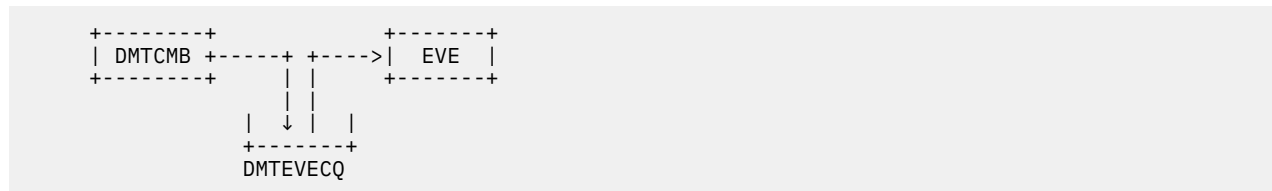


Figure 15. EVE Task Command Queue

Link Driver Command and Message Queues

RSCS converts all link driver commands (for example, HOLD, FLUSH, and FWDSPACE) to command elements and places them on the command queue (LCMN) of the appropriate link driver. The LINKTABL entry of each link driver contains its command queue anchor. For non-NJE link drivers, the routing element of the CMD command is also placed in the command queue.

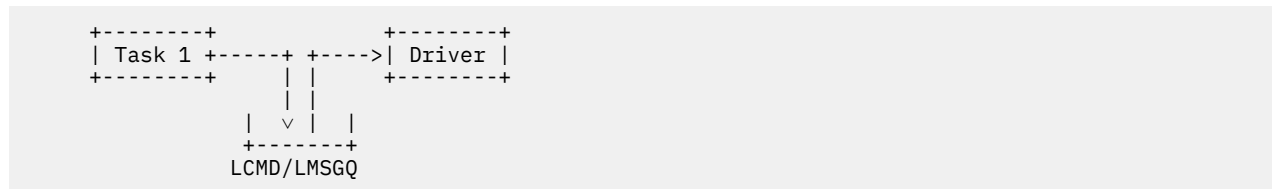


Figure 16. Command and Message Element Queue to Link Drivers

RSCS converts all nodal message records (NMRs) it receives from NJE nodes for which it must forward to other nodes into *routing elements*. These elements, which are in the L3 format of the CMNDAREA macro, are placed on the message queue (LMSGQ) of the appropriate link driver task. The LINKTABL entry for each link driver contains the anchor for the message queue.

If a command or message is routed to another node, the command processing and message-issuing modules also generate routing elements. These elements are placed in the link driver's message queue.

Data Structures

This section describes the primary data structures RSCS uses to perform various tasks. Some data areas represent an external feature of the RSCS network (for example, each defined link has a LINKTABL entry). Other structures are important parts of the processing of each RSCS task.

Primary Data Areas

The communications vector table (CVT) and the common routines vector (CRV), two of the primary data areas, point to other important data areas and routines within RSCS. Other RSCS tasks and user exit routines use these data structures to find information about the RSCS virtual machine.

Communications Vector Table

The CVT contains pointers to other RSCS data areas, which are described in the following table. Given the location of the CVT, tasks can locate other RSCS data structures or routines. Therefore, RSCS passes the address of the CVT to every exit routine (see *z/VM: RSCS Networking Exit Customization* for more information). For more information about the CVT format, see “CVT” on page 202.

Table 5. Communications Vector Table

Data Structure	Anchor Field	Function
LINKTABL	TLINKS	The first LINKTABL entry represents the local node; each LINKTABL entry that follows represents a link.
ROUTEGRP	TROUTEGRP	Represents a combination of routed nodes.
PORT	TPORTS	Represents an address of a switched telecommunications line or binary synchronous communication (BSC) connection to a remote node.
TAGAREA	TTAGQ	Contains pointers to all the file related data structures in RSCS.
AUTHBLOK	TAUTH	Contains the authorization level of a node ID and user ID that can use privileged commands.
REROUTE	TREROUTE	Contains information about how files, commands, and messages are rerouted to a specific node ID and user ID.
DEST	TDEST	Identifies a printer under the control of the Printer Service Facility (PSF/VM) on the local node.
EXITBLOK	TEXITS	Points to a vector of 256 addresses; each pointer is an anchor for a table of contiguous EXITBLOKs representing exit points 0 through 256.
MONITOR	TMONITOR, TMONIMSG	Monitors the command settings of the global SET command (TMONITOR) and the SETMSG command (TMONIMSG).
FORM	TFORMTAB	Represents a print form name and its defined characteristics.
RFCBTAB	TFCBTABA	Defines a forms control buffer (FCB) image to RSCS.
CRV	TCRVTAB	Contains the addresses of common RSCS routines.
EVEBLOK	TEVENTS	Represents an event to the event scheduler task.
EQUATE	TEQUATE	Represents the types of link drivers known to the RSCS system tasks.
RESBLOK	TRESOURC	Represents a resource that many RSCS tasks may share.

Table 5. Communications Vector Table (continued)

Data Structure	Anchor Field	Function
TASKBLOK	TTASKTAB	Points to an index vector containing 1024 pointers; each of these pointers may point to one TASKBLOK or a chain of TASKBLOKS.

The CVT also contains counters for the number of active tasks. The CVT also contains flags (TGLOBAIx fields) that indicate the state of the RSCS virtual machine. The default values for MAXDSH and MSGSKIP parameters for the networking link drivers are set in the CVT when the configuration file is processed. The CVT also points to various work areas within RSCS. It also contains the maximum hop count value, which determines when a file is looping in the network. Finally, the CVT contains the count for the maximum number of messages that can be returned in response to a query command.

The CVT also contains a user exit usage field called TUSER. This field can be used to anchor working storage that must be accessible to multiple exit packages. See [z/VM: RSCS Networking Exit Customization](#) for further details.

Common Routines Vector

The CRV contains pointers to RSCS routines, queue anchors, and locks, which are useful to exit routines outside the RSCS load library (see “CRV” on page 201). No RSCS modules reference the CRV. As [Figure 17 on page 17](#) shows, given the address of the CVT, an exit routine can use the TCRVTAB field to find the CRV.

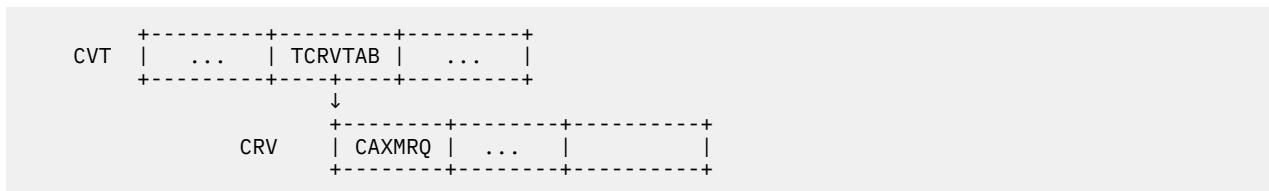


Figure 17. CRV Anchored in CVT

Parameters specified on the RMOD and RCALL macros determine if the exit routines use standard V-constant linkage to call RSCS routines (if the module is link-edited into the RSCS LOADLIB) or if they access routines through the CRV. See [z/VM: RSCS Networking Exit Customization](#) for more information about these macros and the RSCS exit facility.

Defining Network Structure

The structure of the RSCS network is defined by the operands you specify on the LINKDEFINE and ROUTE statements and on the DEFINE, DELETE, START, and ROUTE commands.

This section describes the LINKTABL, NODE, and ROUTEGRP data areas, which describes the RSCS view of the network’s structure. RSCS uses these structures to determine the links on which files or messages can be transmitted, or routed, to their destination nodes. To route a file or message, a task calls DMTCOMGN to locate the following information:

- The NODE entry corresponding to the specified node
- The ROUTEGRP entry for the node, by following the NODERGRP pointer
- If the ROUTEGRP is part of another group, its root group is located by the ROUTGDAD pointer.
- The primary and any alternate links, using the ROUTLNKS and ROUTALNK pointers.

LINKTABL

A link table (LINKTABL) entry describes any RSCS connection to a peer node, printer, or workstation created by a LINKDEFINE statement or by the DEFINE and START commands. Each link defined to the RSCS virtual machine has a LINKTABL entry. The TLINKS field in the CVT points to the chain of LINKTABL

entries. The first LINKTABL entry in the chain contains information about the local RSCS node. The LINKID field in this LINKTABL contains the node ID of the local node.

Most LINKTABL fields correspond to the operands of the LINKDEFINE statement and the DEFINE or START commands. Fields that represent operands of the LINKDEFINE statement and START and DEFINE commands have two forms:

- LACTxxxx fields for the active, or current, value of a field
- LDEFxxxx fields for the default values specified on a LINKDEFINE statement or on the DEFINE command.

The LACTxxxx fields contain copies of the values in the LDEFxxxx fields when the link is inactive. If the link is started by a START command that specified PARM or OPARM values, only the values in the LACTxxxx fields are changed; they are restored when the link becomes idle.

All PARM, OPARM, or UPARM parameters are represented by a pointer in the LINKTABL. This field points to a data area that contains a half-word header, indicating the number of characters in the parameter string, and the text of the string. If no parameters are specified on the link, this pointer contains zeros.

The LINKTABL contains LFLAG fields indicating the link's status. It also contains ECBs and queue anchors for commands and work queues for active links. The LINKTABL anchors and keeps counts for TASHADOW queues that represent inactive files on the link. Other LINKTABL fields govern multistreaming, link message subscriptions, SNA and TCP/IP sessions, and the node and user ID of the START command originator. The LINKTABL also contains a pointer to the ITRACE settings used for the link task. For more information about the LINKTABL format, see [“LINKTABL” on page 205](#).

When a link is deleted, its LINKTABL entry remains in the LINKTABL chain and its LINKID field is cleared. When a new link is defined, RSCS searches the LINKTABL chain to find a LINKTABL entry containing an blank LINKID field. If it finds one, RSCS uses the existing structure to create the LINKTABL entry for the new link. If no links have been deleted (there are no cleared LINKID fields), RSCS creates a new LINKTABL entry for the new link.

When RSCS initializes, all LINKTABL entries are added to a hash table based on their link ID. Tasks call DMTCOMLK to locate LINKTABL entries in the hash table (see [“DMTCOMLK and DMTCOMGG” on page 109](#) for more information on DMTCOMLK).

ROUTEGRP

A route group (ROUTEGRP) represents a group of nodes or a collection of groups. ROUTEGRP entries define the routes within the network. They also contain counters for the number and types of files enqueued on the nodes. (For more information on the ROUTEGRP control block, see [“ROUTEGRP” on page 211](#).) The following terms describe the routing structure with an RSCS network:

Root (top-level) group

A ROUTEGRP entry that is directly routed to a collection of links. A root ROUTEGRP is not *owned* by any other ROUTEGRP.

Bottom-level group

A ROUTEGRP entry that does not own other ROUTEGRP entries. This group entry can contain zero or more nodes.

Honorary group

A ROUTEGRP entry that represents a loosely related set of nodes that are routed through the same combination of primary and alternate links. Honorary ROUTEGRP entries are always both root groups and bottom-level groups. Honorary groups are identified by the ROUTHONR flag.

Real group

Any ROUTEGRP that is not a honorary group. It is the subject or object of a ROUTE statement.

Parent group

Any ROUTEGRP that owns another (child) group.

Child group

Any ROUTEGRP that is owned by another (parent) group. The ROUTGDAD field points to the parent ROUTEGRP entry.

RSCS creates routing groups from the options you specify on the ROUTE command and configuration file statements. For example, if you specify the following configuration file statements, you create the ROUTEGRP hierarchy shown in [Figure 18 on page 19](#) within the RSCS network.

```
ROUTE GROUP COUNTRY TO LINKA LINKB
ROUTE GROUP WEST TO GROUP COUNTRY
ROUTE GROUP EAST TO GROUP COUNTRY
ROUTE GROUP SITE1 TO GROUP WEST
ROUTE GROUP SITE2 TO GROUP WEST
ROUTE GROUP SITE3 TO GROUP EAST
ROUTE GROUP SITE4 TO GROUP EAST
ROUTE NODE1 TO GROUP SITE1
ROUTE NODE2 TO GROUP SITE2
ROUTE NODE3 TO GROUP SITE3
ROUTE NODE4 TO GROUP SITE4
```

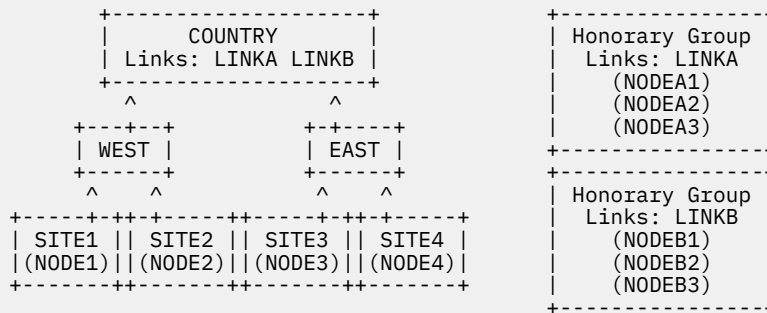


Figure 18. Sample ROUTEGRP Hierarchy

A group can also own links and other groups. In the preceding example, group EAST could contain group SITE1 and SITE2 while owning any number of nodes.

In root group entries, several ROUTEGRP fields contain important information about links:

ROUTLNKS

Points to a vector of pointers to primary links (see [Figure 19 on page 19](#)). The LINKTABL addresses are listed in alphabetic order by link ID.

ROUTLNUM

Contains the number of primary links in the vector.

ROUTALNK

Points to the LINKTABL for an alternate link; contain zero if none was defined.

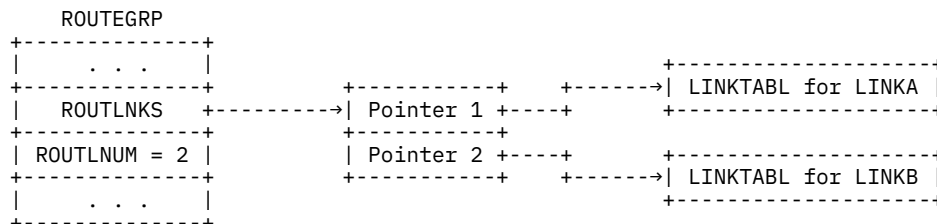


Figure 19. Close-up of COUNTRY ROUTEGRP Entry

ROUTEGRP entries can be accessed in several ways:

- Directly by name, using DMTCOMGG; the ROUTHASH field is the hash chain pointer.
- As a member of their owning group. Child groups owned by a parent group are part of doubly-linked chain linked by the ROUTGNXT and ROUTGPRV fields. This chain is anchored at the ROUTGRPA field of the parent ROUTEGRP.
- By scanning all groups. Groups are part of a doubly-linked chain linked by the ROUTNEXT and ROUTPREV fields. This chain is anchored at the TROUTEGRP field in the CVT.
- As the owner of a child group, using the ROUTGDAD field

- As the owner of a node, using the NODEGRP field as a pointer.

NODE

A NODE entry represents each node defined to RSCS by a ROUTE statement or command. RSCS also creates NODE entries to represent nodes for each link ID and group name. RSCS generally routes these nodes to the matching link or group.

The NODENAME field identifies the name of the node. This field may contain a generic name ending with an asterisk (*). If RSCS cannot find an exact match for a node name, it searches for the NODE entry whose NODENAME entry generically matches the target node name.

As [Figure 20 on page 20](#) shows, NODE entries are chained off their owning ROUTEGRP entry; the NODERGRP field points to the ROUTEGRP entry. Although there are no global NODE chains, tasks can find NODE entries in three ways:

- Directly by name; tasks call DMTCOMGN to locate a NODE entry and a pointer to its root ROUTEGRP entry. (For more information, see [“DMTCOMGN” on page 109.](#))
- As a member of their owning group; NODE entries owned by a group form a doubly-linked chain which is linked by the NODENEXT and NODEPREV fields. The chain is anchored by ROUTNODA in the owning ROUTEGRP (see [Figure 20 on page 20](#)).
- Scanning through all NODE entries.

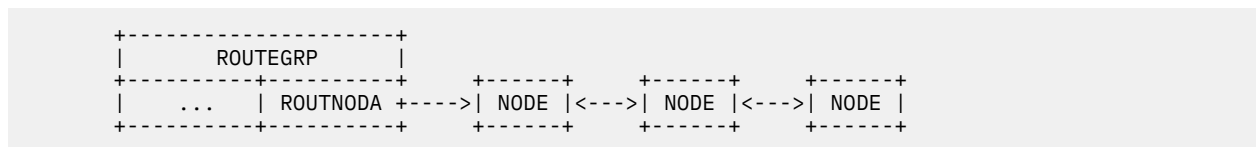


Figure 20. NODE Entries Chained to the Owning ROUTEGRP

REROUTE

RSCS creates a REROUTE element for each REROUTE statement or command that creates a new reroute. REROUTE elements are doubly-linked in a global chain from the TREROUTE field of the CVT. (For more information on the REROUTE control block, see [“REROUTE” on page 210.](#)) REROUTE elements are removed from this chain when the OFF operand is specified on the REROUTE command.

The reroute scanning routine, DMTRER, uses this data structure to determine which REROUTE entry corresponds to a transaction. Each REROUTE element contains the following information:

- Original destination node and user ID of the data to be rerouted (corresponds to the FOR operand of the REROUTE command and statement)
- The new destination node (TO operand)
- The type of data being rerouted (files, messages, or commands)
- Indication if a message should be issued to the file originator (QUIET operand).

REROUTE elements for commands are indexed in a hash table by the node specified on the FOR operand; this HASHBLOK is located at DMTRERHC. REROUTE elements for messages and files are indexed by the user ID specified on the FOR operand. In this case, one HASHBLOK represents each FOR node, for which one or more reroutes have been defined; each of these HASHBLOKs is contained in a RERNBLOK.

The RERNBLOK contains the node name and the number of REROUTE elements indexed from its HASHBLOK. RERNBLOKs are indexed by their FOR node by a HASHBLOK at DMTRERHB.

DEST

RSCS creates a DEST entry for each DEST statement in the configuration file. Each DEST entry identifies the name of a PSF printer on the local node. The DEST entries are anchored at the TDEST field of the CVT.

PORT

RSCS creates a PORT entry for each PORT statement or command that specifies the address of a switched telecommunications line for an auto-answer or auto-dial link. When a port becomes active, the PORTLINK field contains the address of the LINKTABL entry for the link that is using the port. The TPORTS field in the CVT anchors the chain of PORT entries. For more information about the format of the PORT area, see [“PORT” on page 210](#).

Processing Files

RSCS's main function is to send files to, and receive files from, remote nodes. The spool manager task (DMTAXM) is primarily responsible for processing files. DMTRDR, DMTURO, and link driver tasks are also involved in this process. The data structures described in this section represent the files as they are processed through the network.

Files go through many stages as they progress through the RSCS network. TAG elements contain information about the file; TASHADOW contain information about files that are enqueued on links. The following terms describe a file's status:

Inactive

The file is waiting to be transmitted; TASHADOW elements, which represent the file, are enqueued on each link that is eligible to send the file. The TAGTOLOC field in the TAG contains the file's destination node.

Active input

RSCS reads the file from spool and begins to send the file on an eligible link.

Active output

As it receives the file, RSCS is in the process of writing the file data into spool.

Inactive and active input files are identified by their local spool ID, which must be between 0 to 9999. Active output files are not identified by their local spool ID. There can be any number of active output files.

TAGAREA

The TAGAREA contains pointers to all the data structures RSCS uses to process file queues (see [“TAGAREA” on page 216](#)). The CVT field TTAGQ points to the TAGAREA. As [Figure 21 on page 22](#) shows, the TAGAREA contains the following fields:

TAGASVEC

Points to a 10,000 word vector that relates file spool IDs to the TAG slots that represent them.

TAGASLOT

Anchors a doubly-linked chain that contains all the TAG elements in use.

TAGACIN

Anchors the chain of TAG elements for all active input files (files read from spool).

TAGACOUT

Anchors the chain of TAG elements for all active output files (files written to spool).

TAGASLVE, TAGASLVX

Anchor the entry and exit vectors for SLOWDOWN command processing.

The TAGASLOT, TAGACIN, and TAGACOUT chains are doubly-linked by the TAGNEXT and TAGPREV fields in the file's TAG element. The TAGLINK field in TAG elements that are on the TAGACIN and TAGACOUT chains indicates the link on which they are active. The LINKTABL entry does not anchor TAG element chains.

Inactive TAG elements are not enqueued on a link's LINKTABL entry; the TAGLINK field for these elements is not used. Rather, inactive files, which can be enqueued on many inactive links, are represented by TASHADOW elements.

TAGAREA contains two fields, TAGATSTO and TAGASSTO, that point to the TASTORAG storage allocation data structures for the TAG and TASHADOW elements, respectively. The TAGAREA also contains the

number of TAG elements currently in use and a pointer to a vector. This vector locates inactive and active-input TAG elements by their local spool ID.

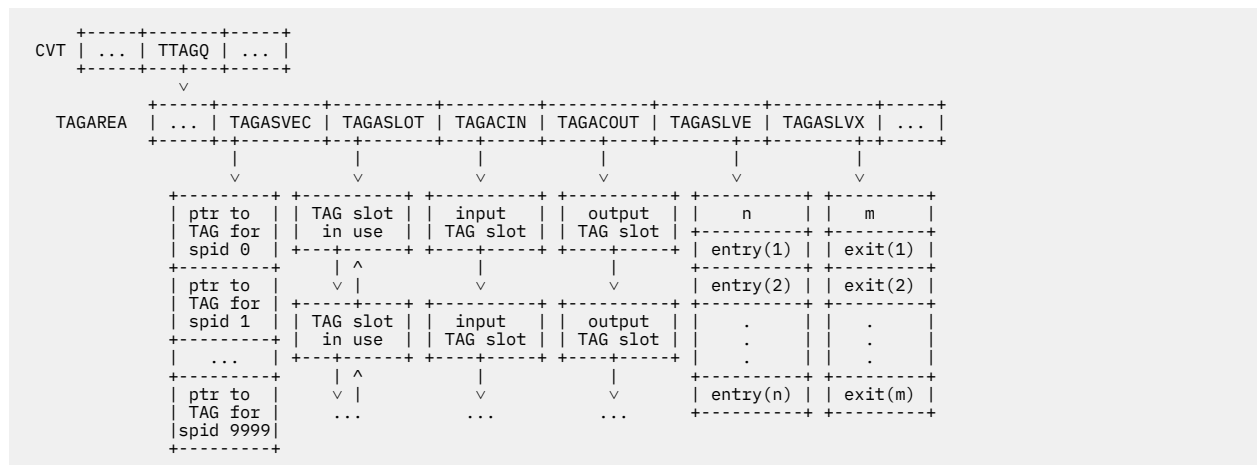


Figure 21. Overview of TAGAREA

TASTORAG

RSCS uses the TASTORAG area to allocate and deallocate the storage needed for TAG and TASHADOW elements (see “TASTORAG” on page 217). There are two TASTORAG areas; the first area, anchored in the TAGATSTO field in the TAGAREA, manages TAG elements. The second area, anchored in the TAGASSTO field in the TAGAREA, manages TASHADOW elements.

Each TASTORAG area (see Figure 22 on page 22) points to a vector that contains the addresses of pieces of available virtual storage. TASTORAG fields also contain the number of pages in each piece of storage and the number of TAG or TASHADOW elements in each page.

The TASTORAG area also contains a pointer, TASBITM, to an allocation bit map, which identifies available virtual storage elements. The number of bits in the bit map equals the number of items for which storage has been allocated.

When an item is allocated, its allocation bit is turned on in the bit map. This bit is turned off when the item is no longer allocated. The byte offset and bit mask needed for this are stored in the items when the chunks are initialized. In the TAG element, this information is stored in the TAGOFFAL and TAGBITMP fields. In a TASHADOW, this information is in the TASOFFAL and TASBTMAP fields.

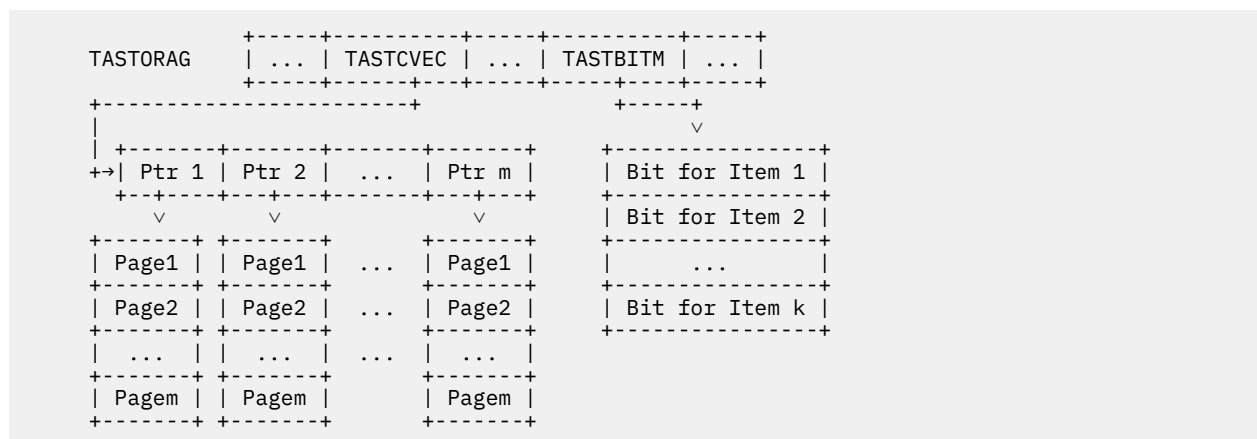


Figure 22. Overview of TASTORAG Allocation

Each page of allocated storage is identified by an eye catcher, which may be used to identify the TAG or TASHADOW elements within a dump. Any free elements in the page of storage are also marked (see Figure 23 on page 23).


```

+-----+
| Eye catcher |
+-----+
|   Item 1   |
+-----+
|   Item 2   |
+-----+
|     ...    |
+-----+
|   Item 1   |
+-----+

```

Figure 23. Overview of a Page of Storage

TAG Element

A TAG element (also called a TAG slot) describes a file's attributes, including: its origin, destination, file size, file name, and file type (see [“TAG” on page 214](#)). During initialization, RSCS acquires storage for 10,000 TAG elements.

When a TAG element is not in use, the corresponding bit in the TAG allocation map in TASTORAG is set to zero. TAG elements that represent files awaiting transmission are kept on a global allocated TAG element queue. This queue is anchored at the TAGASLOT field in the TAGAREA. TAG elements that represent files being transmitted are chained on the active input queue, anchored at TAGACIN in the TAGAREA. The TAG elements for files being written into spool are queued on the active output queue, anchored at TAGACOUT in the TAGAREA.

TASHADOW

A TAG shadow element (TASHADOW) represents a TAG element on each link that is eligible to send a file. Many TASHADOW elements can represent one TAG element. TASHADOW elements only represent inactive files. Each allocated TASHADOW is part of a doubly-linked chain that is anchored at its owning TAG element. Each TASHADOW is also a part of a doubly-linked chain, anchored at the LINKTABL for the link on which it is queued. For more information, see [“TASHADOW” on page 217](#).

TASHADOW elements are identified as *primary* or *alternate* shadows. Primary TASHADOW elements are enqueued on the primary links to the file's destination. They reflect the alphabetic order of the primary links.

Alternate TASHADOW elements represent the file on any alternate link that has been defined to the destination node. The alternate TASHADOW, if present, is the last element in the TASHADOW queue representing the TAG element. If an alternate link is disabled because the primary links are connected, an alternate TASHADOW may be *invisible*.

For example in [Figure 24 on page 24](#), File 1 is represented by TAG 1. This file can be sent on Link A (primary) or Link B (alternate). File 2, which is represented by TAG 2, can be sent on Link B (primary) or Link C (alternate).

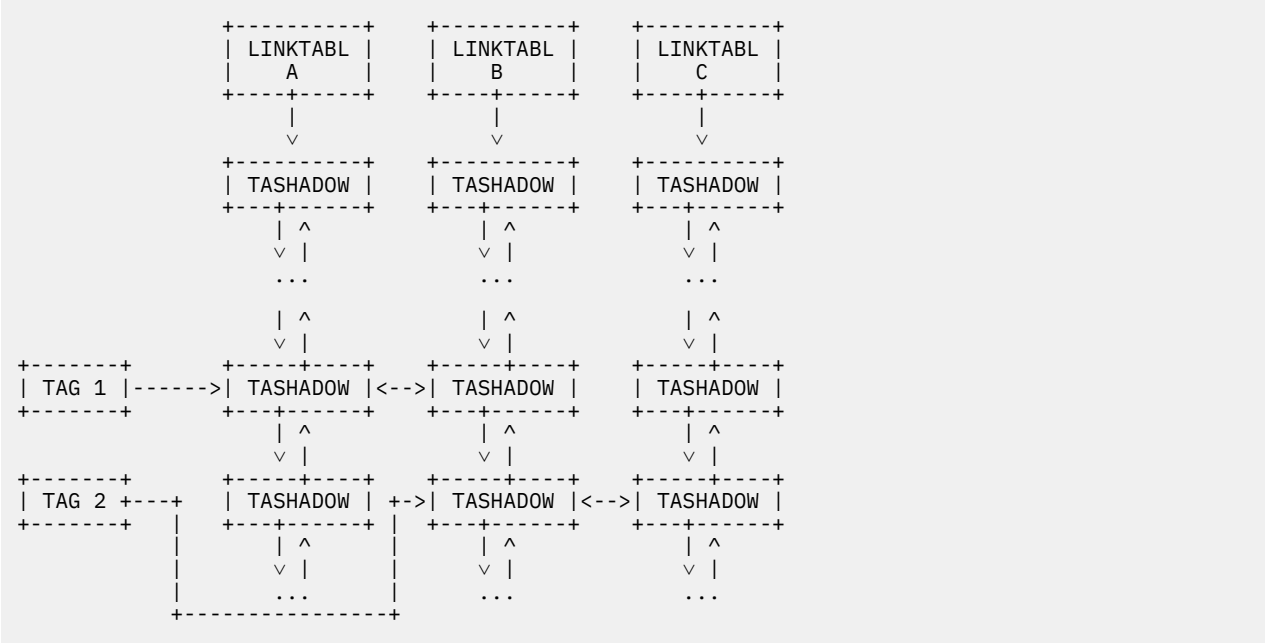


Figure 24. Relationship of TAG, TASHADOW, and LINKTABL Elements

Each TASHADOW element contains information about the file it represents, including: spool ID, a pointer to the LINKTABL, and queuing information. Because this information is available in the TASHADOW elements, RSCS does not process TAG elements each time a file is enqueued on a link or when a link's file queue is reordered.

TASHADOW Queue

RSCS divides a link's TASHADOW queue into 9 equal segments. The LINKTABL contains pointers to the TASHADOW elements that are on the segment boundaries. As Figure 25 on page 24 shows, RSCS inserts a TASHADOW in a file queue by finding the correct segment and location in the shadow queue. RSCS adjusts the boundary pointers each time it adds or removes a TASHADOW element from a link's shadow queue.

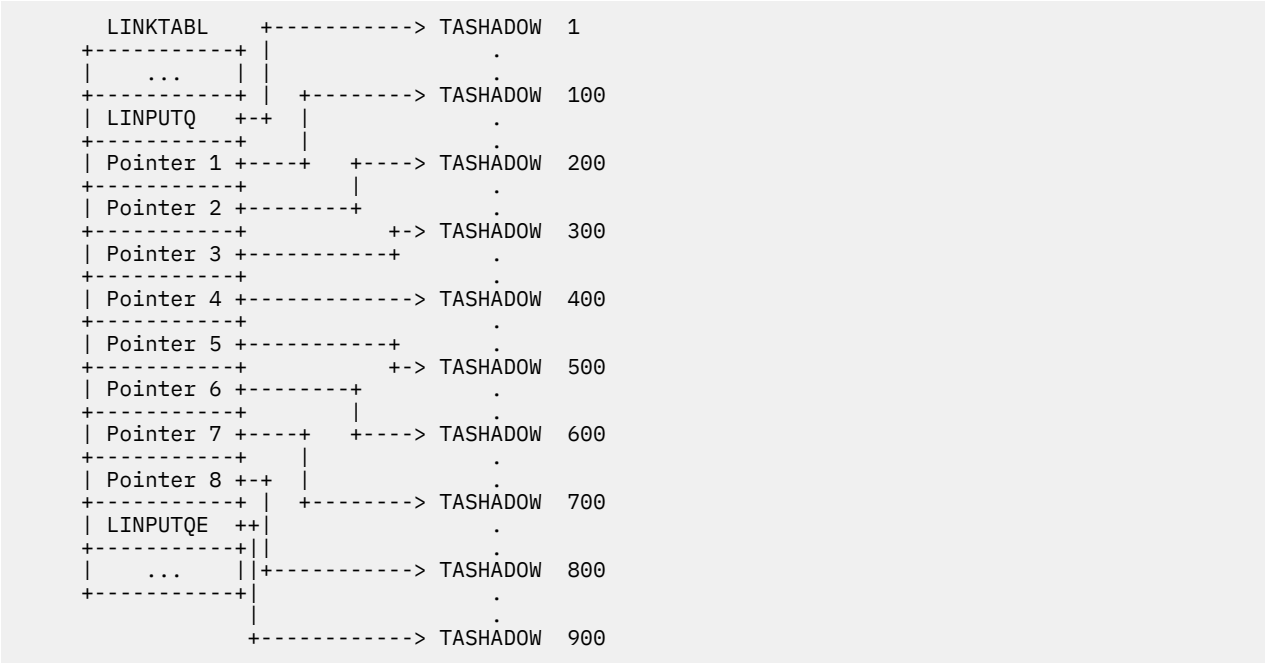


Figure 25. TASHADOW Queue Pointers (900 TASHADOWS on Link)

If a file has been ordered to the top of a link's queue by an ORDER command, its TASHADOW elements remain at the top of the link's TASHADOW queue. If at least one copy of a multiple copy print file has been printed, the TASHADOW elements for the file also retain their order on the queue. When a new file is enqueued on the link, its TASHADOW elements are added to the end of the queue, following the ordered TASHADOW elements.

When RSCS initializes, it reserves 20,000 TASHADOW elements. This number can be modified by the SHADOWS configuration file statement. See *z/VM: RSCS Networking Planning and Configuration* for more information.

If all TASHADOW elements are in use, RSCS tries to obtain more storage for the TASTORAG element that manages TASHADOW storage. If storage is unavailable, RSCS takes the following steps:

- Issues message DMT599W to indicate that RSCS is running in degraded mode.

This message is issued every 100th time RSCS fails to enqueue a TASHADOW element on a link. When a new file arrives in RSCS's virtual reader, RSCS borrows elements from the files that are represented by the most TASHADOW elements. It then uses these borrowed TASHADOW elements to represent the new file.

- When 1,000 TASHADOW elements are available as files are sent or removed from RSCS's reader, RSCS returns 500 TASHADOW elements to the TAG elements that are represented by the least number of shadows and to those that lent the most TASHADOWs.
- When all TASHADOW elements are returned, RSCS issues message DMT598I and returns to its usual processing mode.

Sharing RSCS Resources

Because many tasks run in the RSCS virtual machine, two or more tasks may need to use the same resource at the same time. This section describes the RESBLOK and RESQBLOK data areas, which RSCS uses to serialize the use of its resources. For more information, see [“DMTRES” on page 42](#).

RESBLOK

A resource block (RESBLOK) describes any RSCS resource for which multiple tasks may have to compete. Each RESBLOK contains the name of the resource it manages. All RESBLOKs are in a chain anchored at the TRESOURC field in the CVT.

RESQBLOK

Any RSCS task can use a resource queue block (RESQBLOK) to get the exclusive use of a resource managed by the RSCS resource manager. A task attempts to get a resource by inserting its RESQBLOK at the head of a chain of RESQBLOKs that is anchored at the RESANCH field of the resource's RESBLOK. The task whose RESQBLOK is at the end of this chain owns the resource. All other tasks that have RESQBLOKs on the chain must wait for the owning task to give up control of the resource.

When the owning task stops using the resource, it removes its RESQBLOK from the end of the chain. The owning task then tells the task that now has its RESQBLOK at the end of the chain (if there is one) that the resource is available.

For example in [Figure 26 on page 26](#), Task Z owns the spool resource. When it no longer needs that resource, Task Z removes its RESQBLOK from the chain and tells Task Y that the resource is available. Task Y then owns the resource. Task A owns the ddname resource; however, no other tasks are waiting to use that resource.

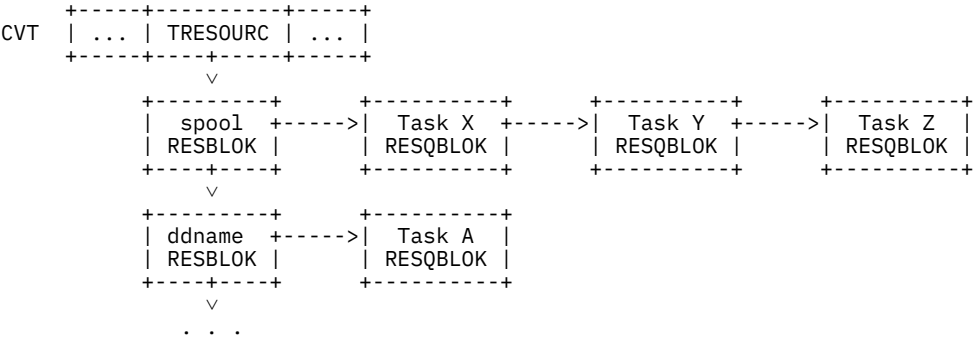


Figure 26. Example of Resource Ownership

Disk File Interface

RSCS uses the file interface routine, DMTCOMFI, to read configuration files and other files from a disk that is accessible to RSCS. This section describes the data areas used as a file is processed. For more information, see “Disk File Interface Routine” on page 109.

FILREQ

The file request block (FILREQ) contains information about the file. It is also used as a master work area to request records from a file. Flags in the FILREQ identify if RSCS accessed the file by a predefined ddname or by its file ID. FILREQ also contains flags that can suppress imbed support and the issuing of error messages.

FILWORKA

RSCS uses a file work area (FILWORKA) to read records from a file. When a file is opened, RSCS creates a FILWORKA and chains it off the FILRAWA field in the FILREQ block. If imbed support is active and RSCS finds a valid IMBED statement in the file, another FILWORKA is queued off the first FILWORKA acquired when the file is opened (see Figure 27 on page 26).

The FILWORKA chain that is anchored on the FILREQ block is a doubly linked list that acts as a stack. The FILREQ area points to the bottom of the stack, while the top of the stack describes the file that is currently processed. The FILWORKA is also maintained in a doubly-linked chain, anchored at the TFILWRKS field in the CVT.

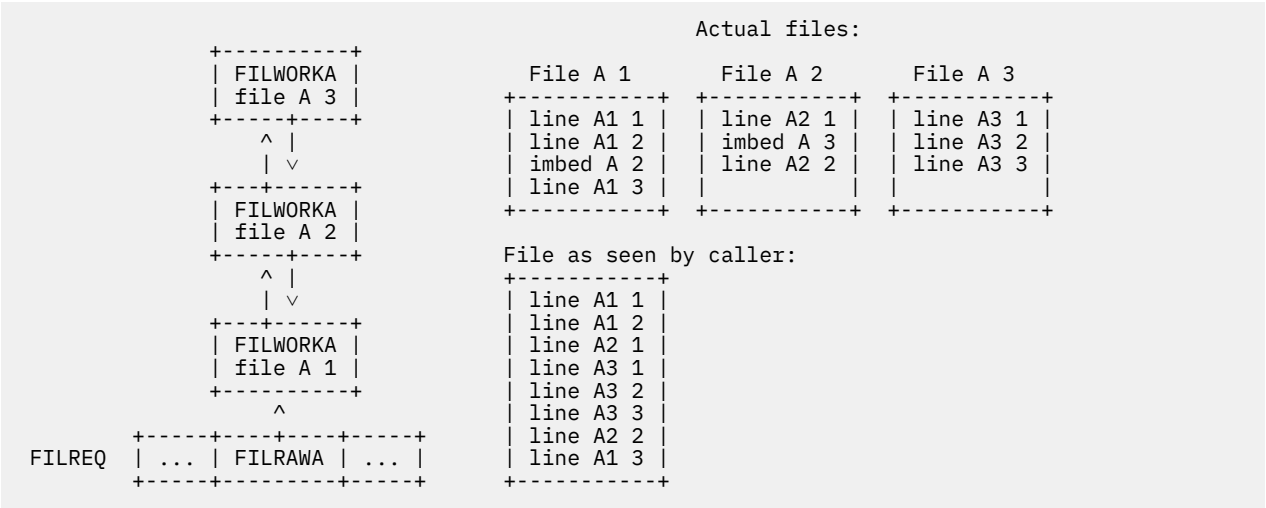


Figure 27. Data Structures as -Line A3 2- is Read

RSCS supports a maximum of 10 FILWORKAs in a stack. If you attempt to imbed more than 10 levels of files, errors will occur and all the files in the stack will be closed.

Dynamic ddname Allocation

If RSCS receives a request to open or imbed a file by its file name and file type, it must define a ddname for the file. To do so, RSCS reserves the ddnames in the range @F000@ to @F999@. An allocation map, pointed to by TDDNMVEC in the CVT, manages the use of these reserved ddnames by multiple tasks. The allocation map contains 1000 bytes of storage; each byte corresponds to a ddname and can have the following settings:

X'FF'

Allocated ddname

X'00'

Available ddname

If all the dynamically defined ddnames are in use when a ddname is needed, an error occurs and all files in that FILREQ's stack are closed.

Defining Tasks

Each system task and link driver task must be defined to RSCS. For correct processing, RSCS must know the task's characteristics (for example, networking or printer link driver). RSCS uses task descriptor blocks and EQUATE entries to identify specific tasks.

TASKBLOK

RSCS maintains a table of task descriptor blocks (TASKBLOKs) about each RSCS task (system, link driver, and auto-answer). A TASKBLOK, which only describes an active task, contains the following information about the task:

- Time it was attached
- GCS task ID
- Name and eye-catcher describing the task
- Type (system, link driver, or auto-answer)
- Pointer to the main control block for the task (SYSIDENT, LINKTABL, or PORT)
- ITRACE settings and a 10-doubleword work area (used by the ITRACE macro to build parameter lists passed to DMTITR).

The TASKBLOKs are maintained by a hash index that is created by DMTTAS. If two or more GCS task IDs create the same index entry, the second and all following TASKBLOKs are chained off a collision pointer in the TASKBLOK.

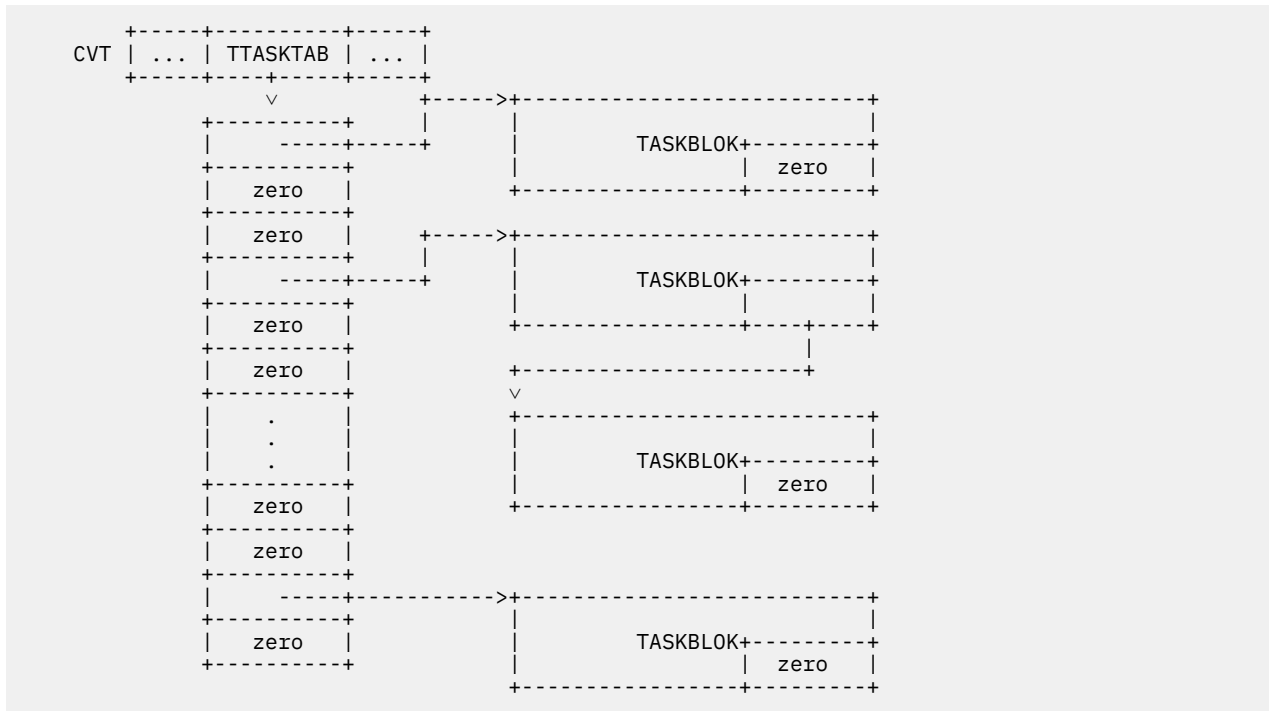


Figure 28. Structure of the RSCS Task Table

System Task Equates

Each type of RSCS system task is represented by an EQUATE entry. The EQUATE table starts with the EQUATE entry for the REX task. EQUATE entries and the GCS IDENTIFY macro identify task names to GCS. They are also used when the REX task calls the ATTACH macro to create the other system tasks.

Link Driver Equates

EQUATE entries describe the characteristic of each link driver. When RSCS initializes, the EQUATE entries for each type of link driver form a chain that is anchored at the TEQUATE field in the CVT. If you define a new link driver with a LINKTYPE statement, its EQUATE entry is inserted at the beginning of this queue.

Each EQUATE entry contains the following information about a link driver:

- Symbolic name
- Entry point that is dispatched when the link driver task is started
- Flags that describe the link driver's characteristics.

The entry point addresses of RSCS-defined link drivers are determined when RSCS initializes. Dynamically created EQUATE entries, those added by LINKTYPE statements, are identified by a flag. Their entry point addresses are determined when the link is started.

Managing Unit Record Devices

The CHANNELS configuration file statement tells RSCS to use unit record (UR) devices on certain virtual channels. RSCS uses UR devices to write files to, and read files, from CP spool.

Channel Table

The channel table anchors the allocation maps for each channel on which RSCS can define UR devices. UR devices can be defined on any virtual channel. The allocation map for each channel contains 256 bytes, one for each of the 256 addresses on the channel. RSCS reuses most types of devices; the byte corresponding to each device in the map may have any of the following settings:

X'FF'

Free address (no device defined)

X'00'

Device address in use

X'F0'

Free RDR device

X'xx'

Free device (device type defined in DEVTYPES copy file).

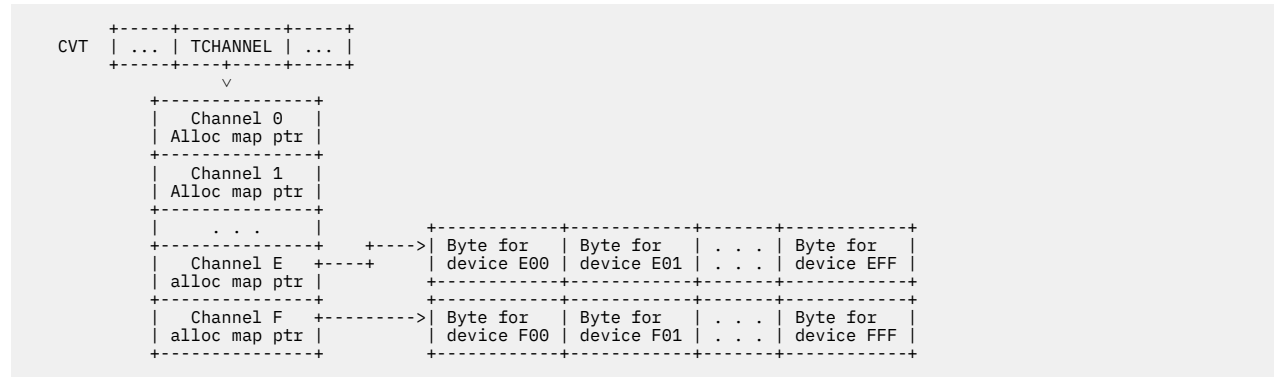


Figure 29. Channels Table Setup with -CHANNELS E F- Specified

Exit Facility

The RSCS exit facility lets you extend and customize RSCS processing without modifying the RSCS source code. Using EXIT statements or commands, you can specify the names of the exit routines called at each exit point. See [z/VM: RSCS Networking Exit Customization](#) for more details on exit points.

EXITBLOK

RSCS creates an EXITBLOK for each entry point specified for an IBM-defined exit point. The EXITBLOK contains the name of the entry point, the address of the exit routine, and flags to indicate if the exit point is active. The TEXITS field in the CVT points at a 256 word vector that accesses the EXITBLOKs for exit points 0 through 255. The EXITBLOKs for each exit point are maintained in a table with the EXBLAST flag on in the last EXITBLOK in the table.

For example, if you specify the following configuration file statements, RSCS creates the EXITBLOK structure shown in [Figure 30 on page 29](#).

```
EXIT 0 PURGEX00 BURSEX00
EXIT 2 PURGEX02
```

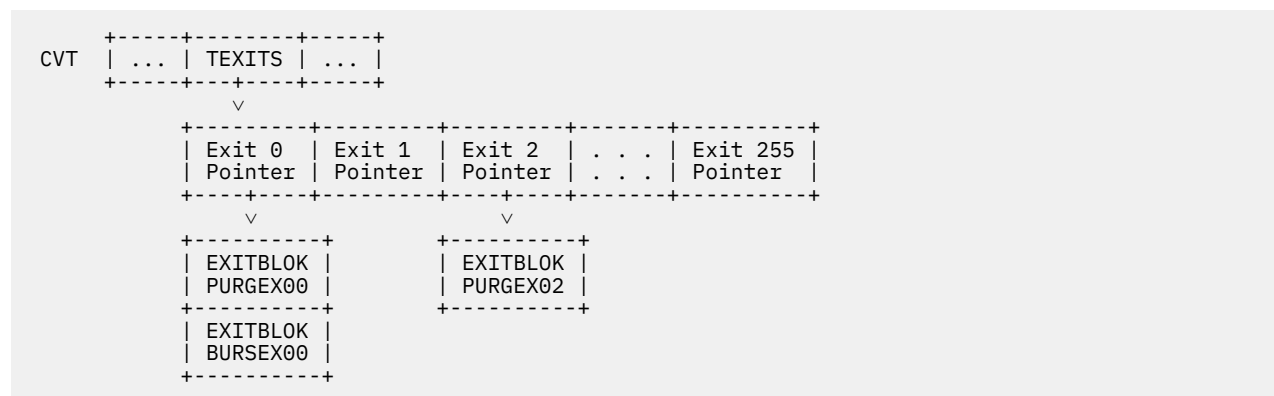


Figure 30. Sample EXITBLOK Structure

Message Subscriptions

You can subscribe node IDs and user IDs to receive certain RSCS messages. If you use the SETMSG command or statement to subscribe to a message, the user receives a copy of that message each time it is issued. If you enter the SET command, the user only receives the message when it is issued to the RSCS console. RSCS uses two versions of the monitor entry (MONITENT) to represent message subscriptions: short version for the SET command and long version for the SETMSG command. See [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#) for more information about message subscriptions.

MONITENT (Short Version)

When a SET command is issued for a message subscription, RSCS creates a short version of the MONITENT. This MONITENT contains the node and user ID of the SET command originator and the node and user ID to receive the subscription. If the subscription is for messages about a link, the MONITENT is added to a chain that is anchored at the LMONITOR field in the LINKTABL for the link. The TMONITOR field in the CVT anchors the MONITENT entries for subscriptions to the RSCS console messages.

MONITENT (Long Version)

RSCS creates a long version of the MONITENT to represent message subscriptions entered by the SETMSG command or statement. In addition to the information in the short MONITENT, the long version contains a 125-byte bit map (1000 bits) that identifies the subscribed message numbers. The chain of long MONITENT entries is anchored at the TMONIMSG field in the CVT.

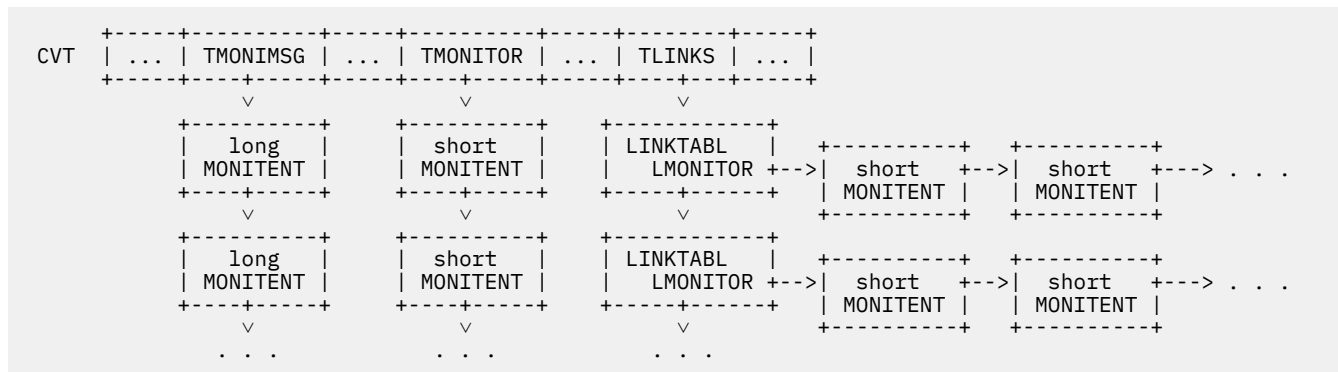


Figure 31. Overview of MONITENT Structure

Message Request and Work Areas

To build a message, RSCS uses a message request block (MSGBLOK), which is passed to the message builder by the task that wants to issue a message. The message building modules also use message work areas (MSGWA) to build the text of a message. See [Chapter 12, “Message Processing,” on page 127](#) for more information about the message building process.

MSGBLOK

A message request block (MSGBLOK), the parameter list passed to the message builder, contains the following information:

- Requested message number
- Destination node and user ID (if any)
- Override routing and severity codes
- Information about any variables to be placed in the message text.

The MSGBLOK also contains information about any Command Response Interface (CRI) options that may have been specified. Also, if the message is issued from a private message repository, the MSGBLOK can contain the pointer to that message repository and a conversion repository.

MSGWA

The message work area (MSGWA) is the basic work area used by the message builder to construct a message. The calling task may supply a MSGWA by placing a pointer to it in the MSGBWA field in the MSGBLOK. If a pointer is not specified, the message builder acquires an MSGWA for the calling task.

The MSGWA contains separate anchors for message header and text issued in the local, network, or private language. It also contains anchors for the text of messages issued in language independent form.

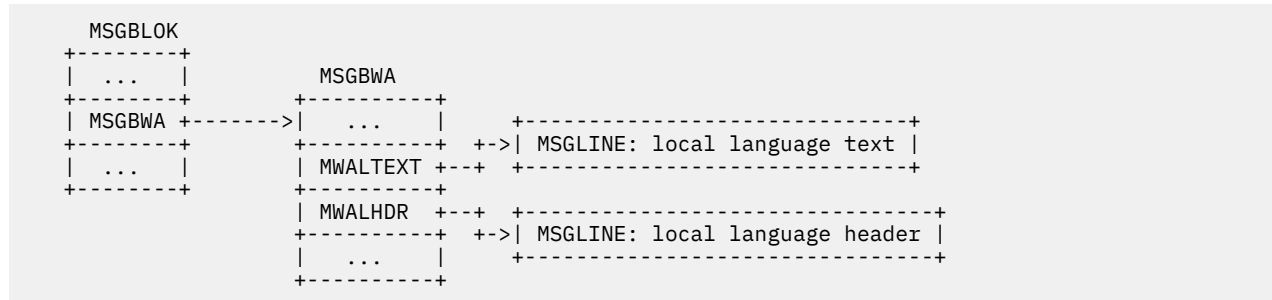


Figure 32. Overview of Message Builder Data Areas

MSGLINE

As the message text is built, RSCS gets a message line element (MSGLINE) to temporarily hold the lines of the message. For multiple line messages, RSCS creates a chain of MSGLINE elements to reflect the order of the lines in the message.

Event Scheduler

The event scheduling task (EVE) runs commands or events at the times specified in the EVENTS file or by the SCHEDULE command. This section describes the data structures used to process scheduled events. For more information, see [“Event Manager Task” on page 53](#).

EVEBLOK

An event block (EVEBLOK) represents each scheduled RSCS event. [“EVEBLOK” on page 225](#) describes the format of the EVEBLOK. The chain of EVEBLOKs is anchored at the TEVENTS field in the CVT. The EVEBLOK chain reflects the order in which the events are scheduled. EVEBLOKs can represent a *repetitive event* (one scheduled several times during a day) or a *single event* (one processed at a specific time). Events are also categorized according to how they were scheduled (by entries in the EVENTS file or by the SCHEDULE command).

The EVE task creates a special EVEBLOK for the *midnight event*. The midnight event ensures that the EVE task processes the EVENTS file every night at midnight. The EVE task then builds a new EVEBLOK queue; this queue contains requests specified in the EVENTS file and events that were schedule using the DAILY operand of the SCHEDULE command.

TANBLOK

The EVE task assigns a unique event task number to each EVEBLOK in the EVEBLOK chain. RSCS refers to this task number when it must suspend, resume, or delete an event. The EVE task uses a TANBLOK to allocate a task number. Each TANBLOK contains a bit map that allocates 1024 task numbers for EVEBLOK entries. The chain of TANBLOK entries is anchored at the TTANQ field in the CVT.

Command Authorization

You can limit user access to privileged commands and system information. RSCS uses AUTHBLOK entries to describe any user privileges.

AUTHBLOK

An AUTHBLOK entry represents the information specified on an AUTH statement. The AUTHBLOK entry contains the node and user ID of a privileged user and information about the user’s privileges (for example, full operator, link operator, or CP command privilege). When RSCS initializes, it creates a chain of AUTHBLOK entries, which is anchored at the TAUTH field in the CVT. For more information on the AUTOBLOK structure, see [“AUTHBLOK” on page 212](#).

Printer Related Areas

This section describes the FCB and FORM tables, which contain information about printing characteristics. At times, printer and workstation link drivers might need additional information to process some files. You may also need to define characteristics to print certain files.

FCB Table

During initialization, RSCS builds the forms control buffer (FCB) table (RFCBTAB) from FCB statements in the configuration file. As [Figure 33 on page 32](#) shows, the chain of RFCBTAB entries is anchored at FCBTABA in the CVT. Each element in the chain contains the name of an FCB image and 256 1-byte areas that correspond to line numbers on a page. These 1-byte areas are initialized to zero before the channels are assigned to the line numbers. Channel numbers are assigned by using their line number counterpart as an index to the 256-byte array; this is similar to the paper tape on a 1403 printer.

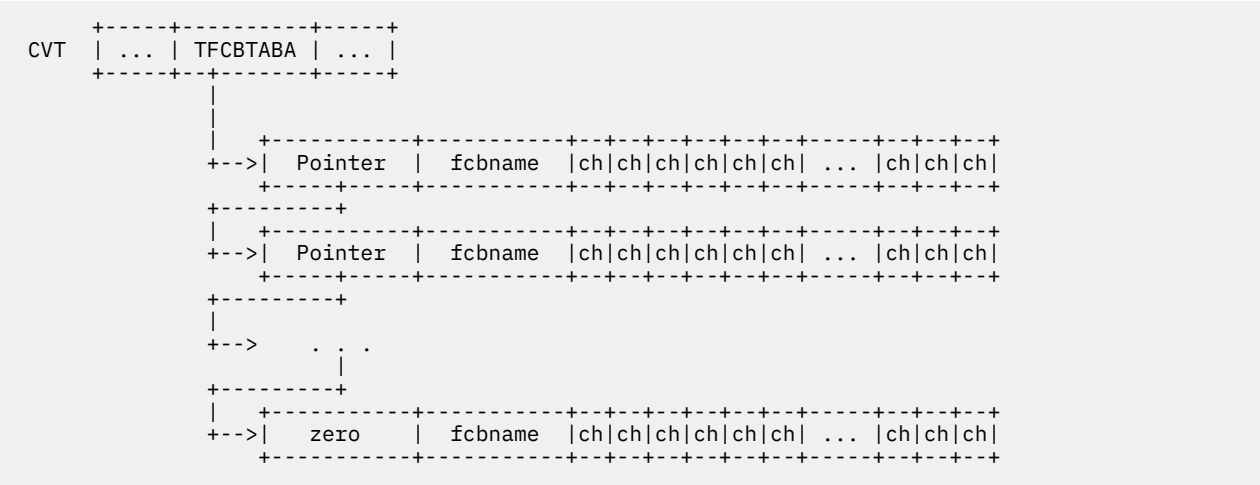


Figure 33. Structure of the FCB Table

FORM Table

RSCS creates a FORM table entry for each valid FORM configuration file statement. FORM entries are added to a chain anchored at the TFORMTAB field in the CVT. (For the entire FORM block structure, see [“FORM” on page 213](#).) Each FORM entry contains the following information:

- Form name
- Type of separator page (VM, RSCS, or none)
- Width, length, and number of lines per inch on the form.

FORM statements define specific printing characteristics and give them a symbolic form name. You can then specify this form name to print files with the printing characteristics associated with the form.

Chapter 3. Task Management

This chapter provides an overview of the facilities and processes that enable RSCS to create tasks.

Using GCS Facilities

This section describes the GCS facilities that enable RSCS to create and manage tasks. For more information, see [z/VM: Group Control System](#).

Task Management Facilities

GCS provides an environment within a virtual machine that can contain many independent threads of execution called *tasks*.

Tasks can create additional tasks by issuing the GCS ATTACH macro. This resulting task is called a *subtask* and is dependent on the *parent task* that created it. Each task is represented in GCS by a *task control block* (TCB). The TCBs are chained to represent the task's relationships. Generally, subtasks must end before their parent task ends. However, a subtask that is attached from a GCS command processing module does not have to end before the command processor ends.

While a task is executing, GCS can pre-empt the task at any point during its processing. GCS can then dispatch another task to do some work. Within a task, GCS maintains *state blocks* to represent the levels of a task's execution. New levels of execution can result from a synchronous operation within a task, such as an supervisor call instruction (SVC). Asynchronous external operations (for example, another task issues the GCS SCHEDX macro to schedule work) also create a new level of execution.

When a new level of execution is created for a task, the task suspends its processing at the current level. When one level has finished, the task continues processing the previous level at the point where its processing was suspended. While work is in progress at the highest level, the task suspends its processing of all lower levels of execution.

GCS Console Task

When GCS initializes, it creates the console task and the command task to interface with RSCS. The GCS console task manages command input and messages written to the RSCS console. The console task processes the following types of commands:

- Immediate commands, like HX.
- Native commands, like QUERY and LOADCMD, are processed by GCS. Execs, which are a special type of native command, are processed by the Procedures Language VM/REXX Interpreter.
- Commands that have been identified by the LOADCMD command, like RSCS. To execute these commands, the GCS command task calls an entry point identified by a previous LOADCMD (for example, when the RSCS INIT command is issued).

GCS Command Task

The GCS command task executes program code that has been identified to GCS by a LOADCMD. GCS can access programs that reside in the following locations:

- A global load library, identified by the GLOBAL LOADLIB command. Programs are identified by entry point names and aliases created by the linkage editor.
- A discontinuous saved segment (DCSS); GCS uses the CONTENTS macro to identify these programs.
- A load module previously loaded into storage; programs are identified using the IDENTIFY macro.

The GCS command task issues the LOADCMD command to load a module into storage and identify it as a GCS command processor. The command task also calls the ATTACH, BLDL, DELETE, LINK, and LOAD macros, which are described in [Table 6 on page 34](#).

GCS Macros

During their processing, RSCS tasks call many GCS macros. [Table 6 on page 34](#) lists these macros and their function within RSCS.

Table 6. GCS Macros Issued by RSCS Tasks

Macro	Function
ATTACH	Loads a module and creates a new task.
CHAP	Changes the run priority of a task. An RSCS task can use CHAP to change its own priority or the priority of another task it created. The issuing task can be suspended if it raises the priority of another task above its own priority.
CMDSI	Allows RSCS to issue commands that ordinarily would be issued from the console. Used in exec processing to issue the GCS FILEDEF command and to execute the exec by name.
DCB	Creates a data control block to manage file I/O.
DCBD	Generates the symbolic name for each field in a data control block for file I/O.
DELETE	Tells GCS to remove a previously loaded module from storage.
DEQ	Allows RSCS to release control of a serially reusable resource.
DETACH	Tells GCS to remove a task from storage.
ENQ	Allows RSCS to request control of a serially reusable resource.
ESTAE	Specifies and describes an exit routine to GCS, which receives control if a task abends and provides additional task termination functions.
FREEMAIN	Returns storage to GCS.
GENIO	Tells GCS to connect RSCS to a virtual I/O device, start a channel program, or release the device.
GET	Moves a record from a file into RSCS storage.
GETMAIN	Gets an area of storage from GCS.
IUCVCOM	Is the GCS interface to the CP IUCV facility.
IUCVINI	Establishes the IUCV environment for RSCS usage.
LINK	Loads a module into storage and calls its named entry point.
LOAD	Brings a load module containing a specified entry point into virtual storage and makes the code at that entry point available for use.
OPEN	Tells GCS to access a file for processing.
POST	Tells GCS to set an ECB when an event has occurred. For more information, see “POST and WAIT Macros” on page 40 .
STIMER	Tells GCS to set a timer for a specified period and defines an exit routine that is scheduled when the timer elapses.
TTIMER	Cancels a timer previously set by STIMER.
WAIT	Suspends a task until an event, identified by a POST macro, occurs. For more information, see “POST and WAIT Macros” on page 40 .
WTO	Tells GCS to display a message at the RSCS console.

Attaching System Tasks

Before RSCS can begin to function, its system tasks must be attached and started. The communications task (REX) is the first task that is attached when RSCS initializes. The REX task then attaches the other RSCS system tasks. This section provides an overview of system task creation and management. For more information about each task, see [Chapter 5, “System Tasks,”](#) on page 43.

Starting the Communications Task

RSCS initialization begins when the GCS command task routes the INIT command to entry point DMTMANEP in the RSCS program. The GCS command task issues the GCS LINK macro to pass this command to RSCS (see [Figure 34](#) on page 35). This creates a state block on the GCS command task to run DMTMANEP.



Figure 34. Processing the INIT Command

After DMTMANEP identifies the INIT command, it calls DMTCMXEP to parse and validate the command. It then calls the GCS IDENTIFY macro to identify the entry points for all RSCS task modules (this may have been done before if RSCS had previously initialized).

As [Figure 35](#) on page 35 shows, DMTMANEP calls the GCS ATTACH macro to start the REX task and waits for it to initialize.

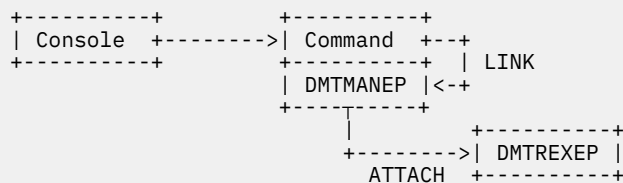


Figure 35. Attaching the Communications Task

When the REX task posts its initialization complete ECB (DMTREXIC), DMTMANEP returns control to GCS. At this point, RSCS is represented to GCS by module DMTREX. All commands, including those issued at the GCS console and initially handled by DMTMANEP, are executed under the REX task. All other RSCS tasks become subtasks of the REX task, which also owns all storage (except file data areas and link driver work areas).

Starting System Tasks

During RSCS initialization, the REX task calls DMTIRX to read the configuration file and calls Exit 0 to prepare any information needed for other exit points. The REX task then attaches each mandatory system task (spool manager, auto-start, EXEC processor, event scheduler, and port redirector) and waits for them to initialize (see [Figure 36](#) on page 36).

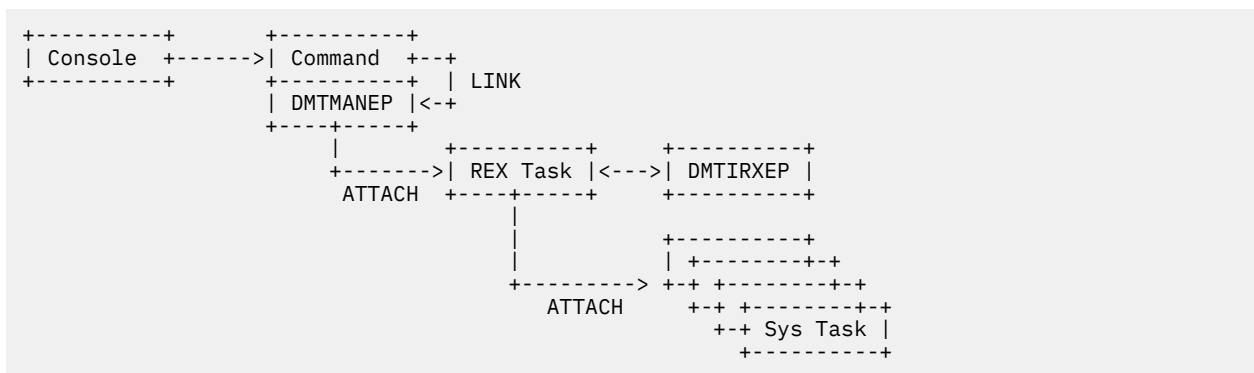


Figure 36. Attaching Mandatory System Tasks

If DMTIRX, an Exit 0 routine, or a system task does not initialize properly, RSCS initialization fails. The REX task tells all initialized system tasks to end and waits for this termination to complete. Termination processing occurs in reverse order; the last system task to initialize is the first to end. Exit 1 is then called to perform any termination processing needed for any other exit routines previously installed. Finally, the REX task ends and indicates the initialization error.

The Port Redirector task, which is the RSCS interface to TCPNJE-type links, is started automatically when you initialize RSCS. For more information about this task, see [“Port Redirector Task” on page 60](#) and [z/VM: RSCS Networking Operation and Use](#).

Starting the SNA Control Task and Auto-Answer Tasks

The REX task does not start the SNA control task and auto-answer tasks during RSCS initialization. Rather, each task is started when a NETWORK START or ENABLE command, respectively, is issued.

SNA Control Task

When the NETWORK START command is issued, the REX task calls DMTCMZ to process the command and attach the SNA control task (SCT). DMTCMZ does not wait for the SCT task to initialize. Rather, it returns control to the REX task so that other commands can be processed. If the SCT task fails to initialize, RSCS continues to operate but no SNA functions will be available. The SCT task ends when a NETWORK HALT command is issued or automatically if it is active when a SHUTDOWN command is issued.

Auto-Answer Tasks

Each time an ENABLE command is issued for a NODIAL port, the REX task attaches an auto-answer task (DUP). Several auto-answer tasks, one for each enabled port, can be active at a time. Because the REX task does not wait for the DUP task to initialize, RSCS continues to operate even if the task initialization fails.

Auto-answer tasks can be ended by a DISABLE command. They also end automatically if they are active when a SHUTDOWN command is issued.

When a port receives a phone call, its corresponding auto-answer task transforms itself into an appropriate link driver task. The caller identifies the needed link driver on the sign-on card it provides.

Starting Link Driver Tasks

RSCS can start the following types of link driver tasks:

- Link driver, started by a START command or by the auto-start task
- Link driver for an auto-dial link, started by a START command or an auto-start task
- Link driver, started when RSCS receives a call on an auto-answer port
- Session driver, started as a primary LU (PLU) by a START command, auto-start task, or the VTAM operator

- Session driver, started as a secondary LU (SLU) in response to a START command issued from a remote RSCS node or by the VTAM operator.

Link Driver Tasks

When a START command is issued for an inactive non-SNA link with a nonzero line address, the REX task calls DMTCMYST. DMTCMYST then processes the command and calls DMTBPLAL. The REX task does not wait for the link driver task to initialize. Rather, it continues to process other available work.

DMTBPLAL attaches the appropriate link driver task and increments the count of active links. If necessary, DMTBPLAL also loads and calls the code for the link (standard RSCS links are loaded when RSCS initializes). If it cannot load the link driver task, DMTBPL issues an error message and returns control to GCS.

If the ASTART option is specified, the link may also be started automatically when a file is enqueued on the link. The AXM task, which detects arriving files, enqueues the START command for the link driver on the REX task's command queue. It then posts REX's command arrival ECB.

Link drivers can be stopped by the DRAIN, STOP, or FORCE commands. If they are active when a SHUTDOWN command is issued, the link ends automatically. When a link driver task ends, it returns control to DMTBPLEP. DMTMANEX then performs clean up for the link (closes files, resets fields) and decrements the active link counters, as needed.

Session Driver as Primary LU

When a START command is issued for an inactive SNA link, the REX task calls DMTCMY, which passes the request to the SCT task. The REX task does not wait for the session driver task to initialize.

The SCT task issues a SIMLOGON request to VTAM to request an SNA session. When the SNA session is ready to start, VTAM issues the GCS SCHEDEX macro to schedule RSCS's LOGON exit, DMTVXTLG. DMTVXTLG, which executes under a state block in the SCT task, verifies logon parameters. It also calls DMTBPLAL to attach the session driver task. DMTBPLAL also increments the count of all links and the count of session drivers.

When processing a START request, the SCT task does not wait for the logon exit to be scheduled; it continues its processing to perform other SNA functions. The START command does not have to be issued to start a session driver. A VTAM operator on a remote node can also drive DMTVXTLG directly. For more information, see [“SNA Control Task” on page 55](#).

Session drivers are stopped by the DRAIN or STOP command. If they are active when a NETWORK HALT or SHUTDOWN command is issued, they stop automatically.

Session Driver as Secondary LU

An SNANJE session driver can also be started as a secondary LU, in response to commands issued by a VTAM operator on a remote node. In this case, VTAM processes the request and issues the GCS SCHEDEX macro to schedule the RSCS SCIP exit, DMTVXTSC. DMTVXTSC executes under a state block in the SCT task and calls DMTBPLAL to attach the session driver task.

Auto-Start Links

When a START command is issued for an auto-dial link, the REX task passes the command to the auto-start task (AST). The REX task does not wait for the task to initialize. The AST task calls DMTBPLAL to attach the link driver task.

Auto-Answer Links

When an auto-answer task (DUP) detects a call on an enabled port, it examines the sign-on record sent by the caller. If the sign-on record requests a link that has been defined on the local node, the DUP task issues the GCS LINK macro. The LINK macro passes control to the entry point of the link driver task.

indicated in the sign-on record. The DUP task then transforms itself into the specified link driver task; RSCS does not create a new task.

Capturing Task Abends

RSCS uses the GCS ESTAE facility to capture task abends. When it initializes, each task issues an ESTAE macro to identify the routine to process any abends. RSCS defines three main ESTAE exit routines:

DMTMANSE

Processes system task abends. RSCS ends, but any CP command specified on the RECOVERY statement may be executed. This command can re-IPL the GCS virtual machine and cause RSCS to be reinitialized. (If the SCT task abends and no session drivers are active, RSCS does not end).

DMTMANDE

Processes abends in link drivers; the link driver task is not restarted.

DMTMANPE

Processes abends in auto-answer tasks.

The ESTAE exit routines in RSCS process a dump, in the format specified by the DUMP statement, and write a dump summary to the console. Exit 35 can be used to suppress multiple dumps of a known problem; see [z/VM: RSCS Networking Exit Customization](#) for more information.

If a task expects the possibility of an abend, it issues the ESTAE macro before the instruction that might cause the abend. The exit routine specified on the ESTAE macro sets a flag if the abend occurs. The task then tests this flag to determine if the instruction executed successfully. After that instruction is executed, the task cancels the ESTAE exit. For example, the AXM task may expect an abend if the z/VM system does not support some Diagnose X'14' subcodes. Also, if the appropriate code is not found, an abend will occur when DMTBPLLX attempts to load an exit routine or link driver task.

End of Task Exit Processing

When each RSCS task is attached, it specifies the end-of-task exit routine, DMTMANEX, on the ETXR operand of the ATTACH macro. DMTMANEX performs any clean-up and issues the DETACH macro to end the subtask. DMTMANEX first identifies the type of task that ended. It can then determine what type of cleanup is needed for the task.

DMTMANEX is scheduled in the parent task when the subtask returns control to GCS. For example, when a link driver ends, DMTMANEX runs as a state block on the REX task. The REX task does not become active until DMTMANEX completes its processing. For session drivers, DMTMANEX runs as a state block to the SCT task. In this case, VTAM cannot issue requests to start other session link drivers. The SCT task does not become active again until DMTMANEX completes its processing.

DMTMANEX and DMTBPPEP maintain counts of the number of active link and session driver tasks. If a NETWORK HALT command is being processed and there are no active session driver tasks, DMTMANEX posts DMTSCT's stop ECB. If a SHUTDOWN command is processed and there are no more active link driver tasks, DMTMANEX posts the REX task's stop ECB.

When a link driver ends, DMTMANEX may try to restart it by enqueueing a request onto the exec processor task. The EXE task can then schedule a recovery exec for the link. When a port ends, DMTMANEX can enqueue an ENABLE command to the REX task to re-enable the port.

When the REX task ends, DMTMANEX issues message DMT100I to indicate that RSCS has shut down. If RSCS stops because of an abend, DMTMANEX also issues the CP command specified on the RECOVERY statement (see [z/VM: RSCS Networking Planning and Configuration](#) for more information).

Chapter 4. Inter-Task Communication

As described in Chapter 2, “RSCS Structure,” on page 11, RSCS tasks must communicate with each other to accomplish most transactions. This chapter describes the RSCS and GCS facilities that enable tasks to communicate.

Using GCS Services

As described in Chapter 3, “Task Management,” on page 33, RSCS uses GCS supervisor services to create tasks. While a task performs work, it appears to run without interference from other tasks. The GCS *dispatcher* causes the processor to process each dispatchable task, in turn, for a few milliseconds at a time.

A task can be *dispatchable* or *nondispatchable*. A dispatchable task is eligible to perform work; it can be running or in the dispatcher queue waiting to run. A nondispatchable task is waiting to receive and perform some work.

If no RSCS tasks are dispatchable (this can happen when no links are active and no files are arriving), the RSCS virtual machine enters an *enabled wait* state. In this state, an interrupt must occur to cause the GCS dispatcher to make at least one RSCS task dispatchable again.

Several RSCS tasks may be dispatchable at the same time. When this occurs, GCS may dispatch a second task between the processing of two instructions in the first. The process by which one task can interrupt another task’s operation at any time is called *preempting*.

Task Synchronization

In many cases, an RSCS task must wait for another task to complete or provide work. For example, this includes the following situations:

- When RSCS initializes, DMTMAN waits for the REX task to initialize before returning control to the console.
- Link driver tasks wait for an I/O operation to complete, a file or command to arrive, or notification to end.
- The AXM task waits for commands, files, or a shutdown request.
- During RSCS termination, the REX task waits for other tasks to end before it ends.

This section describes the facilities RSCS uses to synchronize operations and communications among its tasks.

Task Queues

Each RSCS system and link driver task has one or more queues for messages or commands. When a new element is placed on a queue, the task tries to accomplish the requested work.

Work items for the task are placed on one of these queues. Each item on the queue is represented by a QBLOCK. A QBLOCK contains an 8 byte header field that points to the address of the next and the last QBLOCK associated with the queue. This header field is followed by the command or message element.

The queue anchors for system tasks reside in the primary module that comprises the task. For example, the communications task’s command queue anchor is in DMTREXCQ. The spool manager task’s command queue anchor is in DMTAXMRQ. However, the queue anchors for link driver tasks do not reside in their primary modules. Rather, the LINKTABL entry for each link contains the link driver task’s queue anchors.

Event Control Blocks

Each queue anchor is associated with an event control block (ECB). Tasks use ECBs to notify each other when they must perform work. Tasks also monitor their own ECBs to determine when they have received requests to perform work. Each ECB is a fullword in length and contains the following sections:

Bits	Purpose
0	Indicates that a task has issued the GCS WAIT macro on this ECB. Only one task at a time can wait on a specific ECB.
1	Indicates that a task has issued the GCS POST macro for this ECB.
2-31	May contain more information about the work request.

Most RSCS tasks respond to several types of work requests. For example, the AXM task may receive stop requests, commands, or new files. In this case, the task may contain an *ECB list*. The ECB list contains the addresses that represent each type of work the task may process. The ECBs do not have to be together in storage, but the ECB list must contain a contiguous list of addresses. To identify the end of the ECB list, bit 0 in the last address of the ECB list is set to 1.

POST and WAIT Macros

RSCS tasks often use the GCS POST and WAIT macros to synchronize their communications.

When a task must wait for work, it issues a WAIT macro for the ECB associated with the specific type of work. Before it issues a WAIT macro, a task ensures that all its ECBs are cleared. The GCS supervisor then suspends that task until it is notified of a work request by another task. The task is then considered to be waiting on that ECB; it is not dispatched until the ECB is posted.

When a task has work for another task, it can issue the GCS POST macro for the other task's ECB. The posted ECB is associated with the task's work queue (file, command, message).

When a task posts an ECB that another task is waiting on, the second task can become dispatchable again. However, the task may not begin to run when the ECB is posted. The task may run only when it is selected by the GCS dispatcher. As a result, several ECBs may be posted when the task actually starts to run or the same ECB may be posted several times. When it completes the work associated with an ECB, a task clears the ECB before it issues another WAIT macro to wait for more work.

For example, after the REX task initializes, it attaches the other system tasks. The REX task attaches the spool manager task. It then issues a WAIT macro and waits for the spool manager task to signal that it has successfully started. When it initializes, the spool manager task posts its initialization complete ECB (DMTAXMIC). The REX task then attaches the next system task and waits for it to initialize.

This process continues until all system tasks have initialized or a system task indicates that it cannot initialize successfully. If a task cannot begin, it posts its termination ECB. The REX task also waits on an ECB list for termination ECBs. When one of these termination ECBs is posted, the REX task then posts the termination ECB for each system task that has already initialized successfully.

GCS and VTAM also use the POST macro to notify RSCS of events. GCS posts an ECB associated with the specific device or interrupt code. VTAM may also post ECBs when a VTAM operation completes.

Processing Several ECBs

A task can also issue a WAIT macro to refer to several ECBs at the same time. In this case, the task indicates that it is waiting on an ECB list. If another task does not post one of the ECBs in the list when the WAIT macro was issued, the task waits (becomes nondispatchable) until one of the ECBs is posted. The task must then test each ECB in the list to determine if it was posted by an other task. When it finds the posted ECB, the task then performs the requested work. All RSCS tasks cycle through an ECB list to determine if work has been received.

DMTCOMNQ and DMTCOMDQ

In addition to the POST and WAIT macros, RSCS also uses two routines to notify tasks of requested work. These routines, DMTCOMNQ and DMTCOMDQ, form the basis of the DMTCOMNQ protocol.

Using the DMTCOMNQ protocol, a task can provide additional information (more than that supplied in the 30 spare bits of the ECB) about the work request. For example, to process a command, a task needs to be told of the arrival of the command element and the specified command text.

Each task has an ECB that is associated with a queue. For example, DMTREXCQ, the main command queue for RSCS, is controlled by the DMTREXCM ECB. It also provides a fullword anchor to contain the actual text of the commands.

Issuing a Work Request

To issue a request, a task creates a *request element*, which contains the command text. It then calls DMTCOMNQ specifying the address of the request element and the address of the anchor for the ECB. DMTCOMNQ copies the request into a new storage area (a QSABLOK). It then enqueues this area onto the serving task's anchor. Then, the requesting task issues a POST macro for the ECB. The task should not issue the POST macro until after it calls DMTCOMNQ.

Receiving Work Requests

When its ECB is posted, the serving task calls DMTCOMDQ and specifies the anchor and an area of storage to receive the element. DMTCOMDQ removes the first request element from the anchor's queue and copies it into the serving task's storage area. If there are no request elements on the anchor, DMTCOMDQ issues return code 4.

The task that processes the request should clear the associated ECB before calling DMTCOMDQ. DMTCOMDQ removes one element from the queue each time it is called. The task should call DMTCOMDQ until no elements remain on the queue. After DMTCOMDQ successfully runs, each dequeued request elements is placed in a work area where it can be examined and run by the task.

Request Element Format

The requesting and serving tasks agree on the format of each request. DMTCOMNQ and DMTCOMDQ do not process the details of the request. However, the first byte of each request must indicate the length of the request excluding that byte (the total length, minus one).

The CMNDAREA macro maps most request formats (see “CMNDAREA” on page 231). CMNDAREA defines a standard header format that includes two bytes (CMNDTYPE and CMNDMOD) that determine the format of the rest of the request. The Type L3 format is used for most RSCS commands. The L3TEXT field of this request element contains the text of the command to be processed.

Other types of RSCS request elements include the EVEBLOK and the MSGBLOK. The EVEBLOK is passed from the SCHEDULE command processor to the EVE task to request to schedule or change an event. The MSGBLOK is passed from the message processing routines to the REX task when they need to issue messages.

Task Serialization

In some cases, several RSCS tasks may want to process the same data or use the same resource to perform work. However, they cannot do their work simultaneously because the tasks would interfere with each other. In this case, RSCS serializes the tasks' access to certain resources.

RSCS defines the following types of resources (exits routines can also define new resources to RSCS):

Resource	Description
Spool	Includes all data areas and routines in DMTAXM. It also includes file queue-related data structures (TAG and TASHADOW elements).

Resource	Description
File	Includes the dynamic ddname allocation vector and the file work area (FILWORKA) chain for DMTCOMFI.

To prevent many tasks from simultaneously accessing a resource, RSCS uses "locking" mechanisms, which are described in the following sections. The term *critical section* describes the section of code in a task module that needs to use a resource that other tasks also need. Any number of tasks can wait for access to a resource; however, only one task at a time can use a resource. Tasks receive access to a resource in the order they call DMTRESLO to request a lock on that resource. While a task has a lock on a resource, other tasks can continue to communicate with each other and process requests, such as I/O operations.

For example, all spool-related areas in DMTAXM are serialized. These areas can run under all RSCS system tasks and link driver tasks. DMTAXMRQ ensures that only one task at a time can use these resources.

DMTRES

Routines in module DMTRES ensure that tasks efficiently share resources. Each RSCS resource is represented by a RESBLOK (see [“RESBLOK” on page 25](#)). A task uses a RESBLOK to identify the resource it wants to use; it then calls routines in DMTRES to access the resource.

Locking a Resource

To exclusively use a resource, a task points at the RESBLOK that represents the needed resource and calls DMTRESLO. DMTRESLO then places the task in a wait state until the resource becomes available.

If the resource is in use and more than one task has requested it, each contending task queues a resource queue block (RESQBLOK) on the resource block. Use of the resource is determined on a first-in-first-out (FIFO) basis.

Unlocking a Resource

When a task no longer needs the resource, it calls DMTRESUN to make the resource available to the next waiting task. DMTRESUN issues a POST macro to inform this task that the resource is now available.

Clearing a Lock

If a task ends while it is contending for a resource, the RSCS end-of-task processing routine calls DMTRESCL. If the ended task owns a resource that other tasks are waiting to use, DMTRESCL issues a POST macro to tell the next waiting task that the resource is now available. If the ended task was waiting for a resource but did not own it, DMTRESCL removes that task's RESQBLOK from the resource on which it is queued.

Disabling Interrupts

In some cases, an RSCS task will disable interrupts for the processor on which RSCS is running to prevent other tasks from processing any work. When interrupts are disabled, however, no other tasks can be dispatched while the lock is held, even if they are working in different areas. Also, the task that disables the interrupts cannot make additional work requests (including I/O requests) to other tasks. When it no longer needs that resource, the task enables the interrupts again.

ENQ and DEQ Macros

RSCS also uses the ENQ and DEQ macros to serialize access to a resource, such as a PORT table. Using these macros, tasks can share access to a resource if they do not interfere with each other's processing. They can also identify the resource by its name, rather than storage address. For information about these macros, see [z/VM: Group Control System](#).

Chapter 5. System Tasks

This chapter describes the RSCS system and auto-answer tasks. Except for the SNA control task and the port redirector task, each task must be present for RSCS to function. All system tasks are serially re-usable and nonreentrant.

Each task is identified by two names: a task name and a module identifier. For example, the spool manager task is also called the AXM task. However, all system tasks do some processing outside their main module. Other tasks, such as link drivers, may use routines in a system task's main module for processing.

Communications Task

The communications task (REX) is the main RSCS system task. It is the parent task of all other RSCS tasks (system, link driver, and auto-answer). It owns all storage acquired by the GETMAIN macro, except those work areas needed by active links and data areas representing files. These areas are owned by the link driver tasks and the spool manager task, respectively.

Initialization

The REX task is attached when an INIT command is issued on the RSCS console or by a GCS exec (see [Figure 35 on page 35](#)). The command is passed to DMTMAN, which determines if RSCS is initialized and passes the command to DMTCMX. After verifying the command, DMTCMX returns a zero return code to DMTMAN. DMTMAN then calls the GCS ATTACH macro to attach the REX task at entry point DMTREXEP. DMTREXEP then issues the ESTAE macro to identify the routine GCS will call if the communications task abends.

Reading the Configuration File

The REX task calls DMTIRX to process the configuration file. DMTIRX reinitializes any data areas, including most fields in the CVT, that may have been used if RSCS was previously initialized. It then issues Diagnose code X'00' to determine the following system information:

- Time-of-day (TOD) clock offset from Greenwich Mean Time (GMT)
- Features supported by the level of z/VM RSCS is running on
- User ID of the RSCS virtual machine

DMTIRX also determines whether or not RSCS is fully enabled. This determines which link tasks will be allowed to be started. DMTIRX calls DMTCOMFI to get records from the configuration file. It then calls DMTPAF to parse each record. As [Figure 37 on page 43](#) shows, DMTPAF then passes the record back to a post-processing routine in DMTIRX.

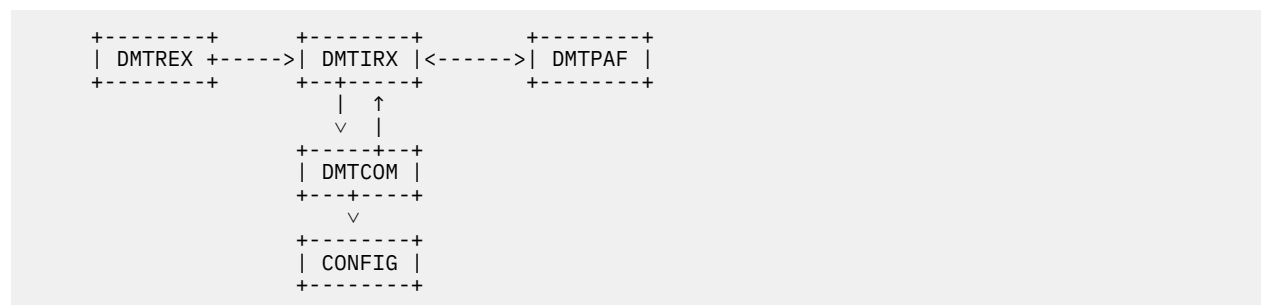


Figure 37. Processing the Configuration File

After the configuration file is processed, the REX task obtains storage for various data areas needed by other RSCS tasks. It also builds the initial hash tables for LINKTABL, LUNAME, NODE, and ROUTEGRP entries.

Exit 0

After the configuration file is processed, the REX task calls Exit 0. Exit 0 can be used to prepare information for other exit routine packages. An Exit 0 routine can also cause RSCS initialization to fail by returning a return code of 8. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Processing DEST Statements

If the configuration file contains DEST statements for a Printer Services Facility (PSF) printer, RSCS does not process a destination identifier file. If the configuration file does not contain DEST statements, RSCS determines if a file with the ddname "DEST" has been defined. If not, RSCS assumes that no destination identifiers have been defined.

If the ddname is defined, RSCS attaches the prefix "DEST" to each record in the file. RSCS then processes the information on each record as if had been specified on a DEST configuration file statement. When all records in the file are processed, the REX task regains control.

Error Processing

If DMTIRX finds errors while processing a file, it issues return code 4 when returning control to the REX task. The following situations may cause an error:

- RSCS is not defined in an ESA-mode virtual machine
- RSCS is installed on an incorrect level of z/VM
- Insufficient storage to initialize
- LOCAL statement or system ID is missing or not valid
- Problems in reading the configuration or destination identifier file
- Errors are found when RSCS is in "no tolerance" mode.

Creating System Tasks

After initializing, the REX task attaches the other RSCS system tasks: spool manager task (AXM), exec processing task (EXE), auto-start task (AST), the event managing task (EVE), and the port redirector task (PRD). For each task, the REX task calls the GCS ATTACH and WAIT macros. These macros suspend the REX task until each attached task indicates if it has initialized successfully.

When all system tasks initialize, the REX task posts its initialization complete ECB (DMTREXIC). DMTMANEP then issues message DMT000I to the RSCS console to indicate the RSU service level and that the RSCS virtual machine is running. It also issues the CP command SET SMSG IUCV, which allows RSCS to receive SMSGs from users and operators on the local node.

Trapping Special Messages

When RSCS receives a special message (MSG) from a local virtual machine, it is trapped and executed as a command. RSCS use an IUCV interrupt handler to trap the MSGs. The REX task calls the GCS IUCVINI and IUCVCOM macros to establish an IUCV path to the *MSG system service.

When RSCS receives an MSG, GCS calls DMTREXIU, the IUCV interrupt handler, to process the IUCV interrupts that occur. DMTREXIU issues GCS macros with the RECEIVE option to receive the element. It then calls DMTCOMNQ to place the element on the REX task's command queue.

Processing Commands

After RSCS initializes, the REX task monitors the execution of commands for the RSCS virtual machine. These commands may be issued from the RSCS console, a remote node, or arrive as an SMSG executed on the local node (like from RSCSAUTH).

DMTMAN receives commands that originate from the RSCS console or from a GCS exec. DMTMAN places these command elements in a static area. This ensures that the REX task only processes one console command at a time.

After calling DMTCOMDQ to remove an element from its command queue, the REX task calls DMTCMX to process the command. However, the REX task may call DMTCOMNQ to pass the command element to another task to complete the command processing. Each RSCS task has a command queue and an ECB through which it is notified of the arrival of a command element. When a NETWORK HALT or STOP command is issued, the REX task posts each active SNA task's terminate ECB. The following table lists each RSCS task and the commands they process.

Task	Commands Processed
Communications	CP, CPQUERY, EXIT, QUERY, ROUTE
Spool manager	CHANGE, ORDER, PURGE, REORDER, TRANSFER
Link driver	BACKSPACE, DRAIN, FLUSH, FREE, FWDSPACE, HOLD
SNA control	START (for SNA links)
Exec processor	EXEC
Event scheduler	SCHEDULE
Port Redirector	TCPIP

Some commands are usually processed by the REX task. However, the REX task may also call the AXM task to re-enqueue existing files according to the network structure. For example, when processing the ROUTE command, the REX task passes a REORDER command element to the AXM task after the network definitions are updated.

When the following commands are issued, the REX task calls the GCS ATTACH and DETACH macros to create or remove system tasks:

START

Attaches a non-SNA link driver task

FORCE

Detaches a non-SNA link driver task

ENABLE

Attaches an auto-answer task (DUP)

NETWORK START

Attaches the SNA control task (SCT)

TCPIP START

Attaches the TCP port redirector task (PRD)

TCPIP STOP

Detaches the TCP port redirector task (PRD)

DMTCMX, the main command processing module, receives all commands from the REX task. DMTCMX calls Exit 19 to determine if RSCS should process the command. It then calls DMTPAF to parse the command element. If RSCS does not recognize the command, DMTCMX calls Exit 29 to process the command. DMTCMX also contains routines to process certain RSCS commands. If it cannot process a command, DMTCMX calls one of the following modules:

- DMTCMA (see [“DMTCMA” on page 177](#))

- DMTCMB (see [“DMTCMB” on page 177](#))
- DMTCMQ (see [“DMTCMQ” on page 177](#))
- DMTCMY (see [“DMTCMY” on page 178](#))
- DMTCMZ (see [“DMTCMZ” on page 179](#)).

A PAFBLOK describes each RSCS command and contains the address of the routine that executes the command. The command's CDEF entry in the RSCSCMDS macro (see [“CDEF Macro” on page 119](#)) contains the name of this processing routine.

Issuing Return Codes

When commands originate from the RSCS console or from an exec, RSCS issues a return code to GCS or to the exec. Return code zero indicates that the command completed successfully; a nonzero return code indicates an error.

A nonzero return code corresponds to the message number of the error message that is generated as the command is processed. DMTCMX passes this number to the REX task, which passes it on to DMTMAN. DMTMAN then returns the number to GCS.

Shutting Down RSCS

When a SHUTDOWN command is issued or most system tasks abend (except for the port redirector task), DMTMANEX posts the REX task's termination ECB and waits for all link driver tasks to end. After the link drivers end, DMTMANEX tells the REX task to end. The REX task then issues the CP command SET SMSG OFF, severs its IUCV connection to the *MSG system service, and posts each system task's termination ECB.

When each system task ends, DMTMANEX posts the task's ECB in the SYSIDENT table. After the system tasks end, the REX task calls Exit 1 to perform termination tasks for any exit routines that were initialized by Exit 0. On return from Exit 1, the REX task ends and returns control to GCS.

When GCS calls DMTMANEX to signal the end of the communication task, it issues messages to signal that RSCS is no longer active. If a CP command was specified on the SHUTDOWN command, the command is executed by Diagnose code X'08'.

If RSCS ends because a system task abends, the REX task runs the command specified on the RECOVER statement. This command can be an IPL command, which may cause RSCS to be reinitialized. See [z/VM: RSCS Networking Planning and Configuration](#) for more information.

Spool Manager Task

The spool manager task, AXM, is the RSCS interface with CP spool functions. It detects files arriving in RSCS's virtual reader and enqueues them on the proper links for transmission. It also maintains the data structures that represent files as they are processed in the network.

The AXM task accepts files and completes command execution. When responding to a link driver's request, however, code in the DMTAXM module executes under the control of the link driver task. The module DMTAXM has three main functions:

- Accepting files that have arrived in the virtual reader
- Completing the execution of commands that manipulate files or queues
- Responding to a link driver's request to open and close files for input and output, and to open transmission algorithms.

All operations in DMTAXM are executed under the control of the resource lock, DMTAXMRS. This ensures that only one task accesses the file and queue data structures.

Initialization

As it initializes, the AXM task calls the ESTAE macro to identify DMTMANSE as its permanent task abend routine.

Setting Up Virtual Devices

The AXM task also manages the use of virtual unit record devices in the RSCS virtual machine. The AXM task defines a virtual reader device for the RSCS virtual machine at address 0001. The control program then places files spooled to the RSCS virtual machine in that virtual device. The AXM task calls the GCS GENIO macro to tell GCS to notify it of any interrupts on device 0001.

The AXM task also manages a pool of input and output UR devices, which are defined by the CHANNELS configuration file statement (see *z/VM: RSCS Networking Planning and Configuration* for more information). The AXM task checks the channels specified on the statement and detaches any devices defined on the reserved channels. The link driver tasks use these devices when attempting to open an input or output file.

Storage Requirements

As it initializes, the AXM task acquires storage for work areas. This includes work areas for accounting and messages and the save areas acquired by calls to DMTQSAAB and invocations of the GCS GETMAIN macro. These save areas are later used by the link driver tasks when they call DMTAXMRQ.

Processing Reader Interrupts

When it initializes, the AXM task posts its initialization complete ECB, which tells the REX task that it is ready to receive work. The AXM task then posts its file arrival ECB. This causes it to process any files that may already be in RSCS's virtual reader. This ECB is then posted each time GCS detects an interrupt for the virtual reader device 0001.

Getting Information About New Files

When its file arrival ECB is posted, the AXM task gets information from CP about the new file in RSCS's virtual reader. It then creates a TAG element for each file. The following routines in DMTAXM enable RSCS to process new files:

DECGET

Converts the file priority value from EBCDIC decimal to binary.

GETSHADO

Gets a free SHADOW element.

GETSLOT

Gets a free TAG queue element.

GSUCCESS

Issues Diagnose code X'14' to get information about the file.

PARMGET

Parses the destination node and user ID and priority.

TAGGEN

Places information from the file's SFBLOK and CP tag data into the TAG element.

TAGFIND

Determines if RSCS already has a TAG element for the spool ID of the new file.

TAGSETUP

Parses the store-and-forward indicator and prepares registers to parse the destination node and user ID.

The AXM task also calls DMTRER to determine if the file should be rerouted; the reroute destination, if any, is also placed in the TAG element.

Exit 2 and Exit 21

Module DMTAXM contains exit points 2 and 21, which you can use to perform the following functions when a new file is being processed:

- Accept the file
- Accept a store-and-forward file that may have been created by an alternate (trusted) virtual machine
- Reject the file and have RSCS issue a message to its originator
- Reject the file without issuing a message.

Exit 2 is not called when a file reorder is in progress; however, Exit 21 is always called. See [z/VM: RSCS Networking Exit Customization](#) for more information about these exit points.

Updating File Queue Structures

When a file is sent to a local user, the AXM task places its origin in the CP tag text placed on the file. The AXM task then transfers the file to the specified virtual machine. If the file is sent to a remote node, printer, or workstation, the AXM task enqueues it on all links that can send the file.

Exit 6 and Exit 31

Before files are enqueued on a link, DMTAXM calls Exit 6, which lets you adjust a file's priority. DMTAXM also calls Exit 31, which lets you monitor the priority of files on a link's queue. You can also use Exit 31 to prevent a file from being sent on a link.

Informing Link Drivers About Files

When queueing files for transmission, the AXM task places TASHADOW elements on each link that can send the file to its destination. Many TASHADOW elements can represent a single file, or TAG element (see [“TASHADOW”](#) on page 23).

If the TASHADOW element is placed on an active networking link, the spool manager task calls the transmission algorithm specified for that link. The transmission algorithm determines if the file may be selected for transmission and, if applicable, the streams on which the file can be sent.

If the TASHADOW is placed on a non-networking link, the AXM task posts the link's file arrival ECB to tell it of the arrival of a new file. This ensures that an inactive printer driver is automatically told about new files. Any multistreaming link drivers that have available transmission streams are also informed of new files.

Processing Spool File Commands

The AXM task also monitors a command queue onto which other tasks enqueue command requests. The command ECB is posted by the task that calls DMTCOMNQ to enqueue the command. When this ECB is posted, the AXM task calls DMTCOMDQ to dequeue the command element. It then calls the appropriate routine in DMTAXM to execute commands, which are described in the following sections.

CHANGE Command

The REX task passes an encoded form (Type A0) of the CHANGE command to the AXM task for execution. The command element describes the link on which the files are to be changed. It also contains a filter program (created by DMTCQC), which describes the files to be changed on the link. To process the CHANGE request, the AXM task may modify the file's CP tag data, determine the operator form name for the file, or execute the CP CHANGE command.

The AXM task examines all TASHADOW elements to find the files that match the criteria specified on the command. When it finds a match, the AXM task sets the TASPULL bit in the TASHADOW element. It then finds all TASHADOW elements marked with the TASPULL bit and removes all the file's TASHADOWs from the link's file queue. The TAGPLACE routine then places all TASHADOWs back on the link. Exit 6 and Exit 31 can then process the file, using any characteristics that were affected by the CHANGE command.

The AXM task ensures that any files that were previously ordered do not lose their position on the link's queue. Before a TASHADOW element is removed from a file queue, its position on the queue is marked.

CLOSE Command

The CLOSE command (Type A1) is not externally accessible to users or authorized operators. It processes any remaining active files on a link that has deactivated. DMTMANEX, the end-of-task routine, builds a CLOSE command element when a link driver task deactivates. When the SCT task or PRD task deactivates, DMTMANEX also builds a CLOSE command element to close any log traces if network or TCP/IP tracing has been specified.

When processing the CLOSE command, the AXM task scans the active input and output queues for files belonging to the deactivated link. Input files are closed and re-enqueued on links so that they can be sent later. Output files are closed and purged. Active trace files, started by a TRACE command or by the TRACE option of the START or DEFINE command, are not purged. The AXM task then posts the LUWORD field in the link's LINKTABL entry. This tells DMTMANEX that the CLOSE command has been processed.

PURGE Command

The AXM task can receive the PURGE command from the REX task or from a user exit routine. The command element (Type A1) may contain a filter program that describes a file or a list of spool IDs for files that are to be purged. The PURGEFLT command modifier indicates if the filter program is used.

If a list of spool IDs is specified, the AXM task ensures that all the files in the list exist on the specified link. It then determines which files are to be purged.

When it identifies a file that is to be purged, the AXM task calls Exit 4, which you can use to create an accounting record. It then removes the file's TASHADOW elements from all links on which they are enqueued and purges the file from RSCS's virtual reader. The AXM task then returns the file's TAG element to a pool of available TAG elements.

REORDER Command

Each time a REORDER request is issued, the AXM task rebuilds or realigns all file-related data structures. The AXM task can receive a REORDER command element when one of the following commands is issued:

- REORDER (with or without the QUICK option)
- DEFINE
- LOOPING
- NETWORK START
- ROUTE
- When a link goes into "connect" status.

A reorder request can also be processed when the AXM task executes a CLOSE command. The command element (Type A2) that is passed to the AXM task indicates the reason for the reorder request.

TRANSFER Command

The REX task passes the TRANSFER command (Type A1) to the AXM task. The files specified on the command are identified by a list of spool IDs or by a filter program created by DMTCQC. If a list of spool IDs is specified, the AXM task determines if all the spool IDs are valid and if the corresponding files are on the specified link. If a filter program is specified, the AXM task examines all files to determine those that meet the criteria of the filter program. These files are then marked (by the TASPULL bit) and are processed later.

The AXM task rewrites the CP TAG data for the specified files to reflect the new destination. It then removes all TASHADOW elements from the links on which the file originated and queues them on the appropriate links to the new destination.

Managing File Routines for Link Drivers

When a link driver task wants to process a file, it calls DMTAXMRQ to request to use an AXM task resource. This request is identified by an RDEVBLOK, which contains the required information for each type of request (described in the following sections). It also provides fields that the AXM task uses to return information to the calling link driver task.

OPENIN Requests

A link driver task makes an OPENIN request to open an input file (for example, selects a file for transmission). The AXM task selects the appropriate file and gets a 4K file I/O area (FIOA) to use as it reads the file from CP spool. It then defines a virtual reader device on a reserved channel and issues Diagnose code X'14' to open the file.

Because the file's TASHADOW elements are removed from all links, the file's TAG element is placed on the active input queue (anchored at the TAGACIN field in TAGAREA). After processing XAB and form specifications, if applicable, the AXM task returns control with the file's TAG address in the RDEVBLOK to the link driver task.

CLOSEIN Requests

A link driver task makes a CLOSEIN request when a file's transmission completes or ends prematurely. The RDEVBLOK provided by the link driver points to the TAG element of the input file to be closed. The AXM task removes the TAG element from the active input queue and processes any XAB related storage for the file. It then calls Exit 3, which can be used to create accounting records when the file is sent.

When the file has been transmitted and no longer is queued on the node, the AXM task closes and purges the file. If the link could not send the file or if more copies of the file remain to be sent (on non-networking links only), the file is closed but not purged. The link driver task, if necessary, determines if the file should be held on the link. The AXM task then frees the storage for the FIOA and the file's TAG element.

OPENOUT Requests

Link driver tasks make OPENOUT requests to open an output file (for example, write a file that is being received into CP spool). The calling link driver task provides a sample TAG element, which is anchored at the RDEVTAG field in the RDEVBLOK. This TAG element contains a one byte field that identifies the type of UR device that is defined to write the file to spool.

The AXM task calls DMTQSAAB to get an output TAG element for the file. It also obtains a FIOA, which it uses to build channel programs that are executed against the output UR device. The AXM task copies all information in the sample TAG element into another TAG element, which it places on the active output queue. It then returns a pointer to that TAG element in the IOTABLE, which is built at the top of the FIOA.

After processing the TAG element, the AXM task defines the requested output device and its required spooling options. When processing a log trace file, the file is spooled to the UR device. If its destination is a remote node, it is spooled to the RSCS virtual reader. For all other files received on the link, the UR device is spooled to RSCS's virtual reader. This ensures that a user does not receive a partial copy of a file if RSCS suddenly ends.

CLOSEOUT Requests

A link driver task makes a CLOSEOUT request when it needs to close an output file that was opened by an OPENOUT request. In the RDEVBLOK, the calling task provides a pointer to the file's TAG element from the active output queue. The AXM task removes the TAG element from the queue and updates any fields that were not initialized by the link driver task.

If the file is received on a non-networking link, the AXM task calls DMTRER to determine if the file should be rerouted. Files received on networking links are processed by DMTRER as their NJE headers are received.

The link driver task may close the file if the transmission from the remote node was not valid or ended prematurely. In this case, the link driver task indicates in the RDEVBLK that the file should be closed and purged. If the file is not to be purged, the AXM task calls Exit 5. Exit 5 routines can create or modify accounting records for each file RSCS receives; see [z/VM: RSCS Networking Exit Customization](#) for more information.

The AXM task then determines if the file is eligible for second-level addressing. If the file is for a local user, the AXM task tags the output device with the text (origin time and the spool, node, and user ID) that is expected when RSCS delivers the file. If, however, the file is to be queued on another link for transmission, the AXM task places a store-and-forward tag on the device. If the link driver task has built an XAB for the file, the XAB is written to spool by Diagnose code X'B4'. The AXM task then spools and closes the device and returns storage for the output TAG element and the FIOA.

OPENINTA Requests

Networking link driver tasks make OPENINTA requests to initialize the transmission algorithm for a multistreaming link. (See [z/VM: RSCS Networking Exit Customization](#) for more information about transmission algorithms.)

The AXM task calls the transmission algorithm, which can reside in DMTAXA or in a separate exit load library, for the link. The transmission algorithm is initialized by passing it an open request and the LINKTABL for the appropriate link. The transmission algorithm will have an opportunity to process each file enqueued on that link when a reorder is initiated in response to the link going into "connect" status.

Serializing Resources

Most spool file resources cannot be used by two tasks at the same time. RSCS serializes the resource to ensure that only one task (the AXM or a link driver task) uses these resources at any time. DMTRES manages the serialization of resources within module DMTAXM.

As part of its initialization process, the AXM task creates a RESBLOK to represent the spool interface resource (DMTAXMRQ) and adds it to the global RESBLOK chain. Tasks then call DMTRESLO to gain exclusive use of the resource and call DMTRESUN to return use of the resource.

Managing the Unit Record Device Pool

RSCS attempts to reuse the devices it has previously defined. The CHANNELS statement specifies the channels RSCS can use for unit record devices. A 256 byte allocation map is created for each channel specified on the statement. Bytes in the allocation map can have the following settings:

X'FF'

No device defined

X'00'

Device defined and in use

X'xx'

Device defined and free (code xx indicates type of device).

When the AXM task receives a request to define a new device, it looks for any matching device type that was previously defined. If it finds a device, the AXM task passes it to the caller. If a free device of the requested type does not exist, the AXM task searches for an address where no device is defined. If an address is free, the AXM task issues a Diagnose code X'08' to define the requested device. If no addresses are available, the AXM task looks for an address where a device is defined but not in use. It then detaches the existing device at that address, defines the new device, and returns its address to the calling task.

The AXM task, in turn, does not usually detach the device that the caller passes to it. The calling task can issue a request to force the device to be detached. This request indicates that the task has closed the device and wants to return it to the pool of free devices.

If a virtual reader, punch, 1403 or 1443 printer is being detached, RSCS does not issue a DETACH command. Other tasks may reuse these devices. However, some devices (such as 3211 or 3800 printers)

may have FCBs loaded and other tasks cannot reuse the device. In this case, the device is always detached.

Auto-Start Task

The auto start task, AST, manages the pool of auto-dial ports. It also monitors ITO for any link task and RETRY intervals for SNA and auto-dial links.

Initialization

The REX task attaches the auto-start task during RSCS initialization. After it sets up its ESTAE exit, the AST task waits on the following ECBs:

- Dynamic port allocation request
- Command
- Port free
- Timer
- Termination.

Dynamic Port Allocation

The spool manager task, and the AST task, post the dynamic port allocation ECB when a request element is placed on the DMTASTSQ queue anchor. This queue contains start request elements for dial-out links that have not been assigned a dial-out port. These links may be started because a file has been enqueued on the link or they are eligible for an auto-start. They may also be started if their RETRY interval has expired.

In each case, the AST task obtains exclusive use of the PORT table and searches for an available dial-out port to assign to the link. If no dial-out ports are available, AST marks the link in the *dial-queue* state until a dial-out port is freed.

If an operator or an exec issues a START command for an auto-dial link that requires a dynamically-allocated port, the AST task does not process the request. Rather, DMTLAXEP attempts to allocate a port. If no ports are available, the START command fails and the command originator is notified. The link is not placed in the dial-queue state and the AST task does not attempt to start the link when a port is freed.

Processing Commands

The AST task processes the FORCE, ITO, and RETRY commands for auto-start links. These commands are represented by the Type C0, Type C1, and Type C2 formats, respectively, of the CMNDAREA.

If a FORCE command is issued for a link that was attached from the AST task, the REX task passes the command to the AST task. This enables the AST task, which attached the link driver task, to detach and terminate the link.

The ITO command, an internal RSCS command element, is enqueued to the AST task by any link that has been defined with a nonzero ITO interval. When a link initializes, it enrolls itself into the ITO process. The link driver task then resets the inactivity time-out value in its LINKTABL when it executes an open request for an input or output file. The AST task monitors the ITO value to determine if the link should be deactivated.

The RETRY command, another internal RSCS command element, is enqueued to the AST task by the end-of-task routine (DMTMANEX). The AST task receives this command element when an SNA or auto-dial link deactivates with an error condition that makes it eligible for a retry. The AST task monitors the number of consecutive retry attempts on the link. It then copies the appropriate retry interval in to the link's LINKTABL entry. The AST task then attempts to start the link after the retry interval expires. The retry intervals default to 1, 10, 19, 27, 34, 40, 45, 49, 52, 54, and 55 minutes. These values can be overridden by the RETRY statement, described in [z/VM: RSCS Networking Planning and Configuration](#).

Free Ports

When a link that has been assigned a dynamically allocated port ends, DMTMANEX posts the free port ECB for the AST task. This tells the AST task that a dial-out port is now available. The REX task may also post the free port ECB when a PORT command defines a dial-out port.

When this ECB is posted, the AST task determines if any links are held in dial-queue status. If it finds a dial-queue link, the AST task assigns the free port and attaches the link driver task.

Timer ECB

When the first link enrolls itself for ITO processing or a link goes into a retry wait state, the AST task starts to maintain a timer. This timer is set at 1 minute intervals. When the interval expires, the AST task determines if it should deactivate any ITO links. It then monitors any links that are in a retry-wait state to determine if a start request should be issued for a link.

The AST task checks if any ITO links are currently processing a file. If a link is active, the AST task does not change the ITO value in the link's LINKTABL. If it is not active, the AST task decrements the ITO value in the LINKTABL by one minute. Each time a link requests to open a file, however, the OPENIN and OPENOUT routines in DMTAXM reset the LINKTABL field to the link's original ITO value. When the inactivity time value reaches 0, it indicates that the link has been idle for the number of minutes specified on the ITO operand. The AST task then posts the link's termination ECB.

If a link is in *retry-wait state*, the AST task also decrements its retry wait interval by one. When this interval reaches zero, the AST task attempts to restart the link.

For SNA links or dial-out links that have dedicated dial-out ports, the AST task passes a START command to the REX task. If, however, a dial-out link requires a dynamically allocated dial-out port, the AST task receives the START request. In this case, the link may be placed in a dial-queue state until a dial-out port becomes available.

In each case, the AST task increases the count of consecutive retry requests performed on the link. The AST task can then determine the next retry interval to use when it receives a RETRY command request.

Termination

The AST task ends when a SHUTDOWN command is issued or if an abend occurs in another RSCS system task. In each case, the AST task's termination ECB is posted. The AST task then ends and returns control to GCS.

Event Manager Task

The event manager task, EVE, issues RSCS commands and causes other types of events to occur at predefined times.

Initialization

After it has identified its ESTAE exit, the EVE task reads the EVENTS file; it then schedules any events that are to take place before midnight.

Scheduled events are represented by EVEBLOKs (for the EVEBLOK layout, see [“EVEBLOK” on page 225](#)), which form a chain anchored at the TEVENTS field in the CVT. This chain is maintained in ascending order, based on the next time each event is to be executed (see [“EVEBLOK” on page 31](#)). After it processes the EVENTS file, the EVE task sets a timer that expires when the event represented by the first EVEBLOK is scheduled to be processed.

Allocating Task IDs

Each EVEBLOK is assigned a unique task ID. The EVE task refers to the task ID to delete, suspend, or resume an event. The EVE task uses a task ID number allocation block (TANBLOK). Each TANBLOK

contains 1024 valid task IDs. If more than 1024 events are scheduled, the EVE task obtains storage for additional TANBLOKS.

Event Types

Events can be scheduled to execute once, at a specific time, or repetitively. Repetitive tasks are executed every *nn* minutes or at a certain time, *mm* minutes, past the hour. The EVEBLOK for a repetitive event retains its task ID until midnight. It does not receive a new task ID each time the event occurs. An event can be specified in the EVENTS file or by a SCHEDULE command.

System Events

All events specified in the EVENTS file are called *system* events. These events can be scheduled to be executed on a day of the week. They can also be scheduled to execute on a special day defined by a SPECIAL record in the EVENTS file. If the scheduled time for an event has already passed when RSCS reads the EVENTS file, the event is not scheduled for that day.

User Events

Events specified by the SCHEDULE command are called *user* events. These events are usually only executed on the day the SCHEDULE command is issued. However, if you specify the DAILY operand, the EVEBLOK for the event is added to the EVEBLOK chain for the following day.

Midnight Event

The EVE task creates the *midnight* event to ensure that it reads the EVENTS file at midnight. The EVE task can then determine the events that are to be scheduled for the next 24 hours. When it finds the midnight event, the EVE task creates a new chain of EVEBLOKs to represent the next day's system events. It then merges this chain with any user events (those that specified the DAILY operand) that are to be carried over to the next day.

Timer Management

The EVE task calls the GCS STIMER macro to ensure that GCS notifies it when the next event is to be executed. The event is executed as the EVE task compares the scheduled time of the event to the current system time.

When the timer expires, GCS schedules the TIMERPOP routine, which posts the timer ECB and returns to GCS. The EVE task then dequeues the EVEBLOK for the scheduled event from the EVEBLOK chain and executes the requested command. If the EVEBLOK represents a repetitive event, the EVE task updates the time of next execution in the EVEBLOK and places it back in the chain.

Processing SCHEDULE Commands

The REX task calls DMTCOMNQ to enqueue an encoded version of the SCHEDULE command element to the EVE task. This command element is a version of the EVEBLOK; it is not described by the CMNDAREA. The EVE task calls DMTCOMDQ to dequeue the command element. If the command schedules a new event, the EVE task creates a new EVEBLOK and adds it to the EVEBLOK chain. The EVE task then returns the task ID assigned to the EVEBLOK to the originator of the SCHEDULE command.

If the command element does not represent a request to schedule new events, it may be a request to manipulate existing events. These requests are specified by the various operands of the SCHEDULE command, which are described in the following sections.

DISKLOAD Operand

The DISKLOAD operand tells the EVE task to process the EVENTS file again when changes have been applied to it. (If the EVENTS file has been changed while RSCS is running, the RSCS virtual machine must have accessed the latest information on the disk where the file resides.) The EVE task deletes all existing

system events from the EVEBLOK chain. It then creates a new chain of system events and merges the new chain with any user events that remain on the EVEBLOK chain.

DELETE Operand

When the DELETE operand is specified, the EVE task deletes an existing event. The delete request can specify a specific task ID or a task name for those events that have a common task name. The EVE task removes all EVEBLOKs that represent the specified tasks from the EVEBLOK chain.

SUSPEND Operand

The SUSPEND operand causes the EVE task to temporarily suspend one or more existing events. The suspend request can specify a specific task ID or a task name. For this request, the EVE task marks all specified EVEBLOKs as suspended. The EVEBLOKs remain on the EVEBLOK chain. However, the EVE task does not execute the event until the RESUME operand has been specified.

RESUME Operand

When the RESUME operand is issued, the EVE task allows any events that were previously specified on the SUSPEND operand to be executed at their next scheduled time.

EXEC Processor Task

The EXEC processor task, EXE, calls execs for other RSCS tasks. The EXE task allows execs to call RSCS commands.

Initialization

During RSCS initialization, the REX task attaches the EXE processor task. The EXE task then establishes its ESTAE exit to process any abends.

Processing Exec Queues

The REX task and DMTMANEX call DMTCOMNQ to enqueue an exec request on the EXE task's exec queue. The EXE task calls DMTCOMDQ to remove each request from the queue. It then issues the GCS CMDSI macro to call the specified exec.

SNA Control Task

The SNA control task (SCT) maintains the RSCS/VTAM interface, which enables RSCS to use SNA to communicate with remote nodes. The SCT task activates and deactivates session drivers. It also processes any VTAM RECEIVE operations that cannot be handled by the session drivers. If no SNA links are defined, however, RSCS does not need the SCT task to function.

Establishing a Session

A logical unit (LU) is a system or device that uses a selected SNA protocol to communicate with another system or device. Before logical units can communicate with each other, they must establish a *session*. In a session, each LU agrees on a protocol to be used when exchanging data. The agreement is established by a *bind image*, which defines the rules that each LU follows when communicating with the other. If the LUs cannot agree on the bind image, the bind process fails and the SNA session is not established. After the session is established, however, either logical unit may request to end the session.

Logical Units

In SNA sessions, one LU acts as the *primary* and the other as the *secondary*. The primary LU can define the BIND image for the session. The LU that initiates the session chooses its role (primary or secondary) in the session. For SNANJE sessions, RSCS may act as the primary or secondary LU. For SNA3270P or

SNARJE sessions, RSCS can only act as the primary LU. RSCS defines the following types of session drivers:

Session	Driver	Function
SNANJE	DMTSNE	Peer to peer connection using the LU_T0 based SNA/NJE protocol. RSCS can be the primary or secondary LU in this connection.
SNA3270P	DMTSPT	Connection to 3270 printers (or a device that emulates one) by LU_T0, LU_T1, and LU_T3 protocols; RSCS is always the primary LU.
SNARJE	DMTSJE	Connection to the System/36 MSRJE facility by a subset of the LU_T1 protocol; RSCS is always the primary LU.

SIMLOGON

When an LU wants to initiate a session and become the primary LU for that session, it issues a SIMLOGON macro. The SIMLOGON request is first handled by the requesting LU’s system service control point (SSCP). The SSCP arranges sessions between eligible LUs that are within or outside the domain it governs. If the secondary LU is outside the SSCP domain, the SIMLOGON request is passed to the SSCP that governs the domain in which the secondary LU resides. If the SSCP grants permission, a control initiate request (CINIT) is sent to the initiating LU (which drives its LOGON exit). The CINIT requests that the LU send a BIND request to the secondary LU (this drives the secondary’s SCIP exit).

If the secondary LU accepts the BIND image, it sends a positive response to the primary LU (see [Figure 38 on page 56](#)). The BIND image presented by the RSCS session drivers generally is not negotiable. However, the SNANJE session driver, when acting as the secondary LU, will tolerate some negotiation.

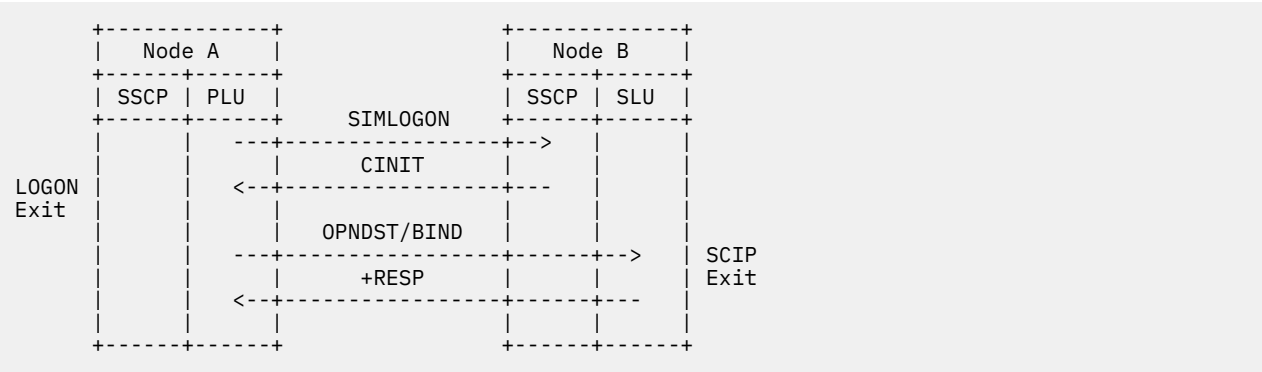


Figure 38. A Successful BIND Request

If the secondary LU does not accept the BIND image, it sends a negative response to the primary LU (see [Figure 39 on page 56](#)).

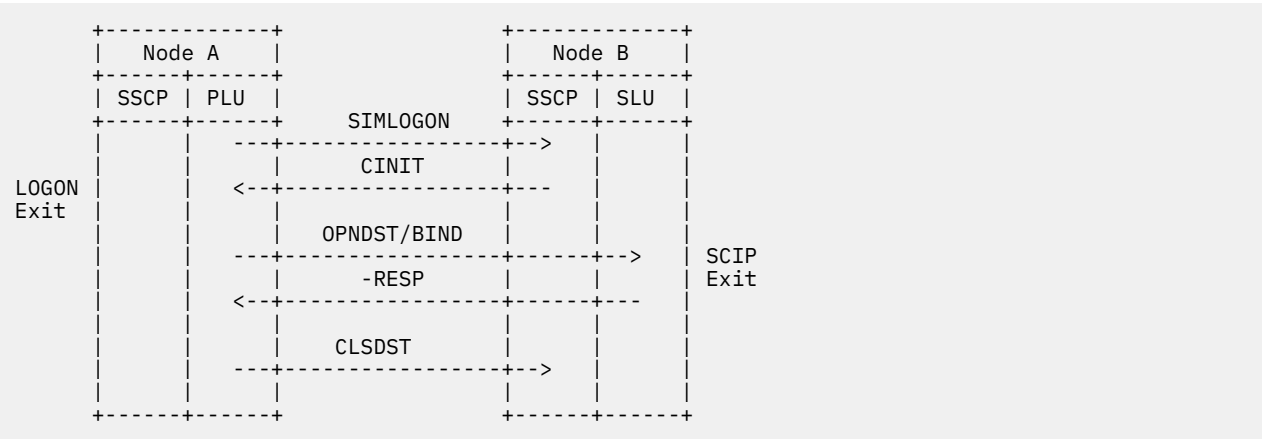


Figure 39. An Unsuccessful BIND Request

Initializing the SNA Control Task

The REX task does not attach the SCT task when RSCS initializes; rather, the SCT task is attached when a NETWORK START command is issued. The SCT task ends when a NETWORK HALT or SHUTDOWN command is issued. If you do not have SNA connections within your RSCS network, the SCT task does not need to be started.

To initialize the RSCS/VTAM interface, the SCT task obtains storage for parameter lists and data areas that are passed to VTAM when sessions are started. The SCT task also opens communications between RSCS and VTAM and prepares to receive any requests that may be waiting for RSCS.

The SCT task monitors incoming requests from VTAM for any session driver or for the RSCS application itself. It also ensures that VTAM can run certain exit routines provided in RSCS. These exit routines become available when the SCT task initializes and opens the ACB (see [“VTAM Exit Routines”](#) on page 57 for more details on the ACB task).

Request Parameter Lists

RSCS uses a request parameter list (RPL) to make an SNA request. The RPL contains information and data areas, which VTAM uses to process the request. When RSCS processes a NETWORK START command for a session driver, it generates a SIMLOGON request, which requires an RPL. The SCT task acquires storage for all SIMLOGON RPLs and associated data areas. The number of RPLs acquired is determined by the RPLS option of the NETWORK START command.

The following data areas are associated with an RPL; their interactions are described in [“Maintaining the RSCS/VTAM Interface”](#) on page 59:

ECB

Posted by VTAM when the SIMLOGON request completes; one ECB is required for each SIMLOGON RPL. Pointers to the ECBs are also passed to VTAM with the SIMLOGON request. Additional ECBs (terminate, receive and command) are used within the SCT task.

Flag bytes

Used as an allocation map for the RPLs used by RSCS.

NIB

The node initialization block contains information that RSCS provides VTAM about general session characteristics (such as, LU name and BIND information).

Access Method Control Block

The access control block (ACB) gives VTAM information about a VTAM application program (in this case, RSCS). It assigns a name to the program and lists the exit routines associated with it. This ACB must be opened before any application can interface with VTAM. The VTAM OPEN macro tells VTAM that the application is running. VTAM can then accept requests for sessions and schedule (or call) any exit routines.

VTAM Exit Routines

DMTVXT contains the exit routines VTAM requires for the RSCS application and the SNA session. Entry points in DMTVXT correspond to a specific VTAM exit routine, which are described in the following sections.

LOSTERM

DMTVXTLT is scheduled when a session is abruptly ended or disrupted. This routine searches LINKTABL entries to find a matching communication identifier (CID) and posts the LTERECB for that link. VTAM generates the CID when a SIMLOGON request is issued and returns its value in the NIB provided by RSCS with the SIMLOGON request. When the SIMLOGON completes, RSCS stores the CID in the LINKTABL for the session driver.

LOGON

DMTVXTLG is scheduled when a SIMLOGON request, initiated by the SCT task in response to a START command, completes. It is also scheduled when RSCS receives a VARY LOGON request from VTAM. DMTVXTLG searches LINKTABL entries for the appropriate LU name and attaches the appropriate session driver task.

If any of the preceding steps fail, DMTVXTLG cleans up the LINKTABL entry (if necessary) and issues a CLSDST request for the session driver. CLSDST tells VTAM to end a session or reject a CINIT between RSCS (the primary LU) and the secondary LU.

To attach the session driver, DMTVXTLG calls the VXTATTH subroutine. This routine acquires storage for a task initialization vector. It then calls DMTBPLAL to attach the session driver task. It also initializes the LINKSTAT control block, which contains data used for QUERY command responses. At this point, the session driver is considered to be *active*.

NSEXIT

DMTVXTNS is scheduled when RSCS receives the following Network Service requests from VTAM:

Clean Up

Called when a session, driven by RSCS, has been ended without prior warning to RSCS.

Notify

Called when VTAM tells RSCS that it cannot successfully process a request. If RSCS has not violated any defined procedures for session initiation, it may choose to retry the request.

Procedure Error

Called if VTAM detects a procedure error when a session starts or stops. This request (NSPE) can also be issued as a Notify request if RSCS has not asked to be notified when a request completes.

With each NSEXIT request, VTAM passes an LU name to RSCS. DMTVXTNS finds the LINKTABL with the matching LU name and posts that link's terminate ECB. It then processes the rest of the LINKTABL and disables auto-start of the link. The routine also determines if it should restart the session, based on the RETRY option specified either on the LINKDEFINE statement or the START and DEFINE commands.

RELREQ

DMTVXTRL lets RSCS share printers and workstations with other VTAM applications. This routine is scheduled when another VTAM application requests an LU that is currently in session with an RSCS SNA3270P or SNARJE session driver. This exit routine finds the LINKTABL entry that contains the matching LU name and posts the LRELECB for that link. The session driver only gives up the session if there are no active files being processed. When its LFILECB is posted, indicating that a file has arrived on the link, the session driver can issue a SIMLOGON request to reacquire the SNA session.

SCIP

DMTVXTSC is scheduled when RSCS receives a Session Control request from VTAM. DMTVXTC processes BIND requests by calling VXTATTH to attach a session driver task as a secondary LU. However, DMTVXTSC passes the following requests directly to the appropriate session driver:

CLEAR

Issued by the primary LU to stop the flow of session requests

RQR

Issued by the secondary LU to request sequence number synchronization

SDT

Issued by the primary LU to start or resume the flow of session requests

STSN

Issued by the primary LU to set and test sequence numbers

UNBIND

Sent by either LU to end a session, usually for unexpected outages.

TPEND

DMTVXTTP is scheduled when VTAM abruptly ends. This may occur when the VTAM operator issues a HALT command, VTAM quiesces because of an internal problem, or VTAM abends. This exit routine posts the SCT task's terminate ECB. Depending on the severity of the error, DMTVXTTP may queue a DRAIN command or post the terminate ECBs for all active session driver tasks. The SCT task then waits for the session drivers to end before completing its termination processing.

However, if an OPEN request fails, RSCS may make several attempts to retry the request. The number of attempts is specified on the RETRY option of the NETWORK START command. RSCS may retry the OPEN request under the following conditions:

- VTAM is not initialized
- VTAM does not have enough storage to open the ACB
- ACB is currently being closed
- Wrong password specified on the NETWORK START command
- APPLID on the NETWORK START command is incorrect or not found.

After the OPEN completes successfully, the SCT task issues a SETLOGON request. This request tells VTAM that the RSCS LOGON exit is ready to receive any CINITs for RSCS session drivers. When the SETLOGON completes, the RSCS/VTAM interface is ready to process data and the TGSSNAUP flag is set in the CVT.

Maintaining the RSCS/VTAM Interface

To maintain the RSCS/VTAM interface, the SCT task monitors and waits on three ECBs: command, receive-any, and terminate. It also waits on a list of ECBs associated with any outstanding SIMLOGON requests.

Before starting its main processing cycle, the SCT task watches for inbound requests destined for RSCS session drivers. The SCT task then issues a RECEIVE ANY request. The RECEIVE macro requests VTAM to transfer data or control information to RSCS's storage areas. This enables the SCT task to use the RSCS hashing routines to search for the matching CID and link; VTAM does not perform this search.

RECEIVE requests can be limited to specific sessions (identified by CID) or to a group of sessions (identified by the application ID assigned during OPEN ACB processing). The RECEIVE ANY parameter tells VTAM that this request applies to all of the RSCS session drivers.

RECEIVE Processing

RSCS generally uses RECEIVE ANY requests for SNA3270P and SNARJE session drivers. The SNA3270P driver does not expect to receive data on a regular basis as opposed to the SNANJE driver which does expect to receive regular data. The SCT task requests the RECEIVE function for the SNA3270P and SNARJE session drivers.

The first time an SNANJE session driver satisfies the SCT task's RECEIVE ANY request, the session driver initiates and maintains a RECEIVE SPECIFIC/CONTINUE SPECIFIC for the duration of the session. VTAM passes any RPLs for that session (identified by CID) to the session driver.

When the SCT task's receive ECB is posted, the CID is retrieved from the RPL passed by VTAM. LINKTABL entries are searched sequentially, unless the NIBUSER field already contains a pointer to the LINKTABL. The NIBUSER field is filled in as the session driver initializes. NIBs are usually associated with an RPL for most VTAM requests. Making use of this facility eliminates the need for sequential searches. The session driver's receive ECB is then posted.

However, RSCS does not retrieve any data from VTAM at this point. VTAM does not transfer the data to RSCS's private storage until the session driver issues a RECEIVE SPECIFIC request. After this request completes, the session driver task issues RESETSR to place the session in CONTINUE ANY mode. This allows the SCT task's RECEIVE ANY request to remain eligible to receive another RPL from VTAM. The SNANJE session drivers must then issue and maintain their own RECEIVE SPECIFIC/CONTINUE SPECIFIC after this RESETSR.

Command and SIMLOGON Processing

When a START command is issued for an SNA link, DMTCMY queues the command element to the SCT task and posts its command ECB. The SCT task then searches the RPL allocation map to find an RPL for the SIMLOGON request. Each byte in the allocation map corresponds to a SIMLOGON RPL, a NIB, an ECB, and an ECB pointer. If it finds a free RPL, the SCT task marks its corresponding byte in the allocation map. It then uses the associated data areas for the SIMLOGON.

If no RPLs are available, the SCT task enters *RPL starvation mode*. This means that RSCS cannot process any START commands for SNA-type links until an outstanding SIMLOGON completes. Any session drivers that are started while RSCS is in RPL starvation mode remain in the “RPL-WAIT” state. When a SIMLOGON RPL becomes available, the session driver enters the “STARTING” state.

When a SIMLOGON completes, VTAM posts the SIMLOGON ECB and the status changes to “LOGON WAIT”. VTAM then schedules DMTVXTLG to attach the session driver task and DMTVXT will also increase the count of active SNA links in the CVT. The SCT task then marks the allocation byte for that SIMLOGON RPL and returns the ECB, its pointer, and the RPL for use by another SIMLOGON. If the SCT task is in RPL starvation mode, it also posts its command ECB because it can now work with an RPL and associated data areas. If not in RPL starvation mode, the SCT task continues to monitor and wait on each ECB.

Termination

The REX task posts the SCT task’s terminate ECB when a SHUTDOWN or NETWORK HALT command is issued, or from the TPEND exit (DMTVXTTP).

When the ECB is posted, the SCT task starts to close the RSCS/VTAM interface to prevent VTAM from driving additional LOGON or SCIP exit routines. When ending as usual, the SCT task issues a SETLOGON request with the QUIESCE option. When an abend occurs (for example, the terminate ECB is posted from the TPEND exit), the SCT task issues a CLOSE ACB request. The CLOSE macro tells VTAM that RSCS is ending its association with VTAM.

The SCT task then sets the VTAM interface shutdown in progress flag (TGSVSIP) in the CVT to prevent RSCS from starting more session drivers. Each session driver’s terminate ECB is posted and the SCT task waits on its terminate ECB until all session drivers terminate. Depending on the severity of the termination, TPEND or normal, the links may quiesce (drain) or abruptly end (STOP). DMTMAN then decrements the count of the active SNA session drivers. When that count is zero and the TGSVSIP flag is on, DMTMAN posts the terminate ECB for the SCT task. The SCT task calls DMTCOMDQ to dequeue any commands and then ends.

SNA Session Cleanup

When a session driver task ends abnormally, GCS schedules the ESTAE exit and the general end of task exit, DMTMANEX. In turn, DMTMANEX, calls the session cleanup routine, DMTSCTCU, for the SCT task. DMTSCTCU performs the necessary session cleanup for the session driver that abends. If the session driver is the primary LU, DMTSCTCU issues a CLSDST request. If the session driver is the secondary LU, it issues a TERMSESS request.

Port Redirector Task

The port redirector task (PRD) performs LISTEN calls on specific TCP/IP ports on behalf of TCPNJE links to wait for incoming connect requests from a specific remote IP address. User-defined TCP/IP link drivers can also use PRD service. When a connect request is received from a remote host that matches the remote IP address specified by the calling task, the PRD task will transfer that connection to the requesting task.

Initialization

The REX task attaches the port redirector task during RSCS initialization. The port redirector task can also be stopped, and restarted, by the TCPIP command (see *z/VM: RSCS Networking Operation and Use*). When started, the PRD task waits to receive work from TCPNJE link driver tasks.

The PRD task can support up to 16 active TCP/IP virtual machines with a maximum of 64 active ports for each virtual machine. It uses one IUCV connection to communicate with each TCP/IP virtual machine.

Processing LISTEN Requests

The PRD task and the calling task use PRDBLOKs to communicate with each other. The PRDTYPE field in the PRDBLOK is used to post requests and responses between the tasks. PRDTYPE may have the following values:

Values	Function
PRDADD	Requests to listen for a remote IP address on a specific local port
PRDDEL	Requests to cancel a previous LISTEN request
PRDERR	Indicates an error response
PRDGIVE	Give socket reply response

Starting and Canceling LISTEN Requests

When a task wants to listen for a connect from a specific host, it posts a PRDADD request to the PRD task. To cancel an existing request, the calling task posts a PRDDEL request. The calling task also updates PRDBLOK fields with the following information; the task then calls DMTPRDNQ to enqueue the PRDBLOK to the PRD task:

PRDTYPE

PRDADD indicates an add request, or PRDDEL to cancel a request

PRDTCPID

The user ID of the TCP/IP virtual machine

PRDTASK

Task name of the calling task

PRDCLIEN

Client ID of the calling task; returned by the GETCLIENTID socket call.

PRDSOCKA

Socket addressing structure, which includes:

- Addressing family (AF_INET)
- TCP port number to listen on
- The IP address of the remote host for which the calling task is waiting to receive a connect request.

Receiving a Reply

After the request PRDBLOK has been posted, the PRD task issues SOCKET functions to allocate a socket and bind the socket to the port specified by the calling task. It then starts a LISTEN call for a connect request on that port for the remote IP address specified by the calling task.

When the LISTEN request successfully completes, the PRD task issues a GIVESOCKET request, using the information (domain, user name, and subtask name) it received from the calling task. The PRD task then posts the calling task's LPRDECB. It also updates the PRDTYPE field in the PRDBLOK to indicate the type of response it received:

- If this is a PRDGIVE response, a matching connection has arrived for this task. The PRD task updates the PRDCLIEN and PRDSOCKN fields in the PRDBLOK. The calling task can then use this information to issue a TAKESOCKET socket call to accept the connection.
- If this is a PRDERR response, the PRDERRNO field in the PRDBLOK will contain a TCP/IP error number. The error number is defined in the SOCKET macro and can be obtained by using the DSECT function (see [“SOCKET”](#) on page 220 for the SOCKET macro layout).

The calling task then calls DMTPRDDQ to receive the response from the PRD task. It then issues a TAKESOCKET request to accept the connection.

The port redirector task will continue to listen for connect requests on the port for the task. If the PRD task receives another request while the calling link driver task is in active state, it indicates that the remote side of the link has gone down and is restarting. In this case, the calling link driver task was unaware the connection had ended. When the calling link driver task issues the TAKESOCKET request, the link will terminate and restart to establish a new connection.

Requests from a task remain queued by the port redirector task until the task posts a delete request or terminates. The PRD task periodically checks for requests for tasks that have ended. A request also remains queued if an error is detected. The PRD task also periodically retries LISTEN calls for ports. Error responses are posted to the requesting task only if the error number changes.

The TCPIP command can also be used to trace the port redirector task. Information that is captured by tracing the task includes IUCV connect information and specifics on socket calls PRD issues.

Termination

The port redirector task is terminated when a SHUTDOWN command is issued or when the TCPIP STOP command is issued. The task may be restarted by the TCPIP START task.

Auto-Answer Tasks

An auto-answer task (DUP) manages an auto-answer port. (The auto-answer task is also known as the dial-up task because the remote caller dials in to RSCS.) Unlike other system tasks, many auto-answer tasks can be present in the RSCS virtual machine. Auto-answer tasks are not attached when RSCS initializes. They are attached when an ENABLE command is issued for a port that has been defined with the NODIAL option. Auto-answer tasks are detached when RSCS ends.

The DUP task monitors ports for incoming phone calls. When RSCS receives a phone call, the caller sends a sign-on record. Using this record, the DUP task identifies the caller's link ID. If the requested link type has been defined to RSCS, the DUP task transforms itself into the link driver task for the specified link.

Initialization

As [Figure 40 on page 62](#) shows, the DUP task is attached when an ENABLE command is issued. The REX task passes the ENABLE command element to DMTCMZ, which attaches the DUP task. The DUP task performs all required processing.

```

+-----+ +-----+ +-----+ +-----+
| ENABLE +---->| DMTCMZ +---->| DMTCMZ +---->| DMTCMZ +---->|
+-----+ +-----+ +-----+ +-----+

```

Figure 40. Attaching the Dial-Up Task

When the DUP task receives control, it executes the GCS ESTAE macro to establish theabend exit routine. It also obtains required storage and initializes a work area for each dial-up task.

The DUP task then starts a channel program for the dial-up port. The channel program enables the modem to receive an incoming phone call. When the modem detects a call, the channel program completes and data can be transferred over the phone line.

Identifying Callers

When a call completes successfully (the modem detects a valid signal carrier from the other modem), the DUP task uses binary synchronous (BSC) protocols to identify the caller. The DUP task issues ACK or NAK control characters in response to any BSC information received from the modem, until a data buffer is received (indicated by a STX control character). When the call is detected, the DUP task sets a 5 minute timer and waits for a data buffer to arrive.

Processing Sign-On Records

The DUP task assumes that the data buffer it receives is a sign-on record that is recognized by a BSC link driver (RJE, MRJE, or NJE). The beginning of the buffer is scanned to determine if it is an NJE or MRJE format sign-on record or if it starts with the word “SIGNON”. If so, the target system name is extracted from the sign-on record. The DUP task then looks for an appropriate type of inactive RSCS link that connects to the calling node.

Invoking Links

The DUP task gives final processing of the sign-on record, and further processing on the BSC link, to the appropriate link driver task. The DUP task updates the LINKTABL entry for the link so that the link appears to be active. It then calls the GCS LINK macro to branch into the link driver code.

The DUP task regains control in the following situations:

- The link completes its processing
- The link is drained
- The remote node signs off
- The link detects an incorrect password
- No password is specified on the link parameters.

The DUP task ends and returns control to the REX task. DMTMANEX then must issue the ENABLE command to enable the port to receive more calls.

Error Processing

The DUP task must process hardware errors and data that is not valid, including indications of possible misuse of the port.

The DUP task follows bisynchronous protocol to process hardware errors. If the error is not severe, the DUP task sends a NAK control character. This character requests that the calling node repeat its last transmission. If a severe error occurs, the DUP task ends the phone connection; it may also request that the REX task re-enable the port.

If it receives invalid data, the DUP task ends and requests the REX task to re-enable the port. Invalid data may include: an incorrect sign-on buffer, unknown link ID, attempts to sign on to an active link, and failure to send a sign-on record within 5 minutes of placing the call.

The DUP task also recognizes attempted misuse of a port. If a caller makes 5 consecutive attempts to send an incorrect sign-on record to a port, the DUP task ends; it does not request that the REX task re-enable that port.

Calling Exit Points

The following IBM-defined exit points let you modify or monitor the processing of the DUP task:

Exit 7

Sign-on time limit expiration

Exit 8

Unrecognizable data

Exit 9

Sign-on validation

Exit 10

Sign-on reject.

You can use these exit points to create accounting records to monitor the use of ports at your installation. You can also use Exit 9 to specify additional restrictions for using auto-answer ports. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Chapter 6. Networking Link Drivers

This chapter describes the link driver tasks that enable RSCS to send and receive files, messages, and commands from other nodes in the network. For more details on NJE, see [z/OS: Network Job Entry \(NJE\) Formats and Protocols \(https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf\)](https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf).

Common Networking Structures

This section describes the RSCS structures and routines that are common to all networking link drivers (NJE, SNANJE, and TCPNJE) and the LISTPROC link driver. The GATEWAY link driver may also use some of these structures and routines.

Data Areas

The key data area for NJE protocol based link drivers is the Networking Dynamic Work Area (NDWA). The NDWA, passed in register 8 to all the NJE modules, contains the following information:

- Pointer to chain of RIBs (Receiver Information Blocks)
- Pointer to chain of active and inactive TIBs (Transmitter Information Blocks)
- MSGBLOK, for issuing messages
- RDEVBLOK
- Flags, indicating the type of link driver and supported options
- Additional work areas.

Building NJE Headers

Networking link drivers call routines in module DMTNHE to build NJE headers for files sent on the link. These NJE headers are built from information in TAG elements and TIBs. RSCS processes the following types of NJE headers:

Job header

Indicates the file's origin and tentative destination

Data set header

Contains the destination and characteristics about the file

Job trailer

Ends the transmission.

Building the Job Header

DMTNHEJH builds job headers from information in the TAG element and TIB. The TIB also indicates if the header is for a SYSIN or SYSOUT file.

The origin and destination fields in a SYSOUT job header are reversed (or swapped) of the fields used for a SYSIN job header. Also, SYSIN streams from some MVS systems may not contain a data set header. If present, the data set header will only contain a Record Characteristics Change Section (RCCS). RSCS does not generate an RCCS section.

When the job header has been built, DMTNHE calls Exit 11. Exit 11 routines can add a user section to the job header. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Building the Data Set Header

DMTNHEDH builds data set headers for all SYSOUT files originating on the local node or processed by a list processor on the local node. This data set header contains the following information:

- General section
- RSCS section of the data set header, which includes:
 - Origin user tag
 - File name and file type of the file
 - Version and release number of RSCS that created the header
- 3800 printer characteristics section (if applicable)
- Data stream characteristics section (if applicable).

The TAGSCAN routine in DMTNHE updates fields in the data set header, based on options specified on the origin user tag. When the data set header is built, DMTNHE calls any Exit 12 routines that can add user sections to the header. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Building the Job Trailer

RSCS calls DMTNHEJT to build a job trailer, which identifies the end of the file transmission. It then calls Exit 13 routines, which can add user sections to the job trailer.

Receiving NJE Headers

When a networking link receives a file, it calls routines in DMTNHD to process the NJE headers associated with the file.

Reconstructing Headers

If a file's NJE header is larger than 256 bytes, the system that sends the file segments the header. When RSCS receives the file, it must reconstruct the NJE header before it can process the file.

DMTNHDHR collects the segments that make up an individual NJE header. As it receives the header segments, DMTNHDHR determines if any segmentation errors have occurred. When it receives all sections of the header, DMTNHDHR tells the calling link driver that the header is complete. It also indicates the type of header it received. The RIB contains a pointer to the data area that contains the NJE header. The calling routine must then pass the complete header to the appropriate processing routines in DMTNHD, which are described in the following sections.

Receiving the Job Header

DMTNHDJH scans information in the reconstructed job header and places it in a TAG element provided by the calling link driver task. Only the information that RSCS needs to process the file is kept accessible in the TAG element. If RSCS is processing a store-and-forward file and VAFP=NO is specified for the link, the complete job header information is written to spool using CCW NOP opcodes. If VAFP=NO is not specified, the information is written to spool using a VAFP device.

Before updating the TAG element with information from the received job header, DMTNHD calls Exit 14. Exit 14 routines can alter information in the job header, record information in the TAGUSER field, or reject the file. Any changes made to the job header at Exit 14 can affect values in the eventual TAG element. After updating the TAG element, DMTNHD calls Exit 41. Exit 41 routines can alter information in the TAG element after RSCS has extracted the job header information. See [z/VM: RSCS Networking Exit Customization](#) for more information about Exit 14 or Exit 41.

Receiving Data Set Headers

DMTNHDSH scans information from the reconstructed data set header and updates the TAG element provided by the calling link driver task. This is the same TAG element that was initialized by DMTNHDJH. However, information in the data set header may override the information in the job header (for example, the destination node and user ID).

Before updating the TAG element with information from the received data set header, DMTNHDSH calls Exit 15. Exit 15 routines can alter information in the data set header, record information in TAGUSER field, or reject the file. Any updates made to the data set header in Exit 15 can affect values in the TAG element RSCS creates to represent the file. After updating the TAG element, DMTNHD calls Exit 42. Exit 42 routines can alter information in the TAG element after RSCS has extracted the data set header information. See [z/VM: RSCS Networking Exit Customization](#) for more information about Exit 15 or Exit 42.

Multiple Data Set Headers

For LISTPROC-type links, RSCS may combine store-and-forward data sets from several files into one file. This may occur if the data sets are sent on the same link and if they specify the same device type.

RSCS uses different criteria for combining data sets for files that are sent to a user on the local node or a printer attached to the local RSCS virtual machine. RSCS only combines these data sets if each file specifies the same node ID, user ID, class, copy count, translation tables, form name, and external writer name.

Receiving the Job Trailer

DMTNHDJT uses the information from the reconstructed job trailer to update the TAG element provided by the calling link driver task. This is the TAG element that was initialized by DMTNHDJH and DMTNHDSH.

Before updating the TAG element with information from the received job trailer, DMTNHD calls Exit 16. Exit 16 routines can alter information in the job trailer, record information in TAGUSER, or reject the file. Any updates made to the job trailer by an Exit 16 routine can affect the values in the TAG element RSCS creates to represent the file. After updating the TAG element, DMTNHD calls Exit 43. Exit 43 routines can alter information in the TAG element after RSCS has extracted the job trailer information. See [z/VM: RSCS Networking Exit Customization](#) for more information about Exit 16 or Exit 43.

Receiving and Transmitting NMRs

DMTNHD also contains routines that the networking link drivers use to send and receive nodal message records (NMRs).

RSCS calls DMTNHDMR to receive NMRs from remote nodes. DMTNHDMR translates the NMR into the appropriate Type L3 element for RSCS. For store-and-forward elements, any non-RSCS fields in the NMR are retained. DMTNHDMR then calls DMTRGX to process the resulting command element.

Networking link driver tasks call DMTNHDMT to send a message or command element to a remote node. DMTNHDMT then converts the Type L3 element into the appropriate NMR format.

General Purpose Routines

DMTNUS contains routines that networking link drivers use to process NJE records.

Compressing Records

DMTNUSCP compresses records presented in intermediate buffers (called TANKs) into the buffer provided by the calling link driver task. DMTNUSCP uses string control bytes (SCBs) to compress the data. SCBs can perform the following functions:

- Compress repeated characters (up to 63 at a time)
- Compress repeated blanks (up to 63 at a time)
- Repeat data as it appears (up to 63 at a time).

DMTNUSCP uses minimum compression when processing files sent on CTC, 3088 (BSC), ESCON®, FICON®, and TCPNJE links. It uses maximum compression for all other types of networking links. You can use the COMPRESS link operational parameter to change the default value for the NJE-type, SNANJE-type, and TCPNJE-type links. See [z/VM: RSCS Networking Operation and Use](#) for more information.

Decompressing Records

DMTNUSDC decompresses a record from the buffer, pointed to by the calling networking link driver, and places the result in a TANK. DMTNUSDC interprets all SCBs that were used by the remote system to compress the headers or records. If decompression errors occur, DMTNUSDC also passes an appropriate return code to the calling link driver. The link driver then issues a message to indicate that a decompression error has occurred.

Creating Coded NOPs

When a networking link driver receives a store-and-forward file, it calls DMTNUSCN. If the system option VAFP=NO was specified, this routine creates NOP records for the file's NJE header and any other records that RSCS may not be able to write to spool. DMTNUSCN then segments the records, if necessary, and writes them into a spool file that is created for the file. The NOPs contain information that enable RSCS to reconstruct the records when it sends the file to the next node.

If VAFP=YES was specified for the system option, the records are written to the spool device using the SRCB as the CCW opcode.

Creating NJE Headers from NOPs

DMTNUSDN reconstructs NJE headers and file records from NOP records. Before sending a file to its next node, networking link drivers call DMTNUSDN to reconstruct the records as they are read from spool.

Initializing Storage

When an NJE link driver initializes, it calls DMTNCRIN to initialize storage and various data fields. These areas and fields include:

- RIBs and TIBs
- TANKs
- NDWA fields
- MSGBLOK fields.

DMTNCRIN also calls DMTAXMRQ to make an OPENINTA request. This request initializes the transmission algorithm specified for the link. When the link driver activates, DMTAXMRQ also reorders the files on the link queues.

Processing Sign-on Records

DMTNCRSG processes and verifies the sign-on records received from remote nodes. Because the buffer size is determined through the sign-on records, DMTNCRSG also obtains storage for TP buffers. Networking link drivers use two types of sign-on records. I records describe the features supported by the local RSCS virtual machine. J records are in response to an I (initial sign-on) record and describe to the remote node the features supported by RSCS on the local node.

Processing Commands

When the command arrival ECB is posted in a networking link's LINKTABL entry, the link driver task calls DMTNCRCD. This routine calls DMTCOMDQ to dequeue the command element. The link driver can process the following commands: DRAIN, FLUSH, FREE, HOLD, START, and TRACE. DMTNCR contains additional routines to process each command.

Accounting

DMTNCR also contains routines that monitor accounting information on the NJE-type links.

DMTNCRCT counts the number of transactions performed on the link. These transactions include each I/O, VTAM, or TCP/IP socket requests performed by the link driver task. DMTNCREC counts I/O errors

(excluding VTAM errors) on the link. DMTNCRT0 counts the number of time outs on a link. However, it does not count the time outs that occur when no data is sent on session drivers during idle periods.

Transmitting Buffers

The link driver tasks (DMTLIS, DMTNET, DMTSNE, and DMTTNE) call DMTNTR when they need to send data in an available buffer.

Managing Output Buffers

Networking link driver tasks use two types of buffers to send data. Large buffers contain message and file data. Smaller buffers contain control characters and are also used as null buffers. If the mixed record control byte (RCB) feature is used, the control characters are sent in the large buffers. In this case, the small buffers only are only used on the links as null buffers.

When a networking link driver task calls DMTNTR, it obtains the first buffer from a queue of large, free buffers. If the mixed RCB feature is used, DMTNTR moves the data from any queued small buffers to the top of the large buffer. It then frees the small buffers for use as a null buffer. If needed, DMTNTR also places a receiver online flow in the buffer.

Processing Message Streams

If messages are queued to the link driver, DMTNTR calls DMTCOMDQ to obtain a message element. DMTNTR calls DMTNHDMT, which formats the element into an NMR, and attempts to place the NMR into the buffer. If the mixed RCB feature is not in use and the MSGSKIP parameter is specified for the link, DMTNTR may place the NMR into a special buffer reserved for messages. This buffer is not sent until it is full or until a certain number of buffers have been filled with file data. DMTNTR repeats this process until no more message elements are available or until the buffer is full.

Dispatching File Streams

When all other data is placed into the buffer (with mixed RCBs) or if no other data is available, DMTNTR determines the next transmission stream it will process. It first dispatches any inactive streams for which files are available. DMTNTR places each TIB, which represent each new file, on the active TIB chain according to the buffer space needed to send the file. DMTNTR then compares the remaining buffer space needed to send all currently active files. When a stream is dispatched, DMTNTR ensures that the other streams that contain active files are credited. This ensures that large files are also transmitted efficiently.

Processing Streams

When DMTNTR selects a transmission stream to dispatch, it places as much file data as possible into the buffer. DMTNTR calls DMTRDREP to read each record of the file. If the file does not have NJE headers, DMTNTR also calls DMTNHE to build the necessary header records to accompany the file.

If the file contains a distribution list and is being processed on a LISTPROC-type link, DMTNTR calls DMTLCR and DMTNHE to convert each list entry onto a data set header. For store-and-forward files, DMTNTR calls DMTNUS to reassemble any NJE header, spanned, or other stored records from NOP spool records.

If a record contains record segments that are longer than 256 bytes, DMTNTR must perform additional processing, called *spanning*. Each record is divided into segments containing a maximum of 254 bytes. DMTNTR then sends these segments with spanned record indicators in the SRCB.

If a networking link driver receives a receiver cancel or negative permission response from the remote node, DMTNTR processes the file and associated TIBs appropriately.

Receiving Buffers

DMTNRV unpacks the buffers received from the remote node. It is called after any I/O, VTAM READ, or SOCKET RECEIVE requests complete on the link driver. The list processor task also calls DMTNRV to empty buffers that were previously filled by DMTNTR.

When it starts to process a buffer (or when it detects an RCB change with the mixed RCB support), DMTNRV determines the RIB associated with the data in the buffer. DMTNRV then passes the address of the RIB and the data to an appropriate routine that processes the specific type of data.

Managing Input Buffers

DMTNRV removes filled input buffers from the input buffer queue and processes all the data in them. As each buffer is emptied, DMTNRV places it on the free receive buffer queue.

Processing Message Streams

When processing message streams, DMTNRV calls DMTNUS to obtain each NMR from the buffer. DMTNRV calls DMTNHDMR to format the NMR into a Type L3 element, mapped by CMNDAREA. It then calls DMTRGXEP to route the element to its destination.

Processing Control Records

DMTNRV calls DMTNUSDC to obtain each RCB from the buffer. It then identifies the type of control record it receives and processes it accordingly:

RCB Type	Meaning	Action
X'90'	RQT	A free RIB is selected or generated and associated with the stream. An answer response is built in a small buffer.
X'A0'	PERMOK	The correct TIB is found and its wait bits are cleared.
X'B0'	NEGP	The correct TIB is found, its rejection bits are set, and any wait bits are cleared.
X'C0'	FUNCOMP	The correct TIB is found, its wait bits are cleared, and the completion bits are set.
X'D0'	RECVONL	The correct TIB is found and its wait bits are cleared.
X'F0'	GENCTL	If the record is an I or J record, DMTNRV issues return code 8 to the link driver task; the link is then deactivated.

Processing File Streams

DMTNRV calls DMTNUSDC to obtain each record segment from the buffer. DMTNRV passes the record segments to DMTNHD, which reassembles them into the appropriate type of NJE headers. If the record is not segmented, DMTNRV writes to each active device in the device chain.

SNA LU_TO NJE Session Driver

DMTSNE is the primary module for the SNA LU_TO SNANJE session driver. Its main responsibility is to set up the SNA session that RSCS uses to send data on the link.

Initialization

During initialization, DMTSNE calls DMTNCR to set up the NDWA and the initial RIB and TIB areas needed for the transmission streams. DMTSNE also validates the BIND image that is received in the LOGON or SCIP exit for the SNA session. If the session driver is the primary LU and the BIND image contains errors, it can attempt to correct the problems. See [z/OS: Network Job Entry \(NJE\) Formats and Protocols \(https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf\)](https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf) for more information. If the session driver is the secondary LU, it cannot correct the fields in the BIND image; the SNA session then terminates.

After it validates the BIND image, DMTSNE issues VTAM OPNDST or OPNSEC requests to establish the SNA session. The session drivers then exchange file mode (FM) SNA headers (the primary LU sends FM headers first) and checks for positive responses.

At this point, the NJE primary is determined by the EBCDIC order of the node names. The node name that is higher becomes the NJE primary. The primary side sends an I record, which is generated by DMTNCR. The secondary side must receive this sign-on record. If it accepts the I record, the secondary sends a J record.

Processing

DMTSNE uses two RPLs to communicate with VTAM. It uses the first to issue SEND macros when filled buffers can be sent on the link. DMTSNE uses the second RPL to issue RECEIVE macros when it must receive buffers from a remote node.

When it uses the SEND RPL, DMTSNE also uses two wait lists. The first wait list checks for files and messages to send to the remote node. It may also check for link driver commands and the completion of the RECEIVE RPL. If data is available to send, DMTSNE calls DMTNTR to fill the buffers.

DMTSNE uses the second wait list when all the send buffers are filled. It waits for VTAM to accept the buffer by posting the completion ECB for the SEND RPL. DMTSNE then calls DMTNTR to fill the buffer with more data.

Terminating the Link

DMTSNE terminates when the DRAIN or STOP command is issued for the session driver. It can also terminate when the remote node requests to end the session. When acting as the primary LU, DMTSNE calls the VTAM CLSDST macro to terminate the session. When it is the secondary LU, DMTSNE sends an RSHUTD Request Unit (RU) request to the remote node.

If the session driver abends when it is the primary LU (for example, after a decompression error), DMTSNE calls the VTAM CLSDST macro. If it abends while it is the secondary LU, DMTSNE issues the VTAM TERMSESS macro. If VTAM ends the session, DMTSNE does not call the macros and returns control to GCS.

BSC and CTC Link Driver

DMTNET is the primary module for the BSC and CTC (connections using either CTCA, 3088, ESCON, or FICON hardware) link drivers. It also supports the hardware associated with these links. Unlike DMTSNE, DMTNET performs error checking and recovery for the links.

Initialization

Initialization of DMTNET is similar to DMTSNE, except that it does not check for BIND requests. However, it must determine the type of adapter for the link and the channel programs set up for the adapter. DMTNET also calls DMTNCR to prepare the NDWA and the initial RIB and TIB areas needed for the transmission streams.

Initial contact differs for BSC and CTC links; see *z/OS: Network Job Entry (NJE) Formats and Protocols* (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf) for more information. The link driver initializes the adapter and attempts to send data to the remote node. If the corresponding link on the remote node is already active, the send completes and a response is received. The link driver then becomes the BSC and NJE primary and sends the I sign-on record.

If the corresponding remote link is inactive, the link driver puts up a PREPARE (for BSC) or waits for an ATTN interrupt (for CTCA). When the remote link sends data, the link driver sends the response and becomes the secondary system. Primary and secondary mode can also be forced with an initialization parameter.

After contacting the remote node and determining the primary node, the primary node sends an I sign-on record. DMTNET calls DMTNCRSG to evaluate the I record and convert it to the response J record. The initial FCS and BCB values are set on the sign-on records.

DMTNET uses one main wait list and several others, which are used for error recovery situations and for PREPARE mode. When its file or message ECB is posted, DMTNET calls DMTNTR to fill as many output

buffers as possible. It may have up to 6 output buffers. When I/O completes successfully (no I/O errors and the BCB is correct), DMTNET calls DMTNRV to empty the buffer. It then schedules a new I/O request, with the first available buffer in the output queue.

Error Processing

DMTNET processes the following type of recoverable and irrecoverable errors:

- BSC I/O adapter errors
- CTC I/O adapter errors
- BCB errors.

DMTNET processes recoverable errors by executing appropriate channel programs to synchronize the adapter with the remote system. For BSC links, DMTNET sends a NAK character to indicate that the last buffer was lost. For CTC links, the only recoverable error condition is getting buffered status. In this case, the link is able to read the last buffer.

When processing unrecoverable errors, DMTNET sends the remote system notification about the error (possible only with BCB errors) and disables the adapter. If it cannot notify the remote system about the error, DMTNET terminates the link driver.

Preparing Protocols

If no files or messages are enqueued on the link and no incoming streams are active, DMTNET attempts to enter prepare mode, if the feature was agreed on in the sign-on records. A special null buffer is set up to request entry to prepare. If the other side agrees to prepare, it responds with a similar buffer. At this point, the two sides have entered prepare mode. After 10 minutes, DMTNET checks the remote system to determine if the connection is still valid.

Terminating the Link

When a BSC or CTC link terminates, DMTNET issues a return code. For unrecoverable errors or invalid sign-on records, the return code indicates if the link should be restarted or auto-started.

TCPNJE Link Driver

DMTTNE is the primary module for the TCPNJE link driver. This link driver is a full NJE link driver and has the same RSCS link features as other NJE links. However, for TCPNJE-type links, TCP/IP is used as the data transport mechanism. After a connection has been established, the two nodes will use NJE CTCA packets to communicate with each other.

Initialization

During initialization, DMTTNE calls DMTNCR to setup up the NDWA and the initial RIB and TIB areas needed for transmission streams. The remote and local nodes establish communications with each other by issuing TCP/IP socket CONNECT calls.

Identifying Unique Connections

In a network, RSCS may need to communicate with more than one NJE node. In this case, RSCS must differentiate where each link driver is connected. For physical links, the subchannel address identifies each unique connection. For TCP/IP sockets, however, the local and remote IP address and port numbers uniquely identify the connection. Port numbers are global within the same IP address.

Each TCPNJE link driver must listen on a specific port (which is defined to the other NJE node by the RMTPORT parameter) and the IP address of the remote node must be defined to the link. The TCPNJE link driver must also define a local port number (using the LCLPORT parameter); this port is used to listen for CONNECT requests from the remote NJE node. If the local and remote NJE nodes are on different systems, the default values for the RMTPORT and LCLPORT parameters can be used to define the links.

See *z/VM: RSCS Networking Planning and Configuration* for more information about other TCPNJE link parameters.

For each TCPNJE-type link, one IUCV connection is used between RSCS and the TCP/IP virtual machine.

Processing

To start the connect process, a LISTEN request must be performed for a specific port. The LISTEN request allows another node to attempt a connection. Either side of the connection (the local or remote node) can attempt to connect to a port on the other node. To do so, the node issues a TCP/IP CONNECT socket call. The other node then accepts the connect request with an ACCEPT socket call. The two nodes then exchange control records followed by an exchange of I and J sign-on records. For more information about establishing a TCP/IP NJE connection, see *z/OS: Network Job Entry (NJE) Formats and Protocols* (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf).

After this connection is established, the two nodes use NJE CTCA packets to communicate with each other. Data transmission is asynchronous. That is, a TCP/IP RECEIVE socket call is always outstanding on the link and a SEND request is issued whenever data can be transmitted. When a RECEIVE socket call is issued, the quantity of data that is received may vary. For example, the link driver may receive one byte of data or the entire length of the data. To account for the varying data length, DMTTNE calls deblocking routines correctly assemble the data it receives.

Terminating the Link

DMTTNE terminates when a DRAIN or STOP command is issued for the link. It can also terminate when the remote node requests to end the connection. When it terminates, it will issue a SOCKET CLOSE function.

GATEWAY Link Driver

The GATEWAY link driver, part of the RSCS gateway programming interface, allows RSCS to exchange NJE data with nodes that use other networking protocols. GATEWAY-type links can also be used to exchange NJE data over nonstandard NJE paths, such as to tape or over a TCP/IP network. The gateway programming interface is also made up of gateway service macros, supplied by RSCS, and gateway programs, which each installation creates for any special purpose. The remote node to which the GATEWAY-type link is connected must also supply code to communicate with the gateway program.

The GATEWAY link driver task, DMTGPI, does not provide a mainline routine for the GATEWAY-type link. Rather, it contains routines, which correspond to the gateway service macros, that read and write data and control the state of the interface. The gateway program accesses the DMTGPI routines by invoking the gateway service macros (see [Figure 41 on page 73](#)).

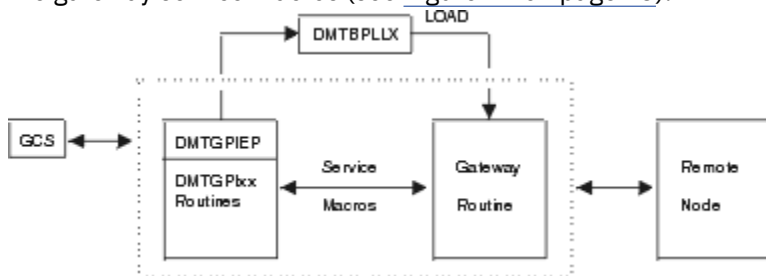


Figure 41. Structure of GATEWAY Link Driver

Initialization

When it initializes, DMTGPI obtains all needed storage and processes all parameters specified for the GATEWAY-type link. DMTGPI calls DMTNCRIN to prepare the NDWA, RIBs, and TIBs. Finally, the gateway program is loaded into storage and invoked. When the gateway program returns, the GATEWAY link driver task returns control to GCS.

Gateway Service Macros

Gateway programs access information from RSCS by invoking the gateway service macros. The macro invocations correspond to various routines in DMTGPI.

NJEOPEN

A gateway program invokes the NJEOPEN macro when it wants to send or receive a file or set up an input or output message stream. The type of NJEFILE determines the action that is taken and any errors are reflected by a return code:

- For the input or output message stream, the message TIB or RIB is set up and associated with the NJEFILE control block.
- For an input file, a free TIB is located and DMTAXMRQ is called (OPENIN request) to obtain an input file. If one is available, it is associated with the TIB and NJEFILE control blocks, which are also associated with each other.
- For an output file, a free RIB is located or built and associated with the NJEFILE control block.

NJECLOSE

The gateway program invokes NJECLOSE to complete sending or receiving a file or to release the input or output message stream.

- For the input or output message stream, the message TIB or RIB is disassociated with the NJEFILE control block and marked as inactive.
- For an input file, DMTAXMRQ is called (CLOSEIN request) to complete the reading of the file. If the entire file has not been read, it is placed in the queue; otherwise, it is purged.
- For an output file, if the entire file has not been received, DMTAXMRQ is called (CLOSEOUT request) for each open output spool file to purge the partial files. If the entire file was received, DMTAXMRQ is called (CLOSEOUT request) to close and accept each output spool file.

NJEGET

Gateway programs invoke NJEGET to obtain another record for an open input stream.

- For the message stream, DMTCOMDQ is called to obtain a routing element. If none is available, a return code is passed back. Otherwise, DMTNHDMT is called to format the element into an NMR, which is then passed back.
- For a file stream, DMTNHE is called to build any necessary NJE headers, which are then passed back to the gateway program with the appropriate SRCB and length. All data records are obtained by calls to DMTRDREP and passed back with the appropriate SRCB and length.

NJEPUT

Gateway programs call NJEPUT to write a record to an open input stream. The type of NJEFILE determines the action that is taken; any errors are reflected by a return code:

- For the message stream, DMTCOMDQ is called to obtain a routing element. If none is available, a return code is passed back. Otherwise, DMTNHDMT is called to format the element to an NMR and that is passed back.
- For a file stream, DMTNHE is called to build any necessary headers, which are passed back to the gateway program with the appropriate SRCB and length. All data records are obtained by calls to DMTRDREP and passed back with the appropriate SRCB and length.

NJERJECT

Gateway programs call NJERJECT to indicate that it wants to stop receiving a file. The reason may be indicated by an NJE reason code. DMTAXMRQ is called (CLOSEIN request) to close and hold the input

spool file or perform any other action that may be indicated. The NJEFILE and TIB control blocks are disassociated.

NJEABORT

Gateway programs invoke NJEABORT to stop sending a file for any reason. The request is processed by calling DMTAXMRQ (CLOSEOUT) to close and purge any open output spool files; the NJEFILE and RIB control blocks are disassociated.

NJECONCT

Gateway programs invoke NJECONCT to notify RSCS that it is ready to process transactions. The link is then marked as “connect”.

NJEDSCON

The gateway program calls NJEDSCON to notify RSCS that it is not ready to process any transactions. The link is then marked “active”.

Terminating the Link

The gateway program indicates when the GATEWAY link driver task should terminate. DMTMANEX, the end of task routine, processes any open spool files. DMTGPI does not perform any session clean up. When the link terminates, it returns control to GCS.

Chapter 7. Printer Link Drivers

This chapter describes the RSCS printer link driver tasks. RSCS can communicate with 3270 information display printers over SNA and non-SNA links.

3270P Printer Link Driver

DMTRPT is the primary module for the 3270 printer link driver task. DMTPCR formats individual records into 3270 data stream format on behalf of DMTRPT. 3270P-type links send output spool files to 3270 printers attached by a 3271, 3272, 3274 or 3276 control unit. 3270P-type links also support Graphic Data Display Manager (GDDM®) files that are sent to 3270-type printers.

DMTRPT performs the following functions between RSCS and the printer:

- Sends spool files to the remote printer
- Sends messages and responses to the remote printer
- Reads asynchronous Intelligent Printer Data Stream (IPDS) interrupts from the printer.

Initialization

The REX task attaches DMTRPT when a START command is issued for a 3270P-type link. DMTRPTEP is the main entry point to the link driver task. Other routines in DMTRPT and DMTPCR initialize the link, process START parameters, process the printer data, and stop the link.

After it is attached at DMTRPTEP, the PRTINIT routine initializes the link driver task. PRTINIT obtains storage and initializes a chain of save areas and work areas (DWAs). It then identifies the ESTAE exit for the task and initializes the ECB list. PRTINIT also defines the read modified CCWs and initializes the write CCW.

Receiving and Sending Data

After PRTINIT completes its processing, it calls the PRTGO routine. PRTGO, the main processing routine, monitors ECBs (LMSGECB, LCMDECB, LFILECB) to determine when work arrives for the link. If applicable to the printer, PRTGO also checks for asynchronous interrupts that can occur while a file is being processed.

When a message is enqueued on the link, PRTINIT calls the MSGPROC routine. MSGPROC calls DMTCOMDQ to remove each message element from the link's message queue. Any blanks in the message text are removed by the COMPACT routine. MSGPROC then places each message in a buffer. When the buffer is full or all messages have been processed, MSGPROC calls the LINEIO routine to send it to the printer. Prefix information is added to each buffer before any data is inserted.

If the printer supports intelligent printer data stream (IPDS) data streams, DMTRPT indicates the beginning of the file in the structured field. This ensures that the messages are not mistaken to be part of any files that might be sent to the printer. When all messages are processed, the final buffer contains an end of file marker. If the printer does not support IPDS, LINEIO only sends messages between the files that are being processed. After all messages are processed, PRTGO regains control.

When a file arrives on the 3270P-type link, the AXSGET routine calls DMTAXMRQ to open the file. AXSGET then returns control to PRTGO, which issues requests to receive the data from spool. The GETBLOCK routine calls DMTRDREP to obtain each file record. The records are packaged into a buffer with the correct prefix already set in the buffer. All buffers are processed in the 3270 data stream format.

When the file is sent to the printer, the AXSPURGE routine purges the file from the local node. PRTGO then receives control and continues to monitor the ECB list.

When a command is issued for the 3270P-type link, as indicated by an ECB, PRTGO calls the CMDPROC routine. CMDPROC processes the following link commands: BACKSPACE, DRAIN, FREE, FWDSPACE, HOLD, READY, START, and TRACE.

Building Data Streams

When a file arrives on the link, the AXSGET routine calls DMTAXMRQ to open the file. AXSGET then returns control to PROGO, which in turn calls DMTRDREP to get the next file record.

The GETBLOCK routine continues to call DMTRDREP until it obtains all records in the file. GETBLOCK prepares the buffer that is sent to the printer and places the appropriate prefix in the buffer. GETBLOCK calls DMTSEPHD and DMTSEPTR to build any required separator pages. Each record is placed in a transmission buffer. When the buffer is filled, the records are compressed. GETBLOCK then calls the LINEIO routine to send the buffer to the printer. DMTPCRTR formats the records into 3270 data stream format.

I/O Processing

LINEIO is the device I/O processing routine for the 3270P-type link. LINEIO prepares the IOTABLE with the channel program and device information and calls XECUTE to execute the I/O request. XECUTE then calls DMTIOTST to execute the channel program on the device. DMTIOTGE returns any resulting I/O interrupts to RSCS. Any interrupts or errors are then evaluated and a return code is passed to the link driver. LINEIO might also attempt to retry an I/O request if some errors occur.

Terminating the Link

3270P-type links end when a DRAIN, STOP, or SHUTDOWN command is issued. The PRRTERM routine processes all termination requests for the link. It also performs any special processing required for IPDS mode. PRTGO calls PRRTERM when the LTERECB is posted. PRRTERM calls DMTLOG to close the I/O transaction log; it then returns control and a return code to GCS.

TN3270E Printer Link Driver

DMTTPT is the primary module for the TN3270E printer link driver task. DMTPCR formats individual records into 3270 data stream format on behalf of DMTTPT. TN3270E-type links send output spool files to printers attached in a TCP/IP network with the capability to initiate a TN3270E session, such as emulators, capable of supporting 3270 print streams. TN3270E-type links also support Graphic Data Display Manager (GDDM) files that are sent to 3270-type printers.

DMTTPT is very similar in processing to DMTRPT, except the printer is attached within a TCP/IP network rather than channel attached to the host. The main differences in processing are:

- The session must be initiated from the printer side rather than by RSCS. A TN3270E session must first be established to the VM TCP/IP stack. Once the session is successfully established, VM TCP/IP will create a printer logical device and attach it to RSCS. Once attached to RSCS, the TN3270E-type printer link can be started. There are configuration steps that need to be performed in the VM TCP/IP stack; see *z/VM: RSCS Networking Planning and Configuration* and *z/VM: TCP/IP Planning and Customization* for details on configuration steps.
- During initialization, DMTTPT will ensure the printer is attached via TN3270E; otherwise the link will terminate.

DMTTPT performs the following functions between VM and the printer:

- Sends spool files to the remote printer
- Sends messages and responses to the remote printer
- Reads asynchronous Intelligent Printer Data Stream (IPDS) interrupts from the printer.

Initialization

The REX task attaches DMTTPT when a START command is issued for a TN3270E-type link. DMTTPTEP is the main entry point to the link driver task. Other routines in DMTTPT and DMTPCR initialize the link, process START parameters, process the printer data, and stop the link.

After it is attached at DMTTPTEP, the PRTINIT routine initializes the link driver task. PRTINIT obtains storage and initializes a chain of save areas and work areas (DWAs). It then identifies the ESTAE exit for the task and initializes the ECB list. PRTINIT also defines the read modified CCWs and initializes the write CCW. PRTINIT reads the device characteristics and ensures the printer is attached through the VM TCP/IP stack as a TN3270E device.

Receiving and Sending Data

After PRTINIT completes its processing, it calls the PRTGO routine. PRTGO, the main processing routine, monitors ECBs (LMSGECB, LCMDECB, LFILECB) to determine when work arrives for the link. If applicable to the printer, PRTGO also checks for asynchronous interrupts that can occur while a file is being processed.

When a message is enqueued on the link, PRTINIT calls the MSGPROC routine. MSGPROC calls DMTCOMDQ to remove each message element from the link's message queue. Any blanks in the message text are removed by the COMPACT routine. MSGPROC then places each message in a buffer. When the buffer is full or all messages have been processed, MSGPROC calls the LINEIO routine to send it to the printer. Prefix information is added to each buffer before any data is inserted.

If the printer supports intelligent printer data stream (IPDS) data streams, DMTTPT indicates the beginning of the file in the structured field. This ensures that the messages are not mistaken to be part of any files that might be sent to the printer. When all messages are processed, the final buffer contains an end of file marker. If the printer does not support IPDS, LINEIO only sends messages between the files that are being processed. After all messages are processed, PRTGO regains control.

When a file arrives on the TN3270E-type link, the AXSGET routine calls DMTAXMRQ to open the file. AXSGET then returns control to PRTGO, which issues requests to receive the data from spool. The GETBLOCK routine calls DMTRDREP to obtain each file record. The records are packaged into a buffer with the correct prefix already set in the buffer. All buffers are processed in the 3270 data stream format.

When the file is sent to the printer, the AXSPURGE routine purges the file from the local node. PRTGO then receives control and continues to monitor the ECB list.

When a command is issued for the TN3270E-type link, as indicated by an ECB, PRTGO calls the CMDPROC routine. CMDPROC processes the following link commands: BACKSPACE, DRAIN, FREE, FWDSPACE, HOLD, READY, START, and TRACE.

Building Data Streams

When a file arrives on the link, the AXSGET routine calls DMTAXMRQ to open the file. AXSGET then returns control to PROGO, which in turn calls DMTRDREP to get the next file record.

The GETBLOCK routine continues to call DMTRDREP until it obtains all records in the file. GETBLOCK prepares the buffer that is sent to the printer and places the appropriate prefix in the buffer. GETBLOCK calls DMTSEPHD and DMTSEPTR to build any required separator pages. Each record is placed in a transmission buffer. When the buffer is filled, the records are compressed. GETBLOCK then calls the LINEIO routine to send the buffer to the printer. DMTPCRTR formats the records into 3270 data stream format.

I/O Processing

LINEIO is the device I/O processing routine for the TN3270E-type link. LINEIO prepares the IOTABLE with the channel program and device information and calls XECUTE to execute the I/O request. XECUTE then calls DMTIOTST to execute the channel program on the device. DMTIOTGE returns any resulting I/O interrupts to RSCS. Any interrupts or errors are then evaluated and a return code is passed to the link driver. LINEIO might also attempt to retry an I/O request if some errors occur.

Terminating the Link

TN3270E-type links end when a DRAIN, STOP, or SHUTDOWN command is issued. The PRTERM routine processes all termination requests for the link. It also performs any special processing required for IPDS mode. PRTGO calls PRTERM when the LTERECB is posted. PRTERM calls DMTLOG to close the I/O transaction log; it then returns control and a return code to GCS.

SNA 3270 Printer Session Driver

DMTSPT and DMTPCR comprise the SNA3270P session driver task. SNA3270P-type links enable RSCS to send output spool files to 3270 printers attached to VTAM by a 3271, 3272, 3274, or 3276 control unit. DMTSPT provides this support for LU_T0, LU_T1 and LU_T3 type sessions.

The session driver, however, does not communicate directly with the 3270 device. The device is described to VTAM by a logmode table. It is presented to RSCS in the BIND image during session initiation; see [“Establishing a Session” on page 55](#) for more information about SNA flows. The session driver uses VTAM macros to communicate with the logical unit. These macros send data to the printer, determine the success of that transmission, and the state of the session.

To notify the SCT task that an event has affected the state of the SNA session, VTAM schedules various exit routines defined for RSCS in DMTVXT. These exit routines are defined to VTAM during initialization of the RSCS/VTAM interface. VTAM can schedule the following exit routines for the SNA3270P session driver:

LOGON

Scheduled because a SIMLOGON request was issued by the SNA control task, or because a command was issued by the VTAM operator to begin a session.

NSEXIT

Scheduled to perform cleanup when the secondary logical unit abends or abruptly stops.

RELREQ

Scheduled when another VTAM application (RSCS, IMS™, or CICS®) issues a SIMLOGON request for logical unit currently in session with this session driver.

LOSTERM

Driven when some error occurs on a session outside the control of each logical unit (for example, a hardware problem in the VTAM path).

These exit routines communicate with the SNA3270P session driver by posting an ECB. The session driver then performs the appropriate processing for the ECB. For more information on these and other VTAM exit routines for the RSCS application, see [“VTAM Exit Routines” on page 57](#).

Initialization

The SCT task attaches DMTSPT when a START command is issued for an SNA3270P-type link, or if a VTAM operator initiates a SIMLOGON. In each case, the VTAM LOGON exit is scheduled for the session driver. The LOGON exit, DMTVXTLG, then attaches the session driver task, which receives control at entry point DMTSPTEP.

When the session driver task receives control, it issues an OPNDST to begin the bind process. The BIND image that the session driver passes to the secondary logical unit is not negotiable. The secondary LU must indicate if it accepts this BIND image by replying with a positive or negative response. If the BIND image is rejected, the session driver does not initialize. If it is accepted, the session driver task continues to initialize and the PRTINIT routine receives control.

PRTINIT obtains and initializes storage for various work areas (DWAs). It also processes the START command parameters, establishes the ESTAE exit, and initializes the ECB list. Start command parameters are processed by calling DMTPAREP and DMTPCRIN.

PRTINIT then verifies the BIND image for the SNA session; it obtains this information from the CINIT RU request. This request is generated when the SCT task issues the SIMLOGON macro to request the SNA

session. The BIND image indicates the type of SNA session being requested and the requested buffer size (RU). PRTINIT then issues a OPNDST request to start the SNA session between DMTSPT and the printer.

Receiving and Sending Data

When the OPNDST request completes, the PRTGO routine receives control. PRTGO, the main processing routine, monitors ECBs to determine when work arrives for the link. PRTGO contains routines that process spool files for the link. This routine also monitors VTAM requests for RSCS to release its session with the printer for another VTAM application.

Receiving Data

The LRECECB indicates when the link has received data. The SCT task posts this ECB when the RECEIVE request, which it issues when establishing the session, completes. When this ECB is posted, PRTGO calls the RECPROC routine.

RECPROC issues a RECEIVE SPECIFIC macro to process the data from the printer. RECPROC accepts the following types of data:

- Valid input commands, such as: RTR, LUSTAT, CANCEL or CHASE. RECPROC accepts this data from LU_T0, LU_T1, and LU_T3 sessions.
- Responses to an IPDS selection received over an LU_T0, LU_T1, or LU_T3 session.
- PA1 or PA2 data received over an LU_T1 session.

If RECPROC receives any other types of data, it issues a negative response. After it receives all valid data, RECPROC issues the RESETSR macro and returns control to PRTGO.

When messages are enqueued on the session driver, PRTGO calls the MSGPROC routine. MSGPROC calls DMTCOMDQ to obtain a message element and places the element in a buffer. When a buffer is filled or there are no more message elements on the queue, MSGPROC calls the SENDRU routine to send the buffer to the printer. PRTGO then regains control.

When a file is enqueued on the link, the AXSGET routine calls DMTAXMRQ to open the file. PRTGO then issues requests for the printer to receive this file. PRTGO also calls the GETBLOCK routine, which calls DMTRDREP to obtain each record in the file. GETBLOCK places each record into a buffer (RU) with the correct flags and indicators set in the RPL. To send the buffer to the printer, GETBLOCK calls the SENDRU routine, which issues SEND macro with that RPL and its associated RU. When the file is sent to the printer, the AXSPURGE routine purges it from the local node and returns control to PRTGO.

The CMDPROC routine processes each command that is enqueued on the SNA3270P-type link. The valid commands for this link include: BACKSPACE, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, READY, START, and TRACE. Each time these commands are issued, the link's command ECB is posted. When CMDPROC has completed its processing, it returns control to PRTGO.

Building Data Streams

When a file arrives in the link, the AXSGET routine calls DMTAXMRQ to open the spool file. Control returns to PRTGO, which then calls DMTRDREP to obtain a record in the file. This record becomes the TAG record of the file. The GETBLOCK routine continues to call DMTRDREP until all records in the file are obtained.

GETBLOCK places any appropriate SNA headers in the buffer and prepares to send the buffer to the remote node. GETBLOCK calls DMTSEPHD and DMTSEPTR to build any required separator pages. Each record is placed in a transmission buffer; when the buffer is filled, the records are compressed. GETBLOCK then calls SENDRU to send the file to VTAM. DMTPCRTTR formats the records into 3270 data stream format.

I/O Processing

VTAM performs all I/O processing for the SNA3270P-type link. The DMTSPT routines issue the RECEIVE and SEND macros to interact with VTAM. Each routine also processes any return codes issued by these macros. The SCT task and DMTVXT process any exit routines driven by VTAM.

Terminating the Link

SNA3270P-type links end when a DRAIN, FLUSH, STOP, or SHUTDOWN command is issued. The PRTERM routine processes all termination requests for SNA3270P-type links. It receives control from PRTGO when the LTERECB or TERMECB is posted or when RSCS must release the SNA session. PRTERM issues the VTAM CLSDST macro to end the session or to end and release the session.

If RSCS must release the SNA session, the SNA3270P-type link remains active and waits to re-establish communication with the session. If the session is to reactivate, DMTSPT issues the SIMLOGON request and attempts a new OPNDST request; the SCT task does not issue this request. If the requests are successful, PRTGO regains control and processing on the session driver continues.

When the session driver ends, PRTERM calls DMTLOG to close the I/O transaction log; it then returns control and a return code to GCS.

ASCII Printer and Plotter Link Driver

ASCII printer and plotter links allow RSCS to send data streams of ASCII characters and control sequences to ASCII printers and plotters. These ASCII devices must be connected to the z/VM system by an IBM 7171 ASCII Device Attachment Control Unit, 9370 ASCII Subsystem Controller, or equivalent ASCII controller.

This link driver task is made up of routines and exits in DMTAPT and customer-supplied ASCII exit routines. The APT task uses the following exits to provide ASCII functional support for printers. These exit points are called at various points in the processing cycle of the driver.

INIT

Performs any required initialization

TAGEX

Processes TAG information

RECEIVE

Receives each spool file record

RESET

Resets the device after any logical end of a spool file

MSGEX

Translates messages into ASCII format

ATTNEX

Processes attention interrupts generated by the 7171 ASCII Device Attachment Control Unit or 9370 ASCII Subsystem Controller

TERM

Called just before the link driver terminates.

As [Figure 42](#) on page 82 shows, each exit in DMTAPT corresponds to an ASCII printer or plotter exit routine. These routines can customize the way the ASCII-type link driver communicates with a specific ASCII device. These exit routines can modify or translate the input spool data. They can also add ASCII control sequences based on the external characteristics of the spool file.

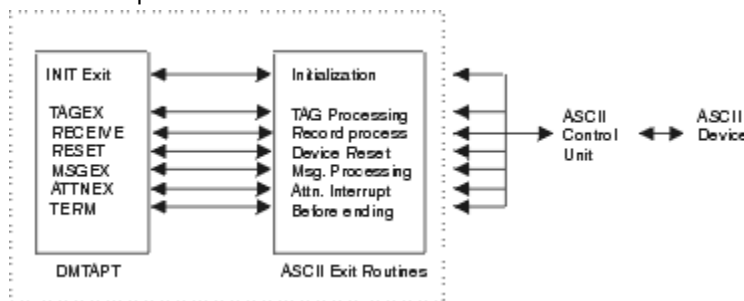


Figure 42. Structure of an ASCII-Type Link Driver

All calls to exit routines follow standard OS conventions. ASCII exit routines must also follow these conventions when returning control to DMTAPT. See *z/VM: RSCS Networking Exit Customization* for more information about ASCII exit routines.

Initialization

The APT task is attached when a START command is issued for an ASCII-type link. The task can be attached by the REX task or by the AST task, if the link is also identified as an auto-start link. DMTAPT contains one entry point, DMTAPTEP. Other routines in the module perform various functions for the link driver.

When the ASCII link driver task is attached, the PRTINIT routine initializes the DWA, ECB lists, CCWs, and target addresses. It determines the line address for the ASCII-type link from information in the link's LINKTABL entry.

PRTINIT then calls the APTBLD routine to load any ASCII exit routines that were specified on the START command of the ASCII-type link. The APTBLD routine calls DMTBPLLX to dynamically load each ASCII exit routine.

When all ASCII exit routines have been loaded, the PRTINIT routine again receives control. It, in turn, passes control to the INIT exit.

INIT

The initialization exit routine is called when RSCS enables the ASCII device. The link is logically switched to the hardcopy device, reset, and set to “stand by” mode. On return from the exit routine, storage is obtained for the output TP buffer. Message DMT141I is issued and the link driver waits for the link to connect. When the link is connected, the DMT162I message is issued and the ASCII-type link can begin to process files.

Receiving and Sending Data

The PRTGO routine in DMTAPT is the main control routine for the ASCII-type link driver. The routine first checks various flags to determine if it must end, hold a file on the link, or determine if a file is currently held. PRTGO then checks if any messages are enqueued on the link.

If no messages are enqueued, PRTGO calls the AXSGET routine to get a file. If no files are available, PRTGO checks the link driver task's ECB list to determine if it has received other requests (commands, termination). If no ECBs are posted, PRTGO waits to receive some work.

If the AXSGET routine obtains a file, control passes to the PROGO routine. This routine determines if any reader commands are pending. If so, it will process the commands before the file is processed.

After any reader commands are processed, the GETBLOCK routine is called to begin file processing. The GETBLOCK routine performs all file processing. This routine calls the TAGEX exit to process the TAG record for the current file.

TAGEX

The TAG processing exit is called when a new spool file is opened. The TAG processing routine receives the file's TAG element. It uses TAG fields, supplied by the file originator, to identify the file or control the output. When the TAG processing routine completes its processing, the RECEIVE exit routine receives control.

RECEIVE

The record processing exit is called for each logical output record received from the input spool file. The exit routine can translate EBCDIC code into ASCII code or perform any other processing that might be necessary for the specific ASCII device. When the exit routine processes enough records to fill a buffer, GETBLOCK returns control to PRTGO. The PRTGO routine then calls LINEIO to send this filled buffer to the printer.

If GETBLOCK finds an error or the end of the file is reached, it calls the RESETEOF routine. This routine, in turn, calls the RESET exit.

RESET

The device reset routine is called when the end of file is reached or when a file is flushed, back-spaced, or forward-spaced. This exit routine places characters, which reset the device to print the next file, into the print line portion of the print line vector. The exit routine then passes control to PRTGO. When PRTGO receives control, it calls the LINEIO routine to process the buffer or complete the file processing. If a message element is enqueued on the ASCII-type link, PRTGO passes control to the MSGPROC routine.

MSGPROC calls DMTCOMDQ to remove each enqueued message. If a message processing exit routine has not been specified, the ASCII-type link does not process the messages. If a message processing routine is supplied, MSGPROC passes the message element to that routine.

MSGEX

The processing exit is called when a message destined to the ASCII device is enqueued on the ASCII-type link. This exit routine translates the EBCDIC message into ASCII code; it can also suppress the message. The exit routine is called once for each message enqueued on the link. When all messages are processed, the message processing routine returns control to PRTGO.

Command Processing

When a command ECB for the APT task is posted, the CMDPROC routine processes the command. These commands include: BACKSPACE, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, READY, START, and TRACE. When the command is processed, CMDPROC returns control to PRTGO.

Processing CP File Characteristics

RSCS supplies sample ASCII exit routines for use with different types of ASCII devices. These exit routines respond to options specified on the CP TAG, SPOOL, and CLOSE commands. For more information about the responses from these sample ASCII exit routines, see [z/VM: RSCS Networking Exit Customization](#).

Building Data Streams

AXSGET calls DMTAXMRQ to request to open a spool file. If there are no files enqueued for the ASCII-type link, AXSGET receives a nonzero return code. AXSGET returns control to the PRTGO routine.

If a file is to be processed on the link, AXSGET initializes the necessary data and work areas. AXGET then calls DMTRDROP to get the TAG record and other information about the file. AXSGET then passes control to PROGO, which starts to process the file. PROGO calls DMTRDREP to get the first record of the file. Each remaining file record is obtained by calls to the GETBLOCK routine.

GETBLOCK prepares the print buffer to be sent to the ASCII printer. It places the appropriate buffer prefix into the buffer. If a separator page is required, DMTSEPHD is called to create the page.

GETBLOCK then calls DMTRDREP to obtain each record of the file. The file records are then placed into the buffer. When the buffer is filled, the records are compressed. The PROGO routine then receives control again and calls LINEIO to write the buffer.

When the end of the file is reached, GETBLOCK checks if a trailer page is needed. It then calls DMTSEPTR, as necessary. When this processing completes, PROGO regains control.

I/O Processing

The LINEIO routine initially processes all line I/O requests. LINEIO prepares all control block information and calls XECUTE, which then calls DMTIOTST. DMTIOTST then passes the I/O request to GCS. LINEIO also performs I/O error verification and recovery needed for the requested write operation.

If the ASCII control unit generates an attention interrupt during this processing, LINEIO calls the AIDDECOD routine. AIDDECOD, in turn, passes control to the ATTNEX exit routine.

ATTNEX

The ATTN exit routine is called when the ASCII control unit generates an attention interrupt. An attention interrupt is usually generated after RSCS sends each data buffer.

To obtain the Attention Identifier (AID) byte, RSCS performs a “3270 Read Modified” operation each time an attention interrupt is generated (required by the 7171). Usually, a null AID (X'8') is returned. However, if the ASCII device has a keyboard, the value of the AID might be changed by pressing an attention generating key (ENTER, PF key, or PA). The ASCII exit routine can examine the AID byte and optionally pass an RSCS command to the link driver for execution.

Terminating the Link

The PRRTERM routine processes all termination requests for the ASCII-type link. It receives control when the link's LTERECB is posted in the LINKTABL. PRRTERM then close the log and returns control and a return code to GCS.

TCPASCII Printer and Plotter Link Driver

The TCPASCII link driver is similar to the ASCII print driver. However, TCPASCII-type links are used to establish connections to ASCII printer or plotters that are attached to a terminal server in a TCP/IP network. The transport media between RSCS and the printer is via TCP/IP socket connections.

This link driver task is made up of routines and exits in DMTTAP and customer-supplied ASCII exit routines. The TAP task uses the following exits to provide ASCII functional support for printers. These exit points are called at various points in the processing cycle of the driver.

INIT

Performs any required initialization

TAGEX

Processes TAG information

RECEIVE

Receives each spool file record

RESET

Resets the device after any logical end of a spool file

MSGEX

Translates messages into ASCII format

ATTNEX

Processes data received from the socket

TERM

Called just before the link driver terminates

As [Figure 43 on page 86](#) shows, each exit in DMTTAP corresponds to an ASCII printer or plotter exit routine. These routines can customize the way the TCPASCII-type link driver communicates with a specific ASCII device. These exit routines can modify or translate the input spool data. They can also add ASCII control sequences based on the external characteristics of the spool file.

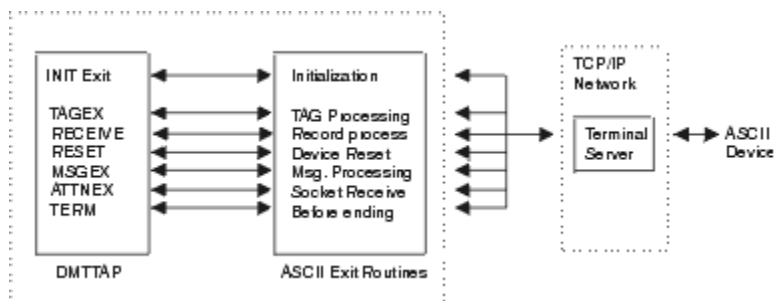


Figure 43. Structure of a TCPASCII-Type Link Driver

All calls to exit routines follow standard OS conventions. Any ASCII exit routines associated with the TCPASCII-type link must also follow these conventions when returning control to DMTTAP. See [z/VM: RSCS Networking Exit Customization](#) for more information about ASCII exit routines.

Initialization

The TAP task is attached when a START command is issued for a TCPASCII-type link. The task can be attached by the REX task or by the AST task if the link is also identified as an auto-start link. DMTTAP contains one entry point, DMTTAPEP. Other routines in the module perform various functions for the link driver.

When the TCPASCII link driver task is attached, the PRTINIT routine initializes the DWA, ECB lists, and target addresses. PRTINIT then calls the TAPBLD routine to load any ASCII exit routines that were specified on the START command for the TCPASCII-type link. The TAPBLD routine calls DMTBPLLX to dynamically load each ASCII exit routine.

When all ASCII exit routines have been loaded, the PRTINIT routine again receives control. It, in turn, passes control to the INIT exit.

INIT

The initialization exit routine is called when RSCS connects to the ASCII device. On return from the exit routine, storage is obtained for the output TP buffer. Message DMT181I is issued and the link driver waits for the SOCKET CONNECT to complete. When the connection is completed, message DMT182I is issued and a SOCKET RECEIVE is set up; the RECVECB will be posted when the RECEIVE request completes. If the SOCKET CONNECT fails due to a network problem or because the remote host rejected the connection, DMTTAP will retry the connect.

Receiving and Sending Sata

The PRTGO routine in DMTTAP is the main control routine for the TCPASCII-type link driver. The routine first checks various flags to determine if it must DRAIN the link, hold a file on the link, or determine if a file is currently held. PRTGO then checks if any messages are enqueued on the link.

If no messages are enqueued, PRTGO calls the AXSGET routine to get a file. If no files are available, PRTGO checks the link driver task's ECB list to determine if it has received other requests (commands, termination). If no ECBs are posted, PRTGO waits to receive some work.

If the AXSGET routine obtains a file, control passes to the PROGO routine. This routine determines if any reader commands are pending. If so, commands will be processed before the file is processed.

After any reader commands are processed, the GETBLOCK routine is called to begin file processing. The GETBLOCK routine performs all file processing. This routine calls the TAGEX exit to process the TAG record for the current file.

TAGEX

The TAG processing exit is called when a new spool file is opened. The TAG processing routine receives the file's TAG element. It uses TAG fields, supplied by the file originator, to identify the file or control

the output. When the TAG processing routine completes its processing, the RECEIVE exit routine receives control.

RECEIVE

The record processing exit is called for each logical output record received from the input spool file. The exit routine can translate EBCDIC code into ASCII code or perform any other processing that might be necessary for the specific ASCII device. When the exit routine processes enough records to fill a buffer, GETBLOCK returns control to PRTGO. The PRTGO routine then calls SOCKIO to send this filled buffer to the printer.

If GETBLOCK finds an error or the end of the file is reached, it calls the RESETEOF routine. This routine, in turn, calls the RESET exit.

RESET

The device reset routine is called when the end of file is reached or when a file is flushed, back-spaced, or forward-spaced. This exit routine places characters, which reset the device to print the next file, into the print line portion of the print line vector. The exit routine then passes control to PRTGO. When PRTGO receives control, it calls the SOCKIO routine to process the buffer or complete the file processing. If a message element is enqueued on the TCPASCII-type link, PRTGO passes control to the MSGPROC routine.

MSGPROC calls DMTCOMDQ to remove each enqueued message. If a message processing exit routine has not been specified, the TCPASCII-type link does not process the messages. If a message processing routine is supplied, MSGPROC passes the message element to that routine.

MSGEX

The processing exit is called when a message destined to the ASCII device is enqueued on the TCPASCII-type link. This exit routine translates the EBCDIC message into ASCII code; it can also suppress the message. The exit routine is called once for each message enqueued on the link. When all messages are processed, the message processing routine returns control to PRTGO.

Command Processing

When a command ECB for the TAP task is posted, the CMDPROC routine processes the command. These commands include: BACKSPACE, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, READY, START, and TRACE. When the command is processed, CMDPROC returns control to PRTGO.

Processing CP File Characteristics

RSCS supplies sample ASCII exit routines for use with different types of ASCII devices. These exit routines respond to options specified on the CP TAG, SPOOL, and CLOSE commands. For more information about the responses from these sample TCPASCII exit routines, see [z/VM: RSCS Networking Exit Customization](#).

Building Data Streams

AXSGET calls DMTAXMRQ to request to open a spool file. If there are no files enqueued for the TCPASCII-type link, AXSGET receives a nonzero return code. AXSGET returns control to the PRTGO routine.

If a file is to be processed on the link, AXSGET initializes the necessary data and work areas. AXGET then calls DMTRDROP to get the TAG record and other information about the file. AXSGET then passes control to PROGO, which starts to process the file. PROGO calls DMTRDREP to get the first record of the file. Each remaining file record is obtained by calls to the GETBLOCK routine.

GETBLOCK prepares the print buffer to be sent to the ASCII printer. It places the appropriate buffer prefix into the buffer. If a separator page is required, DMTSEPHD is called to create the page.

GETBLOCK then calls DMTRDREP to obtain each record of the file. The file records are then placed into the buffer. When the buffer is filled, the records are compressed. The PROGO routine then receives control again and calls LINEIO to write the buffer.

When the end of the file is reached, GETBLOCK checks if a trailer page is needed. It then calls DMTSEPTR, as necessary. When this processing completes, PROGO regains control.

Socket Processing

The SOCKIO routine performs the socket SEND and READ functions for the TCPASCII-type link. If the RECVECB is posted, SOCKIO calls the RDDECOD routine which, in turn, calls the ATTNEX exit to receive the data from the socket as it arrives.

ATTNEX

The ATTN exit routine is called when a socket RECEIVE function completes and RECVECB is posted for the TCPASCII-type link. This exit routine can examine the data and optionally pass an RSCS command to the link driver for execution. On return from the RDDECOD routine, a new socket RECEIVE call is issued.

Terminating the Link

The PRTERM routine processes all termination requests for the TCPASCII-type link. It receives control when the LTERECB is posted in the LINKTABL or when a DRAIN command is issued. This routine then issues the SOCKET CLOSE and SOCKET TERMINATE functions. PRTERM then calls DMTLOGCL to close the trace log file, if one was active on the link. It then returns control and a return code to GCS.

Line Printer Daemon (LPD) Link Driver

The line printer daemon (LPD) link driver receives data streams from a TCP/IP LPR client in a TCP/IP network for distribution to a destination within the RSCS network. LPD-type links act as gateway between the TCP/IP network and NJE nodes in the RSCS network, and can be used as a VM based print router.

The LPD link driver is designed to meet TCP/IP RFC 1179. An LPD-type link processes (receives) one file at a time. However, to receive multiple print streams, several LPD-type links can be started on a system all listening for connect requests on the same port number. See [z/VM: RSCS Networking Operation and Use](#) for further information on managing LPD links.

The link driver task is made up of routines and exits in DMTLPD and customer-supplied exit routines. The LPD task uses the following exits to build specific data streams for placement into CP spool and to control characteristics of the spool file, such as spool device type and destination of the file. For more information about the LPD exits, see [z/VM: RSCS Networking Exit Customization](#).

Initialization

Called when an LPD link driver is being initialized.

Print Command Processing

Called when the LPD link driver receives a print job command from an LPR client.

Data Processing

Called when data has been read from a TCP/IP LPR client by the LPD link driver.

End of File Processing

Called when a file has been completely read from a TCP/IP LPR client by the LPD link driver.

Control File Processing

Called for each line of a control file read from a TCP/IP LPR client by the LPD link driver.

Termination

Called when the LPD link driver is terminating.

Initialization

DMTLPD is attached when a START command is issued for an LPD-type link. The task can be attached by the REX task. DMTLPD contains one entry point, DMTLPDEP.

The LPD link can also be attached by the AST task if the link is identified as an auto-start link. However, no files should ever be queued to this link type because the link will never process them. The LPD link should always be started via the START command and not left waiting for auto-start to start it.

When the LPD link driver task is attached, the PRTINIT routine initializes the DWA, ECB lists, and target addresses. It then calls DMTTPAREP to process any parameters.

PRTINIT then calls the LPDBLD routine to load any exit routines that were specified on the START command of the LPD-type link. The LPDBLD routine calls DMTBPLLX to dynamically load each exit routine. If a routine is not found or cannot be loaded, message DMT820E is issued and the link is terminated.

When all the exit routines have been loaded, the PRTINIT routine again receives control. It, in turn, passes control to the initialization exit.

Initialization Exit

The initialization exit routine is called after the exits are loaded by the LPDBLD routine in PRTINIT. The initialization exit is not passed any link parameters and therefore is not enabled to change TCP/IP-specific information.

Sending and Receiving Data

The INITGO routine connects to the TCP/IP stack, then establishes a socket listen request to wait for inbound connects from LPR clients.

The INITGO routine in DMTLPD is the main control routine for the LPD-type link driver. The routine first checks various flags to determine if it must drain the link or process a command.

It then check various ECBs (command, termination, file) to see if it has work to do. If it does not, the routine waits until work is received.

PRTGO issues socket read requests to receive commands and data from the LPR client. Exit routines are called to process and manage policies for the control file commands and data received.

Control File Command Processing Exit

The Control File command processing exit is called for each record of the LPR control file received. The exit routine will determine specific characteristics of the spool file based on the control file commands. In addition, the exit will be responsible for sending positive and negative responses to the LPR client as required by the protocol.

On exit from this routine, the print vector might contain ASCII data to be transmitted.

Data Processing Exit

The RDATAF routine in DMTLPD is called to read data sent from the LPR client. For each chunk of data read, up to 1024 bytes, the data processing routine will be called to handle the data. The exit is responsible for determining correct record boundaries for each line. The routine also carries out any appropriate translation of the print data from ASCII to EBCDIC. It can translate, ignore, or add data to the record. When the link driver regains control from this entry point, the print vector will contain either a record of data to be spooled including the CCW opcode or a response message in ASCII to be sent back to the LPR client. If it contains a record of data to be spooled, the PUTBLOCK routine is called to write the data into spool.

PUTBLOCK will call DMTAXMRQ to open a spool file for processing on the first call, then call DMTUROEP to write the data to spool.

When all data has been received, the end of file processing exit is called. The JOBCLOSE routine is then called to complete processing. JOBCLOSE will call the PUTCLOSE routine, which calls DMTUROFL to write any remaining data to the spool file. JOBCLOSE then calls DMTAXMRQ to close the spool file, which will then route the file for delivery. On error conditions, the PUTPURGE routine is called to close and purge the spool file via a call to DMTAXMRQ.

End of File Processing Exit

This routine allows for additional information to be spooled for the print file and a response to be sent back to the LPR client. It is called after all data is received on the socket. This could be before or after the control file has been received, depending on the order the LPR client has sent the control and data files.

Terminating the Link

The PRTTERM routine processes all termination requests for the LPD-type link. It receives control when a DRAIN or STOP is issued. It can also receive control if a serious TCP/IP error is detected by the link. PRTTERM issues the SOCKET CLOSE and TERMINATE functions to close the TCP/IP socket interface for the link. It then closes the trace log file and returns control to GCS.

Line Printer Remote (LPR) Link Driver

The line printer remote (LPR) link driver sends data streams to a TCP/IP line printer daemon for distribution or printing in a TCP/IP network. LPR-type links act as a gateway between NJE nodes in the RSCS network and the TCP/IP network. They do so by mapping NJE routes to LPR hosts and printer queues.

The LPR link driver is designed to meet TCP/IP RFC 1179. An LPR-type link processes one file at a time. However, to create multiple LPR streams, several LPR-type links can be started on a system and these links can be defined as members of a ROUTE group. See [z/VM: RSCS Networking Operation and Use](#) for more information about ROUTE groups.

The link driver task is made up of routines and exits in DMTLPR and customer-supplied exit routines. The LPR task uses the following exits to build specific data streams for transmission and to control the remote host and port to which the transmission is destined. For more information about the LPR exits, see [z/VM: RSCS Networking Exit Customization](#).

Initialization

Called when a LPR link driver is being initialized.

TAG Processing

Called when the LPR link driver opens a new spool file.

Record Processing

Called when a record is read from the input spool file for the LPR link driver.

End of File Processing

Called when a file has been completely read from the input spool file for the LPR link driver.

Control File Processing

Called when the LPR link driver needs a control file.

Termination

Called just before link driver termination.

Initialization

The LPR task is attached when a START command is issued for an LPR-type link. The task can be attached by the REX task or by the AST task, if the link is identified as an auto-start link. DMTLPR contains one entry point, DMTLPREP.

When the LPR link driver task is attached, the PRTINIT routine initializes the DWA, ECB lists, and target addresses. It then calls DMTLPREP to process any parameters.

PTINIT then calls the LPRBLD routine to load any exit routines that were specified on the START command of the LPR-type link. The LPRBLD routine calls DMTBPLLX to dynamically load each exit routine. If a routine is not found or cannot be loaded, message DMT820E is issued and the link is terminated.

When all the exit routines have been loaded, the PTINIT routine again receives control. It, in turn, passes control to the initialization exit.

Initialization Exit

The initialization exit routine is called after the exits are loaded by the LPRBLD routine in PTINIT. The INIT routine can change TCP/IP-specific information that was defined on the PARM statement of the LPR-type link. The routine is passed an address pointer to the following areas that contain line printer daemon information:

- Remote host IP address
- Remote host port
- Remote printer queue name
- Link driver flag fields, including the following:

PASS=

Specifies if RSCS perform 1 or 2 passes through the file

CTL1ST

Specifies if RSCS should send the control file before sending the data in the file

- Fully qualified host name
- User defined prefix string
- User defined suffix string
- User defined filter
- User defined translate table
- User defined separator page setting

Sending and Receiving Data

The PRTGO routine in DMTLPR is the main control routine for LPR-type link driver. The routine first checks various flags to determine if it must drain the link, hold a file on the link, or determine if a file is currently held on the link.

It then check various ECBs (command, termination, file) to see if it has work to do. If it does not, the routine waits until it receives work.

PRTGO calls AXSGET to get a file to process. After it obtains a file, PRTGO calls PASSUSER to read any user defined keywords from the spool file. PASSUSER will also check if the spool file form matches a form entry defined by the LPRXFORM exits. Fields that will be passed to the exit routines will then be initialized. The TAG exit routine is then called.

TAG Processing Exit

The TAG processing exit examines a file's TAG element. Based on a file's characteristics, the exit routine can create header lines or separator pages. The exit routine inserts the characters that RSCS passes to the TCP/IP line printer daemon for processing into the print record portion of the print record vector. This exit can be called twice if RSCS performs two passes through the file. The LPR TAG processing exit is also called. Using this exit, an exit routine can customize where individual files are printed in a TCP/IP network by overriding the values specified on the PARM statement for the LPR-type link, by the user, or defined within the LPRXFORM exit.

On exit from this routine, the print vector might contain ASCII or binary data to be transmitted.

If PASS=2 was specified, PROGO will call the PASS1 routine. This routine reads the entire file and calls the receive and end of file exits, which count the number of bytes of data that will be sent. PROGO will

then issue SOCKET functions to connect to the remote port on the host. If an error occurs, the file will be queued in hold status or a retry attempt will be made. PROGO then determines if there are any reader commands pending; if commands are pending, they will be processed before the file is processed.

After any reader commands are processed, the GETBLOCK routine is called to begin file processing. The GETBLOCK routine performs all file processing. This routine calls the Record processing exit for each record in the current file.

Record Processing Exit

The record processing routine carries out any appropriate translation the print data from EBCDIC to ASCII or binary. This entry is called for each record of the spool file. It can translate, ignore, or add data to the record. When the link driver regains control from this entry point, the data from the print record moves into the link driver's output buffer. When it is full, the DMTLPR link driver calls SOCKWRT, which performs a SOCKET SEND function to send the buffer to the TCP/IP line printer daemon.

When the end of the file is reached, the RESETEOF routine is called; this routine, in turn, calls the end of file processing exit.

End of File Processing Exit

This routine allows for additional information to be sent to the TCP/IP line printer daemon. It is entered after the last spool file record has been processed. At this time, any specific device-dependant information (for example, feed paper to the top of a new page) can be transmitted.

For all files sent on an LPR-type link, a control file is also sent. The control file exit routine is called once for each file sent on the LPR-type link. Its processing depends on the settings of the CTL1ST and PASS parameters.

Control File Processing Exit

If the CTL1ST and PASS=1 flags are set, the control file routine exit is called before any of the spool file is read. If CTL1ST is in effect and PASS=2, the exit is called after the first pass and before the second pass through the file. If neither CTL1ST nor PASS=1 is set, the exit routine is called after the entire data file has been transmitted.

Terminating the Link

The PRTERM routine processes all termination requests for the LPR-type link. It receives control when a DRAIN or STOP is issued. It can also receive control if a serious TCP/IP error is detected by the link. PRTERM issues the SOCKET CLOSE and TERMINATE functions to close the TCP/IP socket interface for the link. It then closes the trace log file and returns control to GCS.

Chapter 8. Workstation Link Drivers

This chapter describes the processing of the RSCS workstation link driver tasks. RSCS can communicate with nonintelligent and intelligent (programmable) workstations over SNA and non-SNA links.

RJE Workstation Link Driver

DMTNPT is the primary module for the RJE link driver task. An RJE-type link can emulate the following types of remote job entry (RJE) workstations: 2770, 2780, 3770, or 3780. DMTNPT uses the RJE protocol defined by each device to communicate with the workstation; communication can only occur in one direction at a time.

DMTNPT allows the remote workstations to control the link. The remote devices do not bid for control of the link, which may require intervention by remote operators to break any contention. When a workstation no longer needs to send data on the link, DMTNPT may gain control of the link.

DMTNPT provides the following functions between the local node and remote workstation:

- Accesses guest virtual machines, such as VM Batch, which lets users process jobs without being on the VM host.
- Gives a remote workstation access to any node in the RSCS network.
- Processes input data from a remote workstation as real card reader input for any guest virtual machine running in z/VM.
- Sends print or punch data to the remote workstation.
- Receives commands and returns messages to a workstation's printer.

Initialization

DMTNPT is attached when a START command is issued for an RJE-type link. The REX task calls DMTBPL to attach DMTNPTEP, the main entry point for the RJE link driver task (see [Figure 44 on page 93](#)).

```

+-----+ +-----+ +-----+ +-----+ +-----+
| START +---->| DMTREX +---->| DMTCMY +---->| DMTBPL +---->| DMTNPT |
+-----+ +-----+ +-----+ +-----+ +-----+

```

Figure 44. Initializing the RJE Link Driver Task

DMTNPT routines initialize and process sign-on records for the link. The sign-on card identifies the type of RJE device and the communications protocol for the connection.

If the BUFF, CMPR, LPRT, TRS, or TYPE parameters are specified and the PASS parameter is not specified on the START command or LINK statement for the link, the workstation does not need to send a sign-on card. In this case, the NPTLINK routine processes the options that define the correct configuration for the workstation and enable communication over the RJE-type link.

Receiving and Sending Data

After the RJE-type link initializes and sign-on processing completes, DMTNPT begins its main processing cycle. The NPTGET routine monitors an ECB list to determine the type of work to be processed. When an ECB is posted, NPTGET calls an appropriate routine.

When messages are enqueued on the link, NPTGET calls the MSGPROC routine, which bids for control of the link. If it gains control of the link, MSGPROC sends the message. If it is unsuccessful, MSGPROC responds to any error conditions. After the message is processed, control returns to NPTGET. If no messages are processed, DMTNPT can then process files. Messages are not sent when a file is processed; they are only sent between files.

When a file is enqueued on an RJE-type link, the NPTSTART routine is called. NPTSTART then bids for control of the link. When it receives control, NPTSTART calls the GETBLOCK routine. This routine, in turn, calls DMTRDREP to obtain each record from the file. The records are placed into a transmission buffer, which can be used for a single record or multiple records, depending on options specified on the sign-on card. When full, the buffer may be compressed or defined as a transparent data buffer. When the buffer is sent to the remote workstation, NPTGET again receives control.

If no files are enqueued on the link, the NPTDINIT routine gains control. If TPOLL=NO has been specified on the link, the routine puts up a prepare read CCW. If TPOLL=YES is specified, the routine polls the remote node. In this case, this routine interacts with the workstation after it completes three consecutive read I/O operations that end with a sensed time out. This is immediately followed by an I/O operation that exercises a PREPARE sequence to bid for control of the line. After acknowledgement, the link is reset to control (free) state. This returns control of the link driver to NPTGET.

To send data to the RJE-type link, the workstation can bid for control of the link at any time. When this occurs, DMTNPT suspends any output as soon as possible and gives control to the remote workstation. DMTNPT then prepares to receive a data buffer from the workstation. DMTNPT processes each record in the buffer as a card image. An ID card, which identifies the file's destination, must be the first card in each input file stream.

Building Data Streams

GETBLOCK calls the AXSGET routine, which calls DMTAXMRQ to open the file. If the file is opened successfully, DMTNPT attempts to send the file. Once control of the line is acknowledged, subsequent calls to GETBLOCK are performed. GETBLOCK calls DMTRDREP to get each data record until the end of the file is reached. When the file is sent, DMTNPT sends an EOT request to the workstation to reset control of the link.

I/O Processing

The LINEIO routine receives all I/O requests on the RJE-type link. Each routine that calls LINEIO must provide the correct I/O (CCW) string for each request. LINEIO passes this I/O request to the XECUTE routine, which calls GCS to process the I/O operation. GCS then returns an I/O interrupt to LINEIO, which in turn calls DMTLOGEP to log these transactions.

Terminating the Link

The NPTTERM, LINEDIS1, LINEDIS2 and LINEDROP routines perform termination processing for RJE-type links. The link may end when a STOP or DRAIN command is issued. The RJE-type link may also end if it receives three consecutive incorrect sign-on requests from the remote workstation or if a severe I/O error occurs.

These routines may quiesce all transactions on the link or immediately disable the link. The termination may request an automatic restart of the link or may require operator intervention. When the link ends, the remote workstation signs off from the connection. To resume communication, the RJE-type link and the workstation must again exchange sign-on information.

MRJE Workstation Link Driver

DMTSMML is the primary module for the multi-leaving RJE workstation (MRJE) link driver task. MRJE-type links support two-way, alternate data traffic on a BSC communications link.

MRJE-type links can function in *host* or *remote* (workstation) mode. The function is determined by the START command options specified for the link. In host mode, MRJE-type links support programmable workstations, which must appear as a Job Entry Subsystem (JES2) programmable workstation (HASPRB360). In remote mode, MRJE-type links emulate a workstation and can communicate with VS1 Remote Terminal Access Method (VS1 RTAM), JES3, or JES2 systems.

Two MRJE-type links can also communicate with each other, if one is in host mode and the other is in remote mode to JES2. However, RSCS does not consider this environment to be a peer-to-peer connection.

Initialization

As [Figure 45 on page 95](#) shows, DMTSML is attached when the REX task receives START command for an MRJE-type link.

```

+-----+ +-----+ +-----+ +-----+ +-----+
| START +---->| DMTREX +---->| DMTCMY +---->| DMTBPL +---->| DMTSML |
+-----+ +-----+ +-----+ +-----+ +-----+

```

Figure 45. Initializing the MRJE Link Driver Task

DMTSML contains one entry point, DMTSMLEP. Routines in the module perform functions, such as initialization and receiving and sending data. These routines are described in the following sections.

Host Mode

When an MRJE-type link is started in host mode, the SMLINIT routine starts the line I/O process. This routine reads the sign-on card sent by the remote workstation. After the link is enabled, a 2 byte response (STX ENQ) must be the first buffer sent by the remote node. If DMTSML receives any other response, it ends the link. When it receives this response, DMTSML sends an acknowledgement (ACK0) to the remote workstation. Before processing any additional data, DMTSML must read the next sign-on buffer from the remote node.

Remote Mode

If the MRJE-type link is started in remote mode, DMTSML builds a sign-on card (logon for VS1 RTAM). Before sending the sign-on card in a buffer, however, DMTSML sends the response STX ENQ to the host node. The host must acknowledge the receipt of the buffer by sending a ACK0. The link driver does not accept any other responses. When the correct responses are received, the sign-on process may start.

Receiving and Sending Data

DMTSML contains the following subroutines, which send and receive data on the MRJE-type link:

\$JRTN1

Processes input buffers, received from a workstation, as punch data.

\$PRTN1

Processes input buffers, received from a host node, and writes the records to a virtual printer.

\$RRTN1

Reads spool files and ensures a maximum record size is maintained for the remote workstation. It then calls \$PUT to fill a transmission buffer with these records.

\$WRTN1

Writes input messages to the RSCS operator, if the link is in remote mode; if in host mode, the routine passes commands to the REX task for processing.

\$CRTN1

Processes input buffers that contain a control record, which is indicated by record control byte (RCB) F0.

\$URTN1

Processes input buffers, received from a host, a virtual punch data.

Building Data Streams

The \$RRTN1 routine processes spool files on the MRJE-type link. If the link is in host mode, files must be in print or punch format. Print records are blocked to a length of 150-bytes; punch records are forced to

80-byte lengths. \$RRTN1 then calls \$PUT to place the records in transmission buffers that are sent to the remote node.

If the link is in remote (workstation) mode, the spool files must be in punch format. Workstations can only send card images to the host node; no other types of files are processed.

\$RRTN1 calls AXSGET to open the file and obtain a record. It then calls VMDEBLOK, which in turn calls DMTRDREP to get each remaining record until the end of the file is reached.

I/O Processing

The COMSUP routine performs all I/O processing on the MRJE-type link. COMSUP is responsible for the recovery and processing of TP buffers. It processes the I/O interrupts that result from each transaction that occurs on the link and calls DMTLOGEP to log these transactions.

Terminating the Link

MRJE-type links end when a DRAIN, STOP, or SHUTDOWN command is issued. When the MRJE-link ends, the DEOJ and EOJ routines attempt to quiesce the link or immediately end it. DEOJ calls EOJ for final termination processing after it has attempted to quiesce the link; EOJ completes the termination. If EOJ is called directly, the link will not disable and will not restart automatically.

SNARJE Workstation Session Driver

DMTSJE, the SNARJE session driver task, provides support for a subset of the SNA LU_T1 protocol for a System 36 MRJE workstation. SNARJE-type links can send and receive input from remote nodes to the workstation over the SNA LU_T1 session. SNARJE-type links can also give RSCS users access to guest virtual machines, such as VM Batch. This lets RSCS users process jobs without being locally attached to the local node.

DMTSJE provides the following functions between VM and remote sessions:

- Receives input data streams from the remote session and spools it for any guest virtual machine running in z/VM
- Sends spool files to the remote device
- Receives commands and messages from the remote session
- Sends messages and responses to the remote session
- Allows a remote session to access any node in a network.

A SNARJE-type link does not communicate directly with the workstations; rather, it communicates with VTAM, which controls the devices. VTAM first processes all input from the workstations. When it verifies the data, VTAM passes the data to DMTSJE. VTAM also processes I/O interrupts for the workstations. DMTSJE only processes return codes and information from VTAM, which indicates the state of the SNA session.

When a SNARJE-type link establishes an SNA session, several VTAM exit routines are scheduled. These exit routines, include: DFASY, LOGON, LOSTERM, NSEXIT, RELREQ, RESP, SCIP, and TPEND. DMTSJE processes the DFASY exit. DMTVXT processes the other VTAM exits and posts an ECB in DMTSJE when an exit is scheduled (see [“VTAM Exit Routines”](#) on page 57 for more information on VTAM exits).

Initialization

DMTSJE is the primary module for the SNARJE session driver task. Unlike RJE-type and MRJE-type links, SNARJE-type links are attached by the SCT task when a NETWORK START command is issued. After the SCT task completes its processing to establish a SNA session for the link (see [“SNA Control Task”](#) on page 55 for more information on the SNA Control Task), DMTSJE receives control from GCS.

DMTSJE contains one entry point, DMTSJEEP. Other DMTSJE routines initialize the link and perform other processing. The SJEINIT routine obtains storage for save areas and work areas (DWA and secondary DWA). It also processes the START command parameters, establishes the ESTAE exit (DMTCTCU) for this task, and initializes the ECB list.

SJEINIT then verifies fields in the BIND image for the SNA session. It obtains this information from the CINIT RU request. This request is generated when the SCT task issues a SIMLOGON macro to request an SNA session. If the BIND information is acceptable, SJEINIT issues an OPNDST response to start the SNA session with the remote node.

Receiving and Sending Data

After SJEINIT completes initialization processing, the SJEGO routine receives control. SJEGO monitors a list of ECBs to determine when work arrives for the session driver.

Receiving Data

The LRECECB ECB identifies when the session driver has received input data from the workstation. The SCT task posts this ECB when the RECEIVE request, which it issues when establishing the session, completes. When LRECECB is posted, the RECPROC routine receives control.

RECPROC then issues a RECEIVE SPECIFIC macro to process the input data. This data may consist of commands, messages, or files. RECPROC calls the AXSPUT routine or DMTCOMNQ to send the data to its appropriate destination. RECPROC also receives return codes and feedback information from VTAM. It then issues messages, containing this information, to the RSCS console. After processing all input data, RECPROC issues the RESETSR VTAM macro and returns control to SJEGO.

Sending Data

If a message is enqueued on the SNARJE-type link, its LMSGECB is posted. The MSGPROC routine receives control and verifies the status of the SNA session. If appropriate, MSGPROC calls the SENDRU routine to issue the SEND macro for the FM header.

MSGPROC then calls DMTCOMDQ to remove the message element from the message queue. Extra blanks are then removed from the message text and additional SNA information is placed in the message buffer. Finally, the SENDRU routine sends the message to the remote node. When all messages are processed, control returns to SJEGO.

The AXSGET routine receives control when a file is enqueued on the SNARJE-type link. This routine issues VTAM SEND macros to send the data to the remote workstation. It calls DMTRDREP to obtain each record in the file. The records are then placed in a buffer (RU), with the correct flags and indicators set in the RPL.

When the end of the file is reached, the AXSPURGE routine is called to purge the file from the local node; control then returns to SJEGO.

The CMDPROC routine processes each command that is enqueued to the SNAJRE session driver. Valid commands for the SNARJE session driver include: BACKSPACE, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, READY, START, and TRACE. An ECB is posted for each command. When CMDPROC has completed its processing, it returns control to SJEGO.

Building Data Streams

When a file arrives in the link, the AXSGET routine calls DMTAXMRQ to open the spool file. Control returns to SJEGO, which then calls DMTRDREP to obtain a record in the file. This record becomes the TAG record of the file. The GETBLOCK routine continues to call DMTRDREP until all records in the file are obtained.

GETBLOCK places any appropriate SNA headers in the buffer and prepares to send the buffer to the remote node. For print files, GETBLOCK calls DMTSEPHD and DMTSEPTR to build any required separator pages. Each record is placed in a transmission buffer; when the buffer is filled, the records are compressed. GETBLOCK then calls SENDRU to send the file to VTAM.

I/O Processing

VTAM performs all I/O processing on the SNARJE-type link. DMTSJE routines issue the RECEIVE and SEND macros to interact with VTAM. Each routine also processes any return codes issued by these VTAM macros. DMTSJEDF and DMTVXT process any exit routines driven by VTAM.

Terminating the Link

The SJETERM routine processes all termination requests for the SNARJE-type link. It receives control from SJEGO when the LTERECB or TERMECB is posted for the session driver or when RSCS must release the SNA session. SJETERM issues the VTAM CLSDST macro to end the session or to end and release the session.

If RSCS must release the SNA session, the SNARJE-type link remains active and waits to re-establish communication with the session. If the session is to reactivate, DMTSJE issues the SIMLOGON request and attempts a new OPNDST request; the SCT task does not issue this request. If these requests are successful, SJEGO regains control and processing on the session driver continues.

When the session driver ends, SJETERM calls DMTLOG to close the I/O transaction log. It then returns control and a return code to GCS.

Chapter 9. Special Purpose Link Drivers

This chapter describes the list processor and NOTIFY, UFT, and UFTD link driver tasks. Unlike the link drivers described in the preceding chapters, these special link drivers do not interact with a specific peer.

List Processor

The list processor task processes distribution lists and converts them into NJE headers. The list processor also separates files that contain multiple data set headers.

The list processor is defined by a LISTPROC-type link; its parameters are similar to those supported for NJE-type, SNANJE-type, and TCPNJE-type links. The list processor task also calls many of the common NJE routines used by the NJE, SNANJE, and TCPNJE link driver tasks; see [“Common Networking Structures”](#) on page 65 for more information.

Initialization

DMTLIS is the primary module of the list processor task. The REX task attaches this task, when it receives a START command for a LISTPROC-type link (see [Figure 46 on page 99](#)).

```
+-----+ +-----+ +-----+ +-----+ +-----+
| START +---->| DMTREX +---->| DMTCMY +---->| DMTBPL +---->| DMTLIS |
+-----+ +-----+ +-----+ +-----+ +-----+
```

Figure 46. Initializing the List Processor Task

Like all link drivers, DMTLIS specifies DMTMANDE as its ESTAE routine, initializes an NDWA, and calls DMTNTRSB to process its link parameters. DMTLIS also obtains storage from GCS for a buffer, which it uses to send and receive data. DMTLIS then calls DMTNCR to initialize the NJE-related storage areas (RIBs and TIBs). DMTNCR also calls DMTAXMRQ with a request to initialize the transmission algorithm specified for the LISTPROC-type link.

Receiving and Sending Files

When a file is enqueued on the link, DMTLIS calls DMTNTRSB to fill a buffer with records from the file. DMTLIS then calls DMTNRVEB to empty this buffer. This buffer is continually filled and emptied by calls to DMTNTRSB and DMTNRVEB, respectively. If the file contains an unprocessed distribution list, DMTNTR creates a data set header for each entry in the distribution list.

Using Common Networking Routines

DMTLIS calls DMTNTRSB to fill a buffer to be transmitted. As part of its processing, DMTNTRSB determines if it is being called from a list processor task. It then identifies the files it is reading from spool by one of the following categories:

- Files originating on the local node or containing one data set header (these files contain a distribution list).
- Files containing overflow data sets from prior distribution list processing
- Files containing multiple data set headers (these files need to be further separated).

If the file does not contain multiple data set headers, DMTLIS assumes that it contains a distribution list. If these files do not contain a distribution list, RSCS may purge the input file. DMTLIS then calls DMTNTR to convert each entry in the distribution list into a data set header before placing it in the transmit buffer (see [Figure 47 on page 100](#)). When processing a SYSOUT file from a remote NJE node, DMTLIS bases the new data set header on the original data set header that accompanied the file.

Input:	Output:
<pre> +-----+ NODEA USERA NODEB USERB Common Data +-----+ </pre>	<pre> +-----+ Job Header (common) Data Set Header for NODEA USERA Data Set Header for NODEB USERB Common Data Job Trailer (common) +-----+ </pre>

Figure 47. Simple Distribution List Processed by DMTNTRSB

If the distribution record begins with a number, the specified number of records that follow the distribution record is called the *private* section. The data in this section is only sent to the destination specified in the record (see [Figure 48 on page 100](#)).

Input:	Output:
<pre> +-----+ 2 NODEA USERA Private Data A 3 NODEB USERB Private Data B NODEC USERC Common Data +-----+ </pre>	<pre> +-----+ Job Header (common) Data Set Header for NODEA USERA Private Data A (2 records) Data Set Header for NODEB USERB Private Data B (3 records) Data Set Header for NODEA USERA Data Set Header for NODEB USERB Data Set Header for NODEC USERC Common Data Job Trailer (common) +-----+ </pre>

Figure 48. Distribution List with Private Sections

DMTLIS may call DMTNTRSB several times to process the distribution list and transform its entries into data set headers. Because the data set headers for NODEA (USERA) and NODEB (USERB) in [Figure 48 on page 100](#) must be repeated before the common data section, copies of the data set headers are kept in a chain anchored in the TIB for the stream on which the file is being sent.

If a file already contains multiple data set headers, it is processed as if it were being sent on an NJE-type, SNANJE-type, or TCPNJE-type link. In this case, however, DMTLIS separates more copies of the file as part of *fanout* processing.

The number of unit record output devices DMTLIS obtains for each input stream determines the number of copies it can separate from each file. This number is set for all RSCS links by the MAXURO keyword on the OPTION statement. The number can be overridden for an individual link by the MAXURO option in a link's PARM statement. See [z/VM: RSCS Networking Planning and Configuration](#) for more information about these statements.

Using List Processor Routines

When processing list processor files, DMTNTR and DMTNRV call DMTLCR to perform the following functions:

- Validate the syntax of distribution record parameters and determining if a private section follows.
- Copy the existing data set header and, as needed, reformatting it for use as a model for other data set headers.
- Prepare the model data set header to receive information from the distribution list record.
- Determine if a data set should be sent based on the MAXDSH setting for the link.

The MAXDSH value for a link specifies the maximum number of distinct data sets that RSCS can transmit in one file to a peer node. This value is not needed for processing on a LISTPROC-type link. However, when specified on NJE-type, SNANJE-type, and TCPNJE-type links, it can prevent files, which were created with the help of the list processor, from flooding a node that has limited capabilities for processing many data set headers in a file.

Terminating the Link

The list processor task ends when a STOP or DRAIN command is issued for the LISTPROC-type link. It can also end if necessary storage and UR devices are unavailable.

NOTIFY Link Driver

The NOTIFY link driver enables RSCS to hold files sent to unknown nodes or user ID and send a note to the originator of the file. RSCS routes all files with unknown destination to a NOTIFY-type link. The NOTIFY link driver task can then send a note to any network user. A NOTIFY-type link can also be used for files that are undeliverable by an LPR-type or UFT-type link.

Initialization

When a START command is issued for a NOTIFY-type link, the REX task attaches the NOTIFY link driver task at DMTNOTEP. After processing its parameters, DMTNOT attempts to read a template file for the note from any disk accessible to RSCS. The default file name is the link ID of the NOTIFY-type link; this can be changed with a link parameter. The file type must be TEMPLATE. If DMTNOT cannot find the required file, the link driver issues an error message and deactivates.

```
+-----+ +-----+ +-----+ +-----+ +-----+
| START +---->| DMTREX +---->| DMTCMY +---->| DMTBPL +---->| DMTNOT |
+-----+ +-----+ +-----+ +-----+ +-----+
```

Figure 49. Initializing the NOTIFY Link Driver

DMTNOT reads records from the note template file into storage and initially parses each record. This enables DMTNOT to segment each record into text and variables, which are filled in when the note is composed. Each variable begins with an ampersand (&) and ends with a period (.). See [z/VM: RSCS Networking Planning and Configuration](#) for more information. A NOTSEG element represents each segment of text and variable information; NOTSEGs are kept in a simple chain. A NOTSEG that represents the last segment of a record in the template contains an end-of-line marker.

DMTNOT then determines if the purge period has expired for any files since the link driver was last started. It also sets a time to ensure that the NOTIFY driver is notified of the next midnight. When all required processing is complete, the NOTIFY link driver waits for the arrival of new files, commands, midnight, or a termination request.

Generating a Note

When a new file is enqueued on a NOTIFY-type link, the AXM task posts DMTNOT's file arrival ECB. DMTNOT then calls DMTAXMRQ to request to open the input file. On this call, DMTNOT obtains the address of the TAG element that represents the new file.

DMTNOT then calls any Exit 22 routines, which can perform the following functions:

- Suppress the note and purge the file
- Suppress the note and keep the file on the link
- Customize each record in the note (call Exit 23).

If the note is suppressed, DMTNOT calls DMTAXMRQ to close the input file. The file is also purged, if requested. To generate the note, if it was not suppressed by Exit 22, DMTNOT calls DMTAXMRQ to open an output device to which the note will be written.

The note template begins with one or more records in the form of origin user tags (for example, the original tag data RSCS expects on all files spooled to it from local users). DMTNOT then calls routine PROCLINE to create a line of text. It also scans the line to determine if it is blank. A blank line denotes the end of the distribution list and the start of the body of the note.

When it finds the distribution list, DMTNOT determines if one or more distribution records have been specified. If one distribution record is specified, it is parsed and the destination node and user fields are

placed in the output TAG element. DMTNOT then calls DMTUROEP to write the content of the record into the file (in the form of a CCW NOP). Any options on the record are accessible to networking link drivers.

If multiple distribution records are specified, DMTNOT determines if a list processor is defined. DMTNOT only supports single entry distribution lists if no list processor is defined. If a list processor link is defined, the output TAG is targeted to the list processor link. DMTNOT calls DMTURO to write the distribution records (and the trailing blank record) to the output file.

When the distribution list is complete, DMTNOT calls PROCLINE to obtain records for the body of the note. If the NETDATA=NO parameter was specified for the link, the records are written out to a punch device. If NETDATA=YES was specified for the link, DMTNOT calls DMTDDLEP to write each record to the output device. DMTDDLEP creates the body of the note in NETDATA (DMSDDL) format, which is also produced by the CMS SENDFILE and TSO XMIT commands. In the NETDATA format, the note begins with control records that contain the following information:

- Origin node and user ID
- Destination node and user ID
- Origin time.

When DMTNOT calls DMTDDLEP, it passes a zero pointer to a DDL work area to identify the beginning of a note. DMTDDLEP obtains a work area, creates the initial control records, writes them into spool, and returns the DDL work area address to DMTNOTEP. Each time it calls DMTDDL to format a line of the note, DMTNOT passes the address of the DDL work area and a pointer to the record.

After processing all records in the body of the note, DMTNOT passes another zero pointer to DMTDDLEP. DMTDDLEP then creates and writes a trailing NETDATA record to the output device, frees the DDL work area, and returns control to DMTNOT.

When the note is complete, DMTNOT calls DMTUROFL, which removes any remaining CCWs to be written to the output device. DMTNOT then calls DMTAXMRQ to close the output file. When the note is queued to its destination, DMTNOT closes and holds the input file on the NOTIFY-type link. This prevents the file from being processed again and another note being generated. At this point, DMTNOT has completed its processing of the note.

Building Note Records

The PROCLINE routine in DMTNOT transforms NOTSEG elements into a line of text for the note. PROCLINE is passed a pointer to the TAG element that describes the input file being processed. PROCLINE maintains a place holder in the NOTSEG chain that allows it to continue processing the chain where it left off on a previous call. To build a line of text, DMTNOT copies the text within the NOTSEGs and calls routines to substitute information from the TAG and LINKTABL.

Exit 23

If the return code from Exit 22 indicates that the note should be customized, each composed line of text is passed to Exit 23. Exit 23 routines can edit the note text and request that a blank line of text be supplied to the exit routine the next time Exit 23 is called. This method can expand a section of the note, as needed, for any special purposes.

When all NOTSEG elements are processed, Exit 23 is called again. At this point, Exit 23 can indicate if additional lines should be added to the end of the note. When all records in the template have been processed and Exit 23 does not need another blank line of text, the PROCLINE routine issues return code 4. This return code indicates that the note has been completed.

Purging Files

DMTNOT sets a timer to ensure that it checks all files enqueued on the NOTIFY-type link at midnight. Any file that is queued on the NOTIFY-type link for the specified purge period may be purged from the link. The default purge period, 7 days, can be modified by a parameter option.

If a file has exceeded the time limit, DMTNOT calls Exit 36 to determine if the file should be purged. Exit 36 routines can indicate that the file should be held on the NOTIFY-type link. When all enqueued files have been processed and purged, as needed, DMTNOT resets the timer.

Unsolicited File Transfer (UFT) Driver

The unsolicited file transfer (UFT) link driver sends data streams to a TCP/IP UFT daemon for distribution in a TCP/IP network. UFT-type links act as a gateway between NJE nodes in the RSCS network for the delivery of files to users in a TCP/IP network by mapping NJE routes to LPR destinations (hosts and users).

The UFT link driver is designed to meet TCP/IP RFC 1440a. A UFT-type link processes one file at a time. However, to create multiple UFT streams, several UFT-type links can be started on a system and these links can be defined as members of a ROUTE group. See [z/VM: RSCS Networking Operation and Use](#) for more information about managing UFT links.

The link driver task is made up of routines and exits in DMTUFT and customer-supplied exit routines. The UFT task uses the following exits to build specific data streams for transmission and to control the remote host, port, and user to which the transmission is destined. For more information about the UFT exits, see [z/VM: RSCS Networking Exit Customization](#).

Initialization

Called when a UFT link driver is being initialized.

TAG Processing

Called when the UFT link driver opens a new spool file and prior to when the driver connects to the remote daemon.

Record Processing

Called when a record is read from the input spool file for the UFT link driver.

End of File Processing

Called when a file has been completely read from the input spool file for the UFT link driver.

UFT Command Processing

Called when the UFT link driver needs a command file.

Termination

Called when the UFT link driver is terminating.

Initialization

The UFT task is attached when a START command is issued for a UFT-type link. The task can be attached by the REX task or by the AST task, if the link is identified as an auto-start link. DMTUFT contains one entry point, DMTUFTEP.

When the UFT link driver task is attached, the UFTINIT routine initializes the DWA, ECB lists, and target addresses. It then calls DMTPAREP to process any parameters.

UFTINIT then calls the UFTBLD routine to load any exit routines that were specified on the START command of the UFT-type link. The UFTBLD routine calls DMTBPLLX to dynamically load each exit routine. If a routine is not found or can not be loaded, message DMT820E is issued and the link is terminated.

When all the exit routines have been loaded, the UFTINIT routine again receives control. It, in turn, passes control to the initialization exit.

All exit routines, except initialization and termination, are passed a UFTBLOK structure containing the following information:

- Address of a fullword containing the remote host IP address (dotted decimal)
- Address of the 255-character remote host name (fully qualified).
- Address of a halfword containing the remote host port.
- Address of the 256-character user name the file is destined for.
- Address of the 8-character transform name.

- Address of the 256-character translate table.
- Address of the 1-character record format (either 'V ' for variable or 'F ' for fixed) derived from an INMR02 NETDATA control record.
- Address of a doubleword containing the file logical record length from an INMR02 NETDATA control record.
- Address of a doubleword containing the file size, in bytes, from an INMR02 NETDATA control record.
- Address of a doubleword containing the number of files, from an INMR01 NETDATA control record.
- Address of a 23-character field containing the last change date of the file in standard (UTC) or GMT time zone ISO format (yyyy:mm:dd hh:mm:ss), derived from an INMR02 NETDATA control record.
- Address of a 44-character field containing the file name from an INMR02 NETDATA control record.
- Address of a 1-character field containing the server's UFT level.

Initialization Exit

The initialization exit routine is called after the exits are loaded by the UFTBLD routine in UFTINIT. The INIT routine is not passed any link options and therefore cannot change any TCP/IP specific information.

Sending and Receiving Data

The UFTGO routine in DMTUFT is the main control routine for a UFT-type link driver. The routine first checks various flags to determine if it must drain the link, hold a file on the link, or determine if a file is currently held on the link.

It then checks various ECBs (command, termination, file) to see if it has work to do. If it does not, the routine waits until it receives work.

UFTGO calls AXSGET to get a file to process. UFTGO then calls the PASSUSER routine which will look for UFT keywords contained as NOP records in the spool file. Keywords searched for include:

DESTADDR=

Specifies the name of the user to send the file to

HOSTNAME=

Specifies the fully qualified name of host to send file to

HOSTID=

Specifies the dotted decimal IP address of host to send file to

TRANSFORM=

Specifies how the file should be sent, such as ASCII, EBCDIC, BINARY, or NETDATA.

TRANSLATE=

Specifies a translate table to use if the file requires translation

UFTGO then calls the GETNETD routine to obtain NETDATA control information from the file. Then calls GHBNCALL to translate the fully qualified hostname, if specified, into the dotted decimal format. Finally, UFTGO calls the tag exit routine.

TAG Processing Exit

The TAG processing exit examines a file's TAG element. Using this exit, an exit routine can customize where individual files are sent in a TCP/IP network by overriding the values specified on the PARM statement for the UFT-type link, and as specified within the spool file.

The TAG exit does not pass back any data to DMTUFT.

UFTGO will then issue SOCKET functions to connect to the remote host and port, then read the UFT herald to determine the level supported by the daemon. If an error occurs, the file will be queued in hold status or a retry attempt will be made.

After any commands are processed, UFTGO will call the UFT command processing exit to build the command file.

Command Processing Exit

The UFT command file routine exit is called before any of the spool file data records are read. One UFT command will be built and passed from the exit to DMTUFT at a time. DMTUFT will send each command and wait for a positive acknowledgement. When received, the command processing exit will be called again. This process continues until the command processing exit indicates command file processing has completed.

UFTGO determines if there are any commands pending; if commands are pending, they will be processed between each call to the command processing exit.

UFTGO then calls the GETBLOCK routine to begin file processing. The GETBLOCK routine performs all file processing. This routine calls the Record processing exit for each record in the current file. It will also look for any trailing NETDATA control records.

Record Processing Exit

The record processing routine carries out any appropriate translation of the data from EBCDIC to ASCII or binary. This entry is called for each record of the spool file. It can translate, ignore, or add data to the record. The data is passed to the exit in NETDATA format. If the data is not to be sent to the UFT daemon as NETDATA, the exit will be responsible for handling. In addition, the exit will be responsible for dealing with boundary conditions in the NETDATA. When the link driver regains control from this entry point, the data from the print record moves into the link driver's output buffer. When it is full, the DMTUFT link driver calls SENDCMD which in turn calls SOCKWRT, which performs a SOCKET SEND function to send the buffer to the TCP/IP UFT daemon.

When the end of the file is reached, the RESETEOF routine is called; this routine, in turn, calls the end of file processing exit.

End of File Processing Exit

This routine allows for additional information to be sent to the TCP/IP UFT daemon. It is entered after the last spool file record has been processed.

Terminating the Link

The UFTTERM routine processes all termination requests for the UFT-type link. It receives control when a DRAIN or STOP is issued. It may also receive control if a serious TCP/IP error is detected by the link. UFTTERM issues the SOCKET CLOSE and TERMINATE functions to close the TCP/IP socket interface for the link. It then closes the trace log file and returns control to GCS.

Unsolicited File Transfer Daemon (UFTD) Driver

The unsolicited file transfer daemon (UFTD) link driver receives data streams from a TCP/IP UFT client in a TCP/IP network for distribution to a destination within the RSCS network. UFTD-type links act as a gateway between the TCP/IP network and NJE nodes in the RSCS network, and can be used as a VM based file server.

The UFTD link driver is designed to meet TCP/IP RFC 1440A. A UFTD-type link processes (receives) one file at a time. However, to receive multiple data streams, several UFTD-type links can be started on a system all listening for connect requests on the same port number. See [*z/VM: RSCS Networking Operation and Use*](#) for further information on managing UFTD links.

The link driver task is made up of routines and exits in DMTUFD and customer-supplied exit routines. The UFTD task uses the following exits to build specific data streams for placement into CP spool and to control characteristics of the spool file, such as spool device type and destination of the file. For more information about the UFTD exits, see [*z/VM: RSCS Networking Exit Customization*](#).

Initialization

Called when a UFTD link driver is being initialized.

Connect Processing

Called when a connect request has been received from a TCP/IP UFT client for the UFTD link driver.

Command Processing

Called when the LPD link driver receives a UFT command from a UFT client.

Data Processing

Called when data has been read from a TCP/IP UFT client by the UFTD link driver.

End of File Processing

Called when a file has been completely read from a TCP/IP UFT client by the UFTD link driver.

Termination

Called when the UFTD link driver is terminating.

Initialization

DMTUFD is attached when a START command is issued for a UFTD-type link. The task can be attached by the REX task. DMTUFD contains one entry point, DMTUFDEP.

The UFD link can also be attached by the AST task, if the link is identified as an auto-start link. However, no files should ever be queued to this link type since the link will never process them. The UFTD link should always be started via the START command and not left waiting for auto-start to start it.

When the UFTD link driver task is attached, the INIT routine initializes the DWA, ECB lists, and target addresses. It then calls DMTPAREP to process any parameters.

INIT then calls the UFTDBLD routine to load any exit routines that were specified on the START command of the UFTD-type link. The UFTDBLD routine calls DMTBPLLX to dynamically load each exit routine. If a routine is not found or can not be loaded, message DMT820E is issued and the link is terminated.

The data record vector, used to pass data between RSCS and the exit, has a much different usage than for ASCII/TCPASCII, LPR, LPD, and UFT exits. For UFTD, data is passed to the exit in the data record vector. However, upon return, the data record vector will contain an address to the buffer containing data.

When all the exit routines have been loaded, the INIT routine again receives control. It, in turn, passes control to the initialization exit.

Initialization Exit

The initialization exit routine is called after the exits are loaded by the UFTDBLD routine in INIT. The initialization exit is not passed any link parameters and therefore is not enabled to change TCP/IP-specific information.

Sending and Receiving Data

The INITGO routine connects to the TCP/IP stack then establishes a socket listen request to wait for inbound connects from UFT clients.

The INITGO routine in DMTUFD is the main control routine for the UFTD-type link driver. The routine first checks various flags to determine if it must drain the link or process a command.

It then checks various ECBs (command, termination, file) to see if it has work to do. If it does not, the routine waits until it receives work.

UFDGO issues socket read requests to receive commands and data from the UFT client. Exit routines are called to process and manage policies for the control file commands and data received.

Once a connect request from a UFT client is accepted, the connect processing exit routine is called to determine whether the file should be received. This exit is responsible for creating the positive or negative response to send back to the UFT client.

Command Processing Exit

The command processing exit is called for each UFT command received from the client. The exit routine will determine specific characteristics of the spool file based on the UFT commands received. In addition, the exit will be responsible for sending positive and negative responses to the UFT client as required by the protocol. Finally, the exit is responsible for determining how much data will be received.

On exit from this routine, the data record vector will contain a pointer to the command response, in ASCII, to return to the client.

Data Processing Exit

The RDATAF routine in DMTUFD is called to read data sent from the LPR client. For each chunk of data read, up to 4096 bytes, the data processing routine will be called to handle the data. The exit is responsible for determining correct record boundaries for each line. The routine also carries out any appropriate translation of the print data from ASCII to EBCDIC or NETDATA. It can translate, ignore, or add data to the record. When the link driver regains control from this entry point, the print vector will contain either a pointer to the record of data to be spooled including the CCW opcode, or a response message in ASCII to be sent back to the UFT client. If there is data to send to spool, the PUTBLOCK routine is called to write the data into spool. In addition, DMTUFD must be able to handle zero length records passed back from the exit.

PUTBLOCK will call DMTAXMRQ to open a spool file for processing on the first call, then call DMTUROEP to write the data to spool.

When all data has been received, the end of file processing exit is called. Then the PUTCLOSE routine is called which calls DMTUROFL to write any remaining data to the spool file, then calls DMTAXMRQ to close the spool file which will then route the file for delivery. On error conditions, the PUTPURGE routine is called to close and purge the spool file via a call to DMTAXMRQ.

End of File Processing Exit

This routine allows for additional information to be spooled for the print file and a response to be sent back to the UFT client. It is called after all data is received on the socket. This will be after the end of file (EOF) UFT command is received.

Terminating the Link

The UFDTERM routine processes all termination requests for the UFTD-type link. It receives control when a DRAIN or STOP is issued. It may also receive control if a serious TCP/IP error is detected by the link. UFDTERM issues the SOCKET CLOSE and TERMINATE functions to close the TCP/IP socket interface for the link. It then closes the trace log file and returns control to GCS.

Chapter 10. Utility Routines

This chapter describes utility routines that provide services to many tasks at various times during RSCS processing.

General Purpose Routines

Module DMTCOM contains utility routines that provide services to many RSCS tasks. These services include table searches for links and routes, numeric to data conversions, and other work routines.

Table Search Routines

Tasks call entry points DMTCOMLK, DMTCOMGG, and DMTCOMGN to find information in RSCS control blocks.

DMTCOMLK and DMTCOMGG

The AXM task and command processing tasks call DMTCOMLK when processing a link-oriented command. DMTCOMGG is called when a ROUTE command is processed. DMTCOMLK and DMTCOMGG set up the parameter list needed for calls to DMTHAS routines (see [“Hashed Indexing Routines”](#) on page 112). The calling task must supply a pointer to the name of the target link or group.

DMTCOMGN

Tasks call DMTCOMGN to find the routes to a specific node. DMTCOMGN, in turn, calls DMTHASHG (see [“Finding Entries”](#) on page 112 for further details on finding HASH entries). The NODE entry that most closely matches the requested node is returned to the calling task.

When a NODE entry is found, DMTCOMGN searches the ROUTEGRP hierarchy for the node’s root ROUTEGRP entry. DMTCOMGN then returns pointers to the NODE entry and root ROUTEGRP entry for the target NODE to the calling task. For more information about ROUTEGRP hierarchies, see [“Defining Network Structure”](#) on page 17.

Disk File Interface Routine

DMTCOMFI, the disk file interface routine, simplifies file access on any disk accessed in the RSCS virtual machine. DMTCOMFI gives RSCS tasks and exit routines the following services:

- Access to files by ddname or file name/file type
- Conditional and unconditional support for imbed files
- Diagnostic error messages.

The calling task must give DMTCOMFI a FILREQ parameter list. The FILREQ indicates if the file is identified by a predefined ddname or if DMTCOMFI must dynamically define a ddname to access the file (GCS only accesses files by their ddnames).

The FILREQ also contains flags that indicate if IMBED records are to be interpreted and if the IMBED support is conditional or unconditional. If an IMBED is unconditional and DMTCOMFI cannot successfully access a file specified on an IMBED record, it closes all the files associated with the FILREQ block. DMTCOMFI returns a nonzero return code to the calling task. If DMTCOMFI cannot successfully access a file for a conditional IMBED, it ignores the IMBED record.

The calling task may also give DMTCOMFI an initialized MSGBLOK. DMTCOMFI uses this MSGBLOK to issue messages if it finds errors while accessing a file. If it does not receive a MSGBLOK, DMTCOMFI issues return codes to the calling task but does not issue error messages.

When a task requests records from a file that is identified by file name and file type in the FILREQ or by an IMBED record, RSCS dynamically runs a GCS FILEDEF command to associate a ddname with the requested file. Unique ddnames are assigned to each request. As described in [“Dynamic ddname Allocation”](#) on page 27, DMTRES manages the pool of dynamic ddnames available to RSCS. This pool contains all ddnames between @F000@ through @F999@.

For each file it opens, DMTCOMFI reserves a file work area (FILWORKA). All FILWORKAs associated with a FILREQ are placed in a queue that is anchored at the CVT. The FILWORKA for the currently active file is anchored in the FILREQ. The DMTCOM routines add and delete FILWORKA entries from this stack as IMBED records are processed. The stack can contain a maximum of 10 FILWORKA entries. Each FILWORKA contains the task ID of its owning task. RSCS can then ensure that all work areas are freed when the task terminates.

Time-of-Day Conversion Routines

This section describes the DMTCOM routines that convert time-of-day (TOD) values into the format needed by the calling task or exit routine.

DMTCOMTE

DMTCOMTE converts a z/Architecture® TOD value into an EBCDIC string in the form *mm/dd/yyhh:mm:ss*. The z/Architecture TOD is obtained by execution of a STCK instruction. STCK places the TOD value in a 64 bit doubleword aligned field. The 52 high order bits represent the number of microseconds that have passed since the year 1900.

When a routine calls DMTCOMTE, it provides pointers to the following information:

- The TOD value to be converted
- Time zone number that indicates if the Greenwich Mean Time or the local time zone should be used
- The area where DMTCOMTE should place the converted EBCDIC TOD value
- A work area for the conversion algorithm.

DMTCOMTS

DMTCOMTS converts a human readable TOD format (*mm/dd/yyhh:mm:ss*) into a z/Architecture TOD value. The calling routine gives DMTCOMTS pointers to the following information:

- Human readable TOD to be converted
- Time zone number (a nonzero value tells DMTCOMTS that the original TOD value is not supplied in GMT time; the time is adjusted after conversion to z/Architecture TOD clock format)
- Data area where DMTCOMTS should place the converted z/Architecture TOD value
- Work area for the conversion algorithm.

Number/Data Conversion Routines

This section describes routines in DMTCOM that convert EBCDIC information into decimal values and back.

DMTCOMDG

DMTCOMDG receives decimal values from the calling task. It determines if the value is in a specific range and converts it into a binary number. To make this conversion, DMTCOMDG needs the following information:

- Pointer to the decimal value to process
- Length of the decimal value to convert
- Pointer to a valid range limit (low to high).

DMTCOMDG returns the result, a binary number, to the caller in a general register. If it finds an error, DMTCOMDQ returns a nonzero return code to the calling task.

DMTCOMHG

DMTCOMHG receives hexadecimal values from the calling task. It determines if the value is in a specific range and converts the value to a binary number. DMTCOMHG requires the following information:

- Pointer to the hexadecimal value to validate and convert
- Length of the hexadecimal value to convert
- Pointer to a valid range limit (low to high).

DMTCOMHG then returns the result, a binary number, to the calling task in a general register. If it finds an error (for example, a value out of the specified range), DMTCOMHG issues a nonzero return code.

Specialized Routines

This section describes DMTBPL and DMTCOM routines that perform the following functions:

- Load exit routine code
- Validate phone numbers
- Request changes to a link's state
- Process LINKTABL entries for inactive links
- Scan options on the origin user tag
- Identify link driver types
- Process entry points loaded by RSCS.

DMTBPLLX

Tasks call DMTBPLLX to load an exit routine module and return its address. When it is called, R1 points to the entry point name of the exit routine module. On return, DMTBPLLX points R0 to the address of the loaded module. It also issues a return code in R15 to indicate if the module was loaded successfully.

DMTCOMDV

DMTCOMDV validates the phone numbers specified on a link's PARM statement and converts them for use by the link driver. DMTCOMDV removes all blanks and dashes (-) from the number string; it converts asterisks (*) into pause characters that are used by the dial CCW. If the phone number contains characters that are not valid, DMTCOMDV returns a nonzero return code to the calling task.

DMTCOMLS

DMTCOMLS is called when a link changes state (for example, connect, active, or dial-queue). DMTCOMLS determines the link's current state and its state after the request is run. When a link changes state, DMTCOMLS calls Exit 26, which can be used to perform accounting and recovery tasks. For more information, see [z/VM: RSCS Networking Exit Customization](#).

DMTCOMCL

DMTCOMCL processes the LINKTABL entries of inactive links. It also returns to GCS any storage used by override PARM data and storage used by the active PARM data, if the active PARM differs from the default PARM. DMTCOMCL also resets other fields in the LINKTABL to ensure the QUERY command displays accurate information about active and inactive links.

DMTCOMTG

DMTCOMTG scans the system options specified on a file's origin user TAG (see [z/VM: RSCS Networking Operation and Use](#)). DMTCOMTG returns scan results to the calling task by updating flags in the TAG

element provided by the calling task. RSCS only supports the ENQMSG, SENTMSG, and FINALMSG system options.

DMTCOMGD

DMTCOMGD is the common interface routine for tasks that attach link drivers. It searches the EQUATE entries that are anchored in the CVT for a link driver's symbolic name (for example, NJE, SNARJE, and TCPNJE). If DMTCOMGD finds a match, it returns a pointer to the EQUATE entry to the calling task. If it does not find a match, DMTCOMGD returns a nonzero return code to the caller to indicate that RSCS does not recognize the link driver type.

DMTCOMSM

Tasks call DMTCOMSM to enqueue Type L3 message or command elements on a link. As part of its processing, DMTCOMSM also updates the LINKSTAT area for the link.

Hashed Indexing Routines

DMTHAS contains routines to build, maintain, and locate entries in hash tables. RSCS routines DMTCOMLK, DMTCOMGN, DMTCOMGG, and DMTRERSC call DMTHAS to find LINKTABL, NODE, REROUTE, and ROUTEGRP entries, respectively. Exit routines can also call DMTHAS to access any exit-defined data areas.

HASHBLOK

A HASHBLOK, which is mapped by the HASHBLOK macro, defines the characteristics of a hash table. A HASHBLOK creates a hash index for any key within any data area. When a task calls DMTHAS, it must supply a HASHBLOK.

The HASHBLOK macro maps a HASHBLOK. Operands of the macro describe the characteristics of the HASHBLOK (see [z/VM: RSCS Networking Exit Customization](#)). The KEY, KEYLEN, and COLLIDE operands identify the key to the hash table and a pointer to the collision chain. Each data area reserves a fullword field for use as a collision chain.

The HASHBLOK operands also describe other characteristics. The GENERICS operand indicates if the HASHBLOK supports searches for generic entries. The ANCHORS operand specifies the size of the hash index. The CHAIN operand specifies a global chain pointer, which DMTHASHB uses to build a hash table for a list of data areas. Additional operands can generate an eye-catcher and determine if persistent storage is issued for the HASHBLOK.

When DMTHASHB builds a hash table, it sets up a pointer in the HASHBLOK that points to an array of fullword pointers. Each hashed data area is associated with a fullword in this array. The value is used as an index to select a fullword anchor in the table. All the data areas are chained from that fullword using the collision chain pointer in the data area.

Processing Hash Tables

DMTHAS contains several routines that process the hash table and its entries.

DMTHASHB builds a hash table for data areas. It is called by the REX task when RSCS initializes. DMTHASHC clears a hash table and related storage when RSCS terminates. DMTHASHA adds a data area to the hash table. It does not allocate storage nor check for duplicate keys to the hash table. DMTHASHD deletes a data area from a hash table. The calling routine, however, must first locate the data area before it can be deleted.

Finding Entries

DMTHAS also contains two routines that find entries within a specified hash table.

Tasks call DMTHASHF to find a data area in the hash table. DMTHASHF then returns the address of the data area to the calling routine. DMTHASHG finds entries (including generics) in hash tables. If the HASHBLOK passed to the routine does not support generics, however, DMTHASHG will abend.

Storage Management Routines

DMTQSA routines manage stacks of virtual storage. Each stack represents one type of storage element; for example, there are stacks of LINKTABL storage elements and NODE storage elements.

Each stack of storage is anchored to a QSABLOK that contains information about the length of the stack elements. It also describes the GCS GETMAIN options to be used when acquiring and initializing the storage. These options include:

- 4K optimization
- Persistent or nonpersistent storage
- Conditional or unconditional GETMAIN requests to acquire the storage.

DMTQSAAB

DMTQSAAB is called by any task that wants to acquire storage. Each type of storage is identified by a QSABLOK. The calling task points R1 at the QSABLOK and calls DMTQSAAB to get a storage element. If the QSABLOK does not indicate how the storage element should be initialized, DMTQSAAB initializes the storage to zeros; this is similar to the way the GETMAIN macro initializes storage.

If there are elements in the stack, DMTQSAAB initializes the first element, as defined by the QSABLOK. DMTQSAAB then moves a pointer to the storage in R1 and passes it to the calling task. If the stack is empty, DMTQSAAB issues the GETMAIN macro to acquire more storage from a persistent or nonpersistent subpool (as specified in the QSABLOK).

If 4K optimization is specified, DMTQSAAB requests a page of storage from GCS and subdivides this storage into the specified length elements. All but one of these elements is placed on the stack. DMTQSAAB initializes the remaining element and passes it to the calling routine through a pointer in R1.

If the GETMAIN request fails when the QSABLOK specifies that an unconditional GETMAIN should be used to acquire the storage, the calling task terminates with an 804, 80A or 878 abend. If a conditional GETMAIN was requested and storage is unavailable, DMTQSAAB issues return code 4 to the calling task.

DMTQSAUB

When a task no longer needs a persistent piece of storage acquired by DMTQSAAB, it calls DMTQSAUB to return the storage elements to the appropriate stack. The calling task points R1 at the storage element and points R0 at the QSABLOK that defines the stack for that element. DMTQSAUB places the element on the stack and returns to the calling routine.

DMTQSAFA

DMTQSAFA frees all the storage associated with a QSABLOK. If the QSABLOK manages persistent storage, DMTQSAFA issues a GCS FREEMAIN macro to release each element on the stack. If the storage is nonpersistent, DMTQSAFA zeros out the storage stack anchor in the QSABLOK. The GCS task termination routines automatically free all nonpersistent storage acquired by the task.

DMTQSAFE

The QSABLOK macro builds a table that contains the addresses of all the QSABLOKs defined in a module. Because all of the RSCS-defined QSABLOKs reside in DMTQSA, this table also resides in DMTQSA. When RSCS terminates, DMTMAN calls DMTQSAFE to process all RSCS stack-related storage. DMTQSAFE then calls DMTQSAFA to free each QSABLOK in the table.

I/O Interface Routines

DMTIOT contains routines that translate RSCS I/O requests into GCS GENIO requests. The routines also receive any I/O interrupts from GCS. Link drivers call the DMTIOT routines to perform I/O operations. Module DMTURO also calls the DMTIOT routines to perform I/O requests to input/output UR devices (CP spool).

Entry points DMTIOTST, DMTIOTHD, and DMTIOTGE perform the START, HALT, and I/O interrupt handling functions for RSCS. To perform and evaluate an RSCS I/O operation, DMTIOTST and DMTIOTHD use an IOTABLE. The IOTABLE contains pointers to the I/O device and to the virtual channel program to be run on that device. Each entry point issues return code 0. Any status about the I/O operation is reflected by flags and data areas in the IOTABLE control block.

DMTIOTST

DMTIOTST begins processing the channel program. As input, DMTIOTST needs a pointer to the IOTABLE that describes the requested I/O and a pointer to the LINKTABL of the calling link driver task. DMTIOTST then issues the GCS GENIO macro with the START option to begin the channel program on the specified device. The average elapsed time required to complete the I/O operation is kept in the link's LINKSTAT area. This area is anchored to the LINKTABL that is supplied by the calling task.

DMTIOTHD

DMTIOTHD ends the I/O operation on a device by issuing the GCS GENIO macro with the HALT option. DMTIOTHD uses the IOTABLE to point to the device.

DMTIOTGE

When it calls the GENIO START macro, DMTIOTST specifies DMTIOTGE as the exit routine. DMTIOTGE then receives notification of any I/O errors, asynchronous interrupts, and the completion of the I/O operation.

Spool Interface Routines

This section describes routines in DMTRDR and DMTURO that RSCS tasks call to process spool files.

Input Spool Routines

DMTRDR processes input spool files for RSCS. Each routine in the module (DMTRDROP and DMTRDREP) uses the RDR control block as a parameter list. RDR contains the following information:

- Pointer to the file I/O area
- Address of the virtual reader
- Pointer to the area where a logical record is to be placed after it is read from the file I/O area (a SPLINK in CP spool).

DMTRDROP

If a link driver task calls DMTAXMRQ to open a file, it calls DMTRDROP. This routine determines the maximum record length of the data. If necessary, DMTRDROP calls the GETMAIN macro to acquire another work area. DMTRDROP then converts any input from a real card reader into punch output. If requested by a printer, except LPR, or workstation link driver, DMTRDROP also optimizes any CCWs in the first SPLINK it retrieves when the file is opened.

DMTRDREP

As a link driver task builds a TP buffer to send a file, it calls DMTRDREP to get one data record at a time from a SPLINK. DMTRDREP gets new SPLINKs and optimizes the CCW strings, as needed, until the end of

the file is reached. DMTRDREP then calls the FREEMAIN macro to return any alternate work areas to GCS and resets any flags needed for the next file.

Output Spool Routines

DMTURO contains unit record (UR) device output routines; DMTUROEP and DMTUROFL process records being written to an output file.

DMTUROEP

Each time DMTUROEP is called, it writes one file record (a CCW and data) to an output file I/O area (FIOA). DMTAXMRQ builds the FIOA, when requested by a calling link driver task, and passes it to DMTUROEP with every record. The FIOA contains an IOTABLE, which contains a device address and a pointer to a channel program that executes on that device.

DMTUROEP determines if the output FIOA can contain the CCW and the data for the CCW. If there is space in the FIOA, DMTUROEP copies the CCW and the data into the FIOA and updates the CCW's data pointer. If the output FIOA cannot contain the CCW and the data, DMTUROEP points at the IOTABLE and calls DMTIOTST to execute the channel program.

DMTUROFL

If the end of the file is not reached, the FIOA can be reused to process more file records. If the end-of-file marker is reached or if a FLUSH command is being processed, the link driver task calls DMTUROFL. This routine writes the remaining channel program in the FIOA.

NETDATA Conversion Routine

DMTDDL enables RSCS tasks (for example, DMTNOT) to create NETDATA formatted notes. The NETDATA format is also created by the CMS SENDFILE and TSO XMIT commands.

The NETDATA format segments the records of the original file to create a data stream that consists of an 80 byte record. DMTDDL transforms an incoming stream of records into a stream of output records that is compatible with the NETDATA format. Files created by DMTDDL are always written to a virtual punch device, which the calling task must identify in an IOTABLE.

The NETDATA format consists of headers at the top of the file, records that make up the body of the file, and a trailer. The header and trailer records are identified by the label INMR0x. DMTDDL creates the following NETDATA records for RSCS:

INMR01

Header record, containing the following information about the file:

- Logical record length
- Origin node and user ID
- Destination node and user ID
- Origin creation time
- Number of files contained in this file (used for partitioned datasets that cannot be created by DMTDDL).

INMR02

Header record, containing information about the file's size and record characteristics.

INMR03

Header record, containing information about the file's size and record characteristics.

INMR06

Trailing record.

When a routine calls DMTDDLEP, it must provide the following information:

- The IOTABLE for the device to which the NETDATA records are written

- The TAG element for the file to be processed
- A pointer to, and the length of, an input record (maximum length is 255 bytes)
- A pointer to a DDL work area.

On the first call to DMTDDLEP, the pointer to the DDL work area is zero; this identifies the first record in the new file being processed. DMTDDLEP acquires a DDL work area and creates the INMR01, INMR02, and INMR03 header records. When a complete 80 byte record has been filled, DMTDDL calls DMTUROEP to write the record to the device specified in the IOTABLE. This process continues for all following input records.

DMTDDL writes records to punch devices. Because the calling routine may provide records of varying lengths, DMTDDL segments the records into appropriate lengths for the punch device. Portions of the previous record are often kept in the DDL work area until the calling routine provides the next record.

When the end of the file is reached, the calling routine passes a zero input record pointer to DMTDDL. DMTDDL then generates the INMR06 trailing record and pads the rest of the record with zeros. DMTDDL calls DMTUROEP to write the last record to the specified device. It then releases the DDL work area before returning control to the calling routine. The calling routine must call DMTUROFL to flush any remaining CCWs in the file I/O area.

See [z/VM: CMS Macros and Functions Reference](#) for more information on NETDATA formats.

General Parsing Routines

DMTMPT contains routines to parse text strings into individual tokens, convert decimal to binary, and branch to a processing routine for the keyword. Because the CRV contains the addresses of these routines, exit routines can also call DMTMPT routines.

DMTMPTGP

DMTMPTGP parses a parameter from an input string. It identifies a keyword to indicate its start address and length in the original parameter string. DMTMPTGP can also copy up to 16 bytes into an output area and translate this data into upper-case.

When a task calls DMTMPTGP, it supplies a four-word parameter list in R1, which contains the following information; DMTMPTGP updates this parameter list so that it may be called to parse multiple tokens:

- Address of the 16-byte area used to store the upper-cased version of the parsed parameter
- Address of the start of the previously parsed token
- Length of the previous token parsed
- Address of the first byte after the end of the text to be parsed.

DMTMPTGP skips any initial blanks in the input string. If it does not find any text in the string, it issues return code 4 to indicate that a parameter is missing. When it finds the start of first token, DMTMPTGP stores that address in the second word of the parameter list. It then parses the token until it finds a blank or the end of the input string. DMTMPTGP then places the token, upper-cased and padded on the right with blanks, into the 16-byte area pointed to by the first word in the parameter list. If the token does not fit in this area, DMTMPTGP issues return code 8. It then places the length of the token in the third word of the parameter list.

DMTMPTBP

DMTMPTBP parses a parameter from an input string and compares it to a list of keywords defined in the calling routine. Each keyword is associated with the address of a processing routine. If the parameter matches a keyword, DMTMPTBP passes control to the associated processing routine. If it does not match, DMTMPTGP returns control to the calling routine.

When DMTMPTBP receives control, R0 points to a table of keywords and their associated addresses. The keywords contain 16 characters (including blanks) and a fullword address. A blank character indicates the end of the keyword table (see [Figure 50 on page 117](#)).

	LA	R0,KEYS	Address of keywords
	RCALL	DMTMPTBP	Branch appropriately
	BRC	(BAD,MISSING,TOOLONG)	Handle problems
⋮			
KEYS	DS	0F	Fullword align
	DC	C'DEFine	' ,A(CMDDEF)
	DC	C'Query	' ,A(CMDQUERY)
	DC	C'DElete	' ,A(CMDDEL)
	DC	C'StArt	' ,A(CMDSTART)
	DC	C' '	End of table

Figure 50. Sample Keyword Table

The calling routine also supplies a four-word parameter list in R1, which contains the following information:

- Address of the 16-byte area used to store the upper-cased version of the parsed parameter
- Address of the start of the previously parsed token
- Length of the previous token parsed
- Address of the first byte after the end of the text to be parsed.

DMTMPTBP also supports keyword abbreviations. For example in [Figure 50 on page 117](#), if DMTMPTBP parsed the parameters DEF, DEFI, DEFIN or DEFINE, it would pass control to address CMDDEF.

DMTMPTCK

To find a matching keyword for the calling routine, DMTMPTBP calls DMTMPTCK. This routine compares the parameter to the keyword table. If it finds a match, DMTMPTCK issues return code 0 and passes the corresponding fullword value in R0. If not found, it issues return code of 4.

DMTMPTGD

DMTMPTGD calls DMTMPTGP to read a parameter from an input string; it then converts the value from decimal ECBDIC to a signed binary fullword. The parameter, which must be an integer value, may be preceded by a plus (+) or minus (-) sign.

Task Table Service Routines

DMTTAS contains routines that maintain hash indexes for TASKBLOKs. A TASKBLOK describes a type of RSCS task (system, link driver, and auto-answer); a TASKBLOK only represents an active task.

DMTTASKA

DMTTASKA creates a TASKBLOK for an active task. The routine first calls DMTQSAAB to obtain storage for the TASKBLOK. It then places information about the task in the TASKBLOK and adds the new TASKBLOK to the task table.

The REX task calls DMTTASKA when it attaches a system task. DMTTASKA is also called when a link driver (including those specified by a LINKTYPE statement) is attached. It is also called when a port is enabled and when the DUP task transforms itself into the link driver specified by the incoming phone call.

Each calling routine must provide DMTTASKA with a flag to identify the type of task. It must also provide a pointer to the major control block (SYSIDENT, LINKTABL, or PORT) associated with the task. Each control block contains the task ID and entry point name. It also contains the link ID, port address (*ccuu*), or system task name.

DMTTASKD

DMTTASKD deletes a TASKBLOK from the RSCS task table. It is called when a link driver task is detached, a system task terminates, or when link driver task transfers control back to an auto-answer task. It is also called when a port is disabled. DMTTASKD calls DMTQSAUB to deallocate storage used by the TASKBLOK. It then clears the pointer to the deleted TASKBLOK from the task table.

DMTTASKF

DMTTASKF uses a task ID to find a TASKBLOK in the RSCS task table. It is called from DMTCMZ when the ITRACE settings for a task are set or altered. The TASKBLOK is updated to reflect any changes in the ITRACE settings.

DMTTASKG

DMTTASKG finds a TASKBLOK in the RSCS task table, using the GCS FLS macro to find the task ID for the calling task. DMTTASKG is called when the RSCS ITRACE macro is called. It obtains the task ID from GCS; the TASKBLOK contains a work area.

Chapter 11. Parsing Commands and Statements

To process information, RSCS must parse configuration file statements and commands entered by various users. This section describes the parsing facilities (DMTPAF) and central repositories RSCS uses for command and statement syntax.

Defining Syntax

The syntax of all RSCS commands and configuration file statements is defined in two repositories. These repositories are created by the RSCSCMDS and RSCSSTMT macros.

RSCSCMDS contains the syntax for all RSCS commands. The RSCSSTMT macro contains the definitions of configuration file statements. Within each repository, the CDEF, LDEF, and PDEF macros define the syntax of the commands and statements.

Each syntax repository also refers to the RSCSDDEF macro, which defines the valid data types that can be specified on a command and statement. The DDEF macro defines data syntax in the RSCSDDEF macro.

CDEF Macro

Each syntax repository contains a table of CDEF macros that define each RSCS command and configuration file statement. The CDEF macro contains the following information:

- The verb portion of the command or statement
- Minimum abbreviation
- Name of a post-processing routine, if applicable
- The command code for commands that may be run by tasks other than the REX task.

Each entry begins with a prefix that identifies the users who are authorized to issue the command or statements. Configuration file statements defined in RSCSSTMT always contain the L prefix. The prefix has the following meanings:

Prefix

Authorized User

L

RSCS console (local) operator or system authorized alternate operator

R

Remote workstation and link operator

U

User (local or remote).

For example, [Figure 51 on page 119](#) shows that the SLOWDOWN command can only be issued by the RSCS or a system authorized operator. The START command can be issued by the RSCS operator and by a remote workstation or link operator. Finally, any RSCS user or operator can issue the TRANSFER command.

L	CDEF	'SLOWdown'	,CMASL
LR	CDEF	'STArT'	,CMYST,CCODE=STRTCMD,SCANON=YES
LR	CDEF	'STOP'	,CMZSO,SCANON=YES
LR	CDEF	'TRace'	,CMYTR,CCODE=TRACECMD,SCANON=YES
LRU	CDEF	'TRANsfer'	,CMXTR,CCODE=TRNSCMD,SCANON=YES

Figure 51. Sample Entry in the RSCSSTMT Macro

LDEF Macro

The CDEF table in RSCSCMDS and RSCSSTMT is followed by LDEF entries that define the syntax of the command or statement. Commands may have up to three syntax variations; statements only have one valid syntax. The following operands can be specified on the LDEF macro:

Parameter	Purpose
First parameter	Identifies the next LDEF that is processed. If omitted, the current LDEF is processed until a syntax error is found, the end of the line is indicated, or a level of transition is indicated on the PDEF macro.
INIT	Determines if a special routine should be called when this state is initialized.
EOL	Determines if the current state is a valid final state.
EOCL	Contains the name of a routine to call when end-of-line is detected.
OPT	Determines if a PDEF match is required to make a transition to the next state.
COUNT	Contains the number of matches required at this state before the next state is processed.

Command Syntax Variations

The CDEF prefix identifies the syntax variations for each command. Each CDEF entry is associated with LDEF entries that define the command syntax. The labels on the LDEF entries identify the syntax variation. These labels contain the first four characters of the command (or an abbreviation) and one of the following strings:

LCL

Local RSCS console operator or system authorized operator

RMT

Remote workstation or link operator

USR

Any local or remote user.

The parser then uses the information at the appropriate LDEF entry to continue processing the command syntax. For example, [Figure 52 on page 120](#) shows the syntax variations for the TRANSFER command. In this case, the abbreviation XFER identifies the command name.

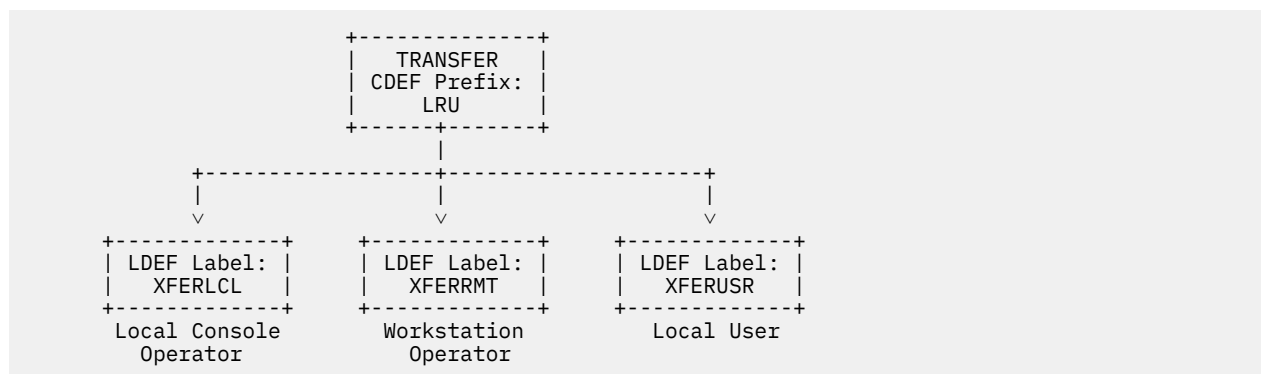


Figure 52. Syntax Structure of TRANSFER Command

PDEF Macro

The PDEF entries that follow an LDEF entry define how the next parameter of a command or statement is processed. The following operands are available to the PDEF macro:

Parameter	Purpose
Keyword, minimum abbreviation	If the first parameter is in quotation marks, it is processed as a keyword with its minimum abbreviation specified by upper case letters.
DDEF name	If the first parameter is not in quotation marks, it is processed as a data definition (DDEF) in the RSCSDDEF macro.
PROCESS	Specifies the name of any special routine that is to process the parameter; it may be used with a keyword or to process the current parameter.
DATADef	Indicates that the keyword specified in the first parameter is followed by another parameter, which is defined by the specified DDEF.
NEXTL	Overrides the default transition to the next state (LDEF) specified on the preceding LDEF.
FLAG	Overrides the default name of the lock byte that is set when the PDEF applies. The default flag name contains the prefix "PAFL" and the first four characters of the keyword or DDEF name.
LOCK	Determines if the lock byte is set when the PDEF applies; by default, the lock is set on unless the FILTER or QMASK options are specified.
FILTER	Creates a filter program, which is later used by the specific processing routine for the command.
QMASK	Specifies a mask, which selects the columns displayed in a columnar message.
UWORD	Specifies a user word whose value is passed to the command processing routine.

As Figure 53 on page 121 shows, the following LDEF and PDEF statements define the syntax of the DISCONNECT command:

```

L      CDEF  'DISConnect',CMZDI
:
:
DISCLCL LDEF  UID,EOL=YES,
        PDEF  'LOG      '
        PDEF  'NOLOG    '
UID      LDEF  EOCTX,EOL=YES
        PDEF  USERID

```

Figure 53. Syntax Definition of the DISCONNECT Command

This example shows that the command can only be issued by the local RSCS operator. EOL=YES indicates that the LOG and NOLOG parameters are optional; if they are specified, the *userid* is also optional. A DDEF macro, which is in the RSCSDDEF macro, describes the USERID data. The parser also sets a byte in the PAFBLOK. This tells the DISCONNECT command processing routine, DMTCMZDI, if LOG or NOLOG was specified when the command was issued.

A PAFBLOK describes the results of parsing a line of text. It contains various locks and pointers to the data areas that contain the node names, link IDs, and other items that are found in the command or statement.

Figure 54 on page 122 shows an example of a PDEF macro specification that overrides the default transition to the next state defined on the LDEF statement. The PDEF statement for the HALT operand overrides the default transition to the next LDEF from OPT to QUIK. This example also shows keywords used with data definitions (DDEFs).

```

L      CDEF 'NETwork ',CMZNE
:
NETWLCL LDEF
PDEF 'STArT ',NEXTL=OPT
PDEF 'HALT ',NEXTL=QUIK
OPT     LDEF EOL=YES
PDEF 'APPLid ',DATADEF=APPLID
PDEF 'Pass ',DATADEF=PASS
PDEF 'RETry ',DATADEF=RETRY
PDEF 'RPLs ',DATADEF=RPLS
QUIK    LDEF EOL=YES
PDEF 'QUICK '

```

Figure 54. Syntax Definition for NETWORK Command

The PROCESS option of the PDEF macro specifies the name of a routine in DMTPAF that processes the parameter. For example, as [Figure 55 on page 122](#) shows, the option defines the syntax of the CPQUERY command.

```

LRU      CDEF 'CPQuery ',CMZCQ
:
CPQULCL LDEF EOCTX,
PDEF 'CPUId '
PDEF 'CPLEVEL ',UWORD=DMTCMZC1
PDEF 'INDicate',UWORD=DMTCMZC2
PDEF 'Files ',UWORD=DMTCMZC3
PDEF 'LOGmsg ',UWORD=DMTCMZC4,FLAG=LMSG
PDEF 'Names ',UWORD=DMTCMZC5
PDEF 'Users ',UWORD=DMTCMZC6,PROCESS=UIDOPTN
PDEF 'Time ',UWORD=DMTCMZC7
CPQURMT LDEF EQU=CPQULCL
CPQUUSR LDEF EQU=CPQULCL

```

Figure 55. Syntax Definition for CPQUERY Command

This example also includes the UWORD parameter. In this case, the user word in the CPQUERY command points to the actual CP command that is run by the command processor. The EQU operand on the LDEF macro indicates that the local console operator, remote operator, and general users can use the same CPQUERY command syntax.

Command PDEF Options

RSCS uses filter programs and column display masks to process the many operands of the QUERY command. The QUERY command processing modules call DMTCQC to build these filter programs. Each filter program processes data areas to determine if a data area meets the criteria of the command.

For example, for the command QUERY LINKS TYPE NJE LINE < 4F2, DMTCQC builds a filter program for the following criteria:

- R15 points to the filter program
- If the LINKTABL passed to the filter program represents an NJE-type link with a line address over 4F2, the filter program returns to the address specified in R14.
- If the LINKTABL does not meet the specified criteria, the filter program returns to the address specified in R1.

DMTCQC defines many types of filter programs. Each filter program is built during repeated calls to DMTCQC, which are specified by FILTER options on the PDEF macro.

Column Masks

Display masks identify the columns that are displayed in a columnar message (see [Chapter 12, “Message Processing,”](#) on page 127 for more information). Because columnar messages can contain up to 64 columns, RSCS may not display all columns in a message at the same time.

The QMASK option on the PDEF macro tells the parser to copy the specified mask in to the PAFQMASK field of the PAFBLOK. The SHOWMASK macro defines each mask, which contain 64-bit fields that map each column defined in the message file for a message number.

When the QMRESET routine is called, it resets the PAFQMASK field to zeros. The mask is then accumulated again, based on the SHOW operands selected for a QUERY command. This mask determines the columns to be displayed in the response message. For example, the RSCSCMDS macro contains the following syntax statements for the QUERY LINKS SHOW NAME LUNAME command:

```

LRU      CDEF  'Query  ',CMQEP
...
QUERLCL  LDEF  ...
          PDEF  'LINKs  ',UWORD=DMTCQXLK,NEXTL=LINK,QMASK=SHLFULL
...
LINK      LDEF  EOL=YES
          PDEF  'SHOW   ',PROCESS=QMRESET,NEXTL=LSHO
...
LSHO      LDEF  EOL=YES,COUNT=(GT,0)
          PDEF  'NAME    ',QMASK=SHLID
          PDEF  'LUName  ',QMASK=SHLALUN

```

Figure 56. Sample QUERY Command

If a mask name is preceded by a + sign, the mask specified after the QMRESET routine is added to the existing mask (a + is the default). The column identified by the mask is then added to the columnar message of that QUERY command. If preceded by a - sign, the mask is removed from the existing mask. The corresponding column is then not displayed in the columnar message.

DDEF Macro

Command and statement tokens may be considered a *keyword* or a *parameter*. Keywords in a command or statement are identified to the command processing routines by fields in the PAFBLOK. Unless a special routine is specified on the PROCESS operand (see [Figure 55 on page 122](#)), all parameters are processed by data definitions.

Each DDEF entry in the RSCSDDEF macro defines the following information about a parameter:

- Type: character, hexadecimal, or decimal
 - If character data, the maximum number of characters accepted
 - If in hexadecimal or decimal, the valid ranges.
- Where the parameter should be stored (hexadecimal or decimal types are converted to binary before being stored)
- If the parameter is stored in the PAFBLOK or in a work area in preparation for post-processing
- The address of a post-processing routine in DMTPAF, if post-processing is specified
- The address of a message routine in DMTPAF that is called if an error is found
- If the parameter being processed is in the current position or if the next token should be parsed and treated as the parameter.

Given the label of a DDEF macro, DMTPAF determines if a parameter is valid and places it in the PAFBLOK. If the parameter is not valid, DMTPAF issues an error message.

Finding Command and Statement Definitions

Before it can parse a command or statement, DMTPAF must first find its definition in the CDEF macros. When DMTPAFCL is called to find a CDEF, it receives the following information:

- Command or statement text string
- Pointer to a syntax definition repository (RSCSCMDS or RSCSSTMT)
- Pointer to a PAFBLOK, which stores the parsed information.

DMTPAF searches the CDEF entries to find a CDEF that matches the first token of the text string. If DMTPAF finds a CDEF, the address of the CDEFBLOK is anchored in the PAFBLOK.

If the SCANON=YES option was specified on the matching CDEF, DMTPAF parses the next token in the string. This enables DMTPAF to process the syntax variations of commands that have multiple formats.

When it completes its search, DMTPAFCL issues a zero return code to the calling routine. If it does not find a match for the command or statement, DMTPAF issues a nonzero return code.

Parsing Commands and Statements

DMTPAFCP is called after DMTPAFCL finds the specified entry. Flags in the PAFFLAG field of the PAFBLOK indicate the command syntax variations to be checked (statements only have one syntax). When DMTPAFCP finds the address of the first LDEF, it starts to parse the command or statement.

If the INIT operand is specified on an LDEF statement, DMTPAFCP calls the specified routine to initialize processing for that state. DMTPAFCP then parses the next token in the text string. When it reaches the end of a line, DMTPAF performs the following checks:

- If EOL=YES was not specified on the current LDEF, the line ended prematurely.
- If EOL=YES was specified on the current LDEF, DMTPAF determines if the COUNT parameter was also specified:
 - If COUNT was specified, DMTPAF checks if the required number of PDEFs specified under this LDEF have been matched. If this number was not matched, the line ended prematurely.
 - If COUNT was not specified or the required match count was met, DMTPAF accepts the valid end-of-line.

If the end of the line is valid, DMTPAF calls any routine specified on the EOCL operand of the LDEF macro. It then returns to the calling routine.

If it finds another token on the text string, DMTPAF searches the list of PDEF entries that follow the current LDEF for a match. If the first parameter in the PDEF is in quotation marks, it is processed as a keyword. The keyword must match the minimum abbreviation for the command or statement. If the first parameter is not in quotation marks, DMTPAF processes it as a data definition, which must match the criteria specified on the DDEF macro.

The first PDEF that is matched determines how DMTPAF processes the token. If the token does not match any of the specified PDEF entries, DMTPAF performs the following checks:

- If OPT=YES was specified on the current LDEF, DMTPAF moves to the next state (LDEF).
- If not specified, the parser tells the calling routine that the token is not valid.

If a PDEF matches a token, DMTPAF performs the following processing:

- If the QMASK option is specified, the PAFQMASK field is updated with the specified mask.
- If a FILTER option is specified, DMTPAF calls DMTCQC to build part of the filter program.
- If a UWORD option is specified, the parser stores the appropriate user word value in the PAFUWORD field in the PAFBLOK.
- If any of the following conditions are true, DMTPAF sets a byte in the PAFLOCKS field to identify the matched PDEF:
 - QMASK, FILTER, or LOCK=NO operands are not specified on the PDEF.
 - QMASK or FILTER operands are specified with the LOCK=YES option.
- If the first token on the PDEF is not in quotation marks, DMTPAF processes it using the specified data definition (DDEF).
- If the DATADEF operand is specified, DMTPAF processes the *next* token on the line according to the specified DDEF.
- If the PROCESS operand is specified, DMTPAF passes control to the specified routine, which processes the token.

If DMTPAF finds errors as it processes the token, it issues a nonzero return code to the calling routine. It may also issue an error message to the command originator if the calling routine provided a MSGBLOK.

When the token is processed successfully, DMTPAF takes the following steps to process the next LDEF statement:

- If the NEXTL operand is specified on the matched PDEF, the specified state is processed as the next state.
- If NEXTL is not specified, the next state specified on the LDEF is taken to be the next state.
- If neither option was specified, DMTPAF remains in the same state (LDEF). For example, this occurs when several options can be selected for a CHANGE command.

Before DMTPAF processes the next LDEF, it checks the criteria specified on the COUNT operand. This ensures that the criteria to leave the previous state has been met. If the criteria has not been met, the parser issues a nonzero return code and returns to the calling routine.

DMTPAF continues processing the tokens on the input line until it reaches the end of the line in a valid state or finds an error. DMTPAF then passes the parsed command or statement to the calling routine in the PAFBLOK provided by the calling routine.

QUERY Command Processing

QUERY commands are initially processed by DMTCMX and parsed by DMTPAF. DMTCMQ is then called to run the command. DMTCQX, DMTCQY, and DMTCQZ may also be called, depending on the operands specified.

When a QUERY command is issued, it returns one or more messages containing the requested information. The messages are issued using a MSGBLOK that is supplied by DMTCMX. QUERY response messages are issued as if SET EMSG TEXT is in effect; the message number is not displayed.

Simple Queries

Some QUERY commands, like QUERY SYSTEM LOCAL, generate a single-line message. For these responses, DMTCMQ places appropriate information in a MSGBLOK. It then calls DMTMGX to issue the response message.

Some commands, like QUERY *linkid* FILES, issue a single-row columnar message. In this case, DMTCMQ identifies the necessary columns for the message in the MSGBLOK. After the response message is issued, DMTCMQ issues a message (TBEND) to identify the end of the columnar message.

Filtering and Columnar Messages

Most QUERY commands, however, create columnar messages that contain many columns. For these QUERY commands, RSCS locates the requested information in various tables and data areas and issues a columnar message.

For example, the command QUERY FILES CLASS M REC > 100 SHOW RSCS displays information about the origin and destination of class M files that contain more than 100 records. DMTCQXFI, which processes this QUERY command, scans all TAG elements to find information about all the files that RSCS is processing.

Each QUERY command processing module runs a filter program to determine the data areas that provide the requested information. DMTCQC builds filter programs as the QUERY operands are parsed. DMTCQC compiles different sections of the filter programs. The FILTER operand of the PDEF macro determines the type of filter program needed for the QUERY command.

Each section of the filter program tests a data area against specific criteria. If the data area meets this criteria, the filter program continues. If the test fails, the filter program branches to an address, specified by the calling routine, to perform appropriate processing (for example, indicate that no requested items were found). When all FILTER operands are processed, DMTCQC is called to create the final section of the filter program. The final section branches to a different address where the calling routine placed all items that matched the filter program.

When a data area passes all filter tests, the requested information is placed in a MSGBLOK that is passed to DMTMGX. The MSGBLOK contains a pointer to the appropriate data area and the message number. The MSGBSHOW field contains a bit mask that identifies the columns that are displayed in the message.

The message building modules use information in the message conversion repository to locate data and determine how it is displayed.

Querying Network Structure

DMTCQY processes all QUERY commands that request information about the structure of the network (QUERY SYSTEM NODES and QUERY SYSTEM GROUPS). A filter program, built by DMTCQC, identifies the nodes for which information is displayed.

The DISPLAY keyword on the QUERY command selects the type of information to be displayed in the response. You cannot select the individual columns that are displayed in the message; they are determined by the QUERY command processing module.

For example, DISPLAY LINKS generates one or more DMT636I messages. Each message has a column for the node, five columns for primary links, and one for an alternate link. The query command processor stores the information in each column of the message as a parameter in the MSGBLOK. If only two primary links are defined, the other columns contain blanks. If six links are defined, the message is issued twice. The first time, the node and the first five primary links are displayed. The second time, the node column contains blanks, the first column displays the sixth primary link, and the other columns contain blanks.

Propagating QUERY Commands

Some QUERY commands propagate through each node in the network to display the requested information. These commands include the QUERY *nodeid* PATH and some QUERY QUEUES and QUERY FILES commands.

After the QUERY command is processed on the local node, DMTCQXPR propagates the command to the next node. DMTCQXPR calls DMTCOMGN to find the root group of the node toward which the command is propagated. The command is then sent on every primary link that is connected to the node. If there are no “connect” primary links, DMTCQXPR sends the command on the alternate link. If the alternate link state is not defined or its state is not “connect”, a message is issued to the command originator and the command propagation ends.

A QUERY command may be sent to a node several times by different routes. DMTCQXPR builds an identifier for each propagating QUERY command. This identifier contains a five-digit serial number; it also identifies the origin node of the command. When a remote node receives a propagated QUERY command, DMTCQXFL determines if the command has been run on the node in the last minute (for a maximum of 500 commands). If the command was executed during that time, it will not be processed on the node again.

Chapter 12. Message Processing

This chapter describes how RSCS processes messages. It also describes the modules, data structures, and files RSCS uses to build and issue messages.

Message Structure

A message is any text that RSCS writes to its console or sends to a user in another virtual machine or on another node. RSCS can issue *text* and *columnar* messages.

All messages are described by a MSGBLOK (see “MSGBLOK” on page 30), which the calling task provides to the message processing modules.

A message can be sent to a combination of destinations. RSCS uses the following routing codes to determine where it should send a message.

Code	Destination	Explanation
R	RSCS console	The message is sent to the RSCS virtual console (the RSCS operator or the user ID specified on the DISCONNECT command). When sent to the RSCS operator, the message contains a time stamp. If another user ID is specified on the DISCONNECT command, the message is delivered by the CP MSGNOH or MSG commands and does not contain a time stamp.
C	CP operator	The message is sent to the OPERATOR virtual machine (CP operator).
O	Any user ID on any node	MSGBLOK fields indicate the destination. If sent to a local user, the message is delivered by the CP MSGNOH, MSG, or SMSG commands. If sent to a remote user, the message is sent through the RSCS network before being delivered to its final destination.
V	Any user ID at the local node	This routing code is similar to the O code. However, here, RSCS ignores the remote node field in the MSGBLOK and the message is always routed to a local user.
P	Same as the R, O, and V routing codes	The message is <i>private</i> and cannot be part of a SET or SETMSG subscription. The message can be routed to any user.

Messages are identified by a message number (between DMT000 and DMT999) for the RSCS server and a severity code (I, W, E, S, and T). Each message also contains fixed text, controlled by message repositories, and substitution fields, which are controlled by values in the MSGBLOK. To issue a message, RSCS takes the following steps:

1. Looks for the format of the message in repositories, using the message number in the MSGBLOK as an index.

These repositories (translation and conversion) issue messages in different languages. Two execs, MCONV and MCOMP, compile the repositories into TEXT decks that can be linked into RSCS load module or into an exit routine package. See “Message Repositories” on page 134 for more information.
2. Builds the message using the format identified in the repositories. Any substitution parameters, passed in the MSGBLOK by the calling task, are also included in the message.
3. Issues the message to all destinations identified by the routing codes.

Text Messages

Text messages indicate that an event has occurred (for example, RSCS has initialized or received a file). Text messages are also issued as responses to some RSCS commands; they are the most commonly-issued type of RSCS messages.

Columnar Messages

RSCS and exit routines can also issue columnar messages (also called table displays) in response to QUERY and EXIT commands. These messages can contain many rows and columns of related information. Columnar messages are also used to display information for some single-line command responses.

Each columnar message contains a *header* and *body* text. The header contains one or more lines of column heading text; some headings may apply to more than one column. The body of the message contains the information placed under each item in the header. Each item in the body is represented by one line of message text.

Most QUERY commands have SHOW options to let you select the columns that RSCS displays in the message. For some QUERY commands, however, you cannot change the columns that are displayed by using the SHOW options. For example, the ACTIVE keyword in the QUERY *linkid* ACTIVE command determines the information RSCS displays. For the QUERY SYSTEM NODES command, the DISPLAY keyword determines the contents of the multiple-line responses. See [*z/VM: RSCS Networking Operation and Use*](#) for more information about the QUERY command.

EMSG Settings

When RSCS issues a message, the CP EMSG setting of the user ID receiving the message determines how the message is displayed. For QUERY command responses in columnar format, however, RSCS edits the message as if EMSG TEXT is the current setting.

National Language Support

RSCS messages can be translated into any language that uses a 256 code-point EBCDIC character set. Using the LANGUAGE configuration file statement, you can select the *local* and *network* language used to issue messages. The local language is used for all messages issued to the local node. RSCS uses the network language for all messages issued to remote nodes.

The *translation repository* contains all the message elements that can be translated into a national language. These elements include:

- Fixed message text (the part of the text that does not change)
- Dictionary terms that are inserted in the message text (for example, "active" or "transferred")
- TOD clock formats
- Columnar message headers (for single and multiple columns).

The translation repository does not contain routing codes or information about message severity or type. Also, the translation repository does not support multiple-format messages that were introduced in Version 2.3 of RSCS.

Command Response Interface

The command response interface (CRI), which may also be called the application programming interface (API), allows execs to associate commands with the response messages they receive from RSCS. [*z/VM: RSCS Networking Operation and Use*](#) describes the format of CRI prefixes.

When a command contains a CRI prefix (for example, `msg rscs (mv.123456) query system links`), the command processor sets flags and fields in the MSGBLOK. This information determines the format of the message response.

When a message is built, the specified CRI prefix can override the local or network language specified on the LANGUAGE statement. As the message is issued, RSCS then adds the CRI prefix to the message.

The first character of the CRI prefix determines how the message is issued to the local node (using the MSGNOH, SMSG, or MSG commands).

When the language-independent form of the CRI is specified, RSCS does not use any information from the translation repository to build the message. Rather, all of the message information is taken from the conversion repository. The conversion repository contains all nontranslatable formatting information and the default routing and severity codes for each message.

If the language independent form of the CRI is specified on a columnar message, RSCS does not display column headers or fixed text in the message. Substitution fields are issued in the order they are defined in the conversion repository. Dictionary items (preceded by the letter D) are referred to by their index value. All TOD clock values in the message are issued in the format: yyyymmddhhmmssuuuzzzzzz

Message Subscriptions

Subscriptions for specific messages can be entered using the SETMSG command or statement and the SET command. These messages may be about a link or may include all RSCS console messages. RSCS uses MONITENT entries to identify each message subscription. See “Message Subscriptions” on page 30 for more information about MONITENT entries. *z/VM: RSCS Networking Planning and Configuration* and *z/VM: RSCS Networking Operation and Use* contain more information.

Processing Messages

To issue a message, an RSCS task calls DMTMGXEP, directly or by invoking the RMSG macro (see [z/VM: RSCS Networking Exit Customization](#)). As [Figure 57 on page 130](#) shows, the DMTMGXEP entry point works with the following modules to process and format the message:

DMTMGXEP

Accepts a message request from an RSCS task, gets any needed work areas, and calls DMTMGJAR to issue the message to the correct destinations. Before returning control to the caller, DMTMGXEP frees any work areas it obtained.

DMTMGIAR

Issues the message according to its routing codes and subscriptions. Before a message is issued to a destination, DMTMGUAR calls DMTMGFFM to format the message.

Every 100th time a message that is part of a SETMSG subscription is not received, DMTMGIR issues message DMT616E to the command originator. Here, it calls the RMSG macro to enqueue an appropriate MSGBLOK on the message queue (DMTREXMQ) of the REX task. The REX task then calls DMTMGX to start processing the DMT616E message.

DMTMGFFM

Formats all lines of the message, according to the type of message and the language specified. It calls DMTMGSUB to process any substitutions.

DMTMGSUB

Processes individual substitutions in a message.

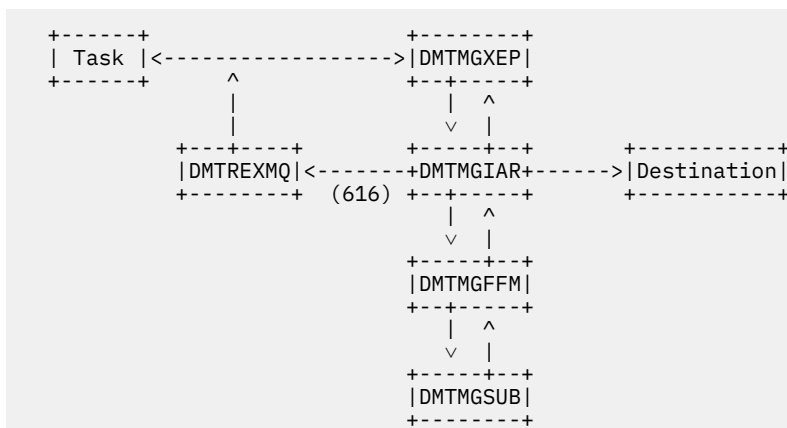


Figure 57. Overview of Message Processing

Preparing to Issue Messages

When a task calls DMTMGXEP, it points R1 to a MSGBLOK that contains all the information about the message to be issued. The information includes the message number and the repositories where the message resides. Because it is reentrant, many concurrently-executing tasks can call DMTMGXEP if each task provides its own MSGBLOK.

Before calling DMTMGIAR, DMTMGXEP ensures that a message work area (MSGWA) is available. If a MSGWA is unavailable, DMTMGXEP obtains one before calling DMTMGIAR and releases it before returning control to the calling task. Because some tasks may keep an allocated MSGWA, DMTMGXEP only allocates and deallocates a MSGWA as needed.

DMTMGXEP also attempts to find the MCNMSG entry for the requested message in the conversion repository. If this message is not in the repository, DMTMGXEP issues message DMT099E and returns control to the calling routine.

If less than 10 save areas are allocated when DMTMGXEP is called, it calls DMTQSAAB to allocate 10 additional save areas. DMTMGXEP calls DMTQSAUB to free those save areas before it returns control to the calling task.

Processing Columnar Messages

To issue the first line of a columnar message, DMTMGXEP calls DMTMGIAR twice. The first call formats the header; the second call formats the first row of information in the message body. To build the remaining lines of the message, DMTMGXEP only calls DMTMGIAR once.

Between each call to DMTMGIAR, information about the columnar message is kept in the MSGWA and an extension data area called a table display work area (TABWA). RSCS does not free the MSGWA or TABWA while the columnar message is being processed, even if DMTMGXEP called DMTQSAAB to obtain the MSGWA.

All columnar messages end with a text message. When the text message is detected, the TABWA is deallocated immediately and the MSGWA may be deallocated before DMTMGXEP returns control. Some commands that issue columnar messages do not end with a text message. Rather, these columnar messages are ended by a message containing the special message number, TBEND. When the TBEND message is specified, DMTMGXEP continues the usual end-of-table processing, but does not call DMTMGIAR to issue a message.

Unlike text messages, information required for columnar messages must be carried from one call to the next. Here, DMTMGXEP uses the same MSGBLOK supplied by the calling task to build each line of the columnar message.

Exit 27

DMTMGXEP calls Exit 27, which can inspect or change fields in the MSGBLOK before the message is issued. Exit 27 routines can also change the language in which a message is issued by specifying a translation repository and calling DMTMGXEP again. DMTMGXEP ensures, however, that Exit 27 is not called recursively. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Returning Control to the Calling Task

When DMTMGXEP returns control to the calling task, register 15 contains one of the following return code values:

0

Message issued successfully

12

Message is not defined in the conversion repository

Other

Indicates an error condition occurred when the message is delivered by a CP command (for example, 45 if the user ID is not logged on).

Issuing Messages to All Destinations

When DMTMGXEP complete its initial processing of the message request, it calls DMTMGIAR with the following entry conditions:

R7

Points to the MCNMSG information about the message in the conversion repository.

R8

Points to the MSGBLOK.

R9

Points to the MSGWA, which contains MWAFLAG flags that identify the type of message being formatted (the MSGBWA field in the MSGBLOK also points to the MSGWA).

DMTMGIAR processes this information to ensure the message is sent to the destinations specified by its routing codes and subscriptions. DMTMGIAR contains the following processing sections:

- Code that checks the message routing codes (R, C, O, and V) and subscriptions and calls an issuing routine. CRI options, such as incrementing the response counter, are also processed.

After the routing codes are determined, the routines check for any message subscriptions (SETMSG, SET *, and SET *linkid*). The MGIMONIT routine processes messages that are part of a subscription.

- Routing code routines (MGIRSS, MGICP, and MGIORIG), which process the routing codes and subscriptions.

Each of these routines calls DMTMGFFM to format the message in the appropriate language. The message language is chosen according to the following criteria:

- If a language is specified on a CRI prefix, RSCS issues the message in that language.
- If a user language is specified in the MSGBLOK, the message is issued in that language
- If Exit 28 is enabled and an exit routine issues a return code to indicate an alternate language, the specified language is used
- If no other languages are specified, RSCS issues the message in the local or network language specified on the LANGUAGE statement.

When DMTMGFFM returns the formatted message, the routing code routines call an issuing routine (described below) to issue the message to the specified destination.

- Issuing routines, which send the formatted message to its destination. These routines append EMSG headers, time stamps, and CRI headers, as appropriate, to the message before sending the message.

Two routines (MGICONS and MGILOCAL) process messages sent to the RSCS console and to users on the local node. A third routine (MGINET) sends messages to users on remote nodes. DMTCOMGN is called to identify all links to the specified node. The routine then calls DMTCOMNQ to enqueue the message on the first available, connected link to that node.

- Utility routines (MGIAPIHR and MGIMONIT) to process subscriptions and CRI information.

MGIAPIHR formats the CRI header, if relevant, for messages sent to local or remote users.

MGIMONIT issues subscription messages to the specified users. It also ensures that the message is not duplicated to the destination specified by its routing codes.

If a message that was subscribed to by a SET command is not received, RSCS cancels the subscription. It enqueues a SET ... OFF command on the command queue for the REX task. However, RSCS does not cancel subscriptions entered by the SETMSG command. Rather, it issues message DMT616E to the console every 100th time the subscribed message is not received. To do so, it invokes the RMSG macro to enqueue the message request to the REX task. In turn, the REX task calls DMTMGXEP to issue the message (see [Figure 57 on page 130](#)).

Exit 28

DMTMGIAR calls Exit 28, which can change the language in which a message is formatted. Exit 28 routines can also suppress the message. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Formatting Messages

After a DMTMGIAR routing code routine determines the message's destination, it calls DMTMGFFM.

DMTMGFFM formats the message in the language specified by the MWAFLAG field. Each line of the formatted message is placed in a MSGLINE area. These areas are chained together in the order the message is displayed. Before formatting the message, DMTMGFFM checks if the current MSGLINE chain has already been formatted. If it has, DMTMGFFM returns a pointer to DMTMGIAR and does not continue processing.

If the chain has not been formatted, DMTMGFFM calls one of six formatting routines, which process the translated and language-independent forms of each type of message:

- Text message (MGFTTEXT and MGFITEXT)
- Columnar message header (MGFTHEAD and MGFIHEAD)
- Columnar message body lines (MGFTBODY and MGFIBODY).

Message Formatting Routines

Each of the DMTMGFFM formatting routines is called with the following entry conditions:

R1

Points to the MWALTEXT field of the MSGWA, which points to the start of the chain of MSGLINE entries for the message.

R6

Points to conversion information about the message, which is mapped by the MCNMSG DSECT in the conversion repository.

R7

Points to translation information about the message, which is mapped by the MGRMSG DSECT in the translation repository.

R8

Points to the MSGBLOK.

R9

Points to the MSGWA. The MWAFLAG field indicates the language in which the message is formatted.

When formatting a text message that requires translation, the formatting routine (MGFTTEXT) allocates a MSGLINE for each line of the message text. The routine calls DMTMGSUB to process each substitution value in the message text. Each line of the formatted message is then placed in a MSGLINE.

If the language-independent form of a text message was specified, the formatting routine (MGFITEXT) only allocates one MSGLINE. Substitution values are processed, in the order they appear in the conversion repository, by individual calls to DMTMGSUB and placed in the MSGLINE.

Two DMTMGFFM routines process columnar message headers. The MGFTHEAD routine processes the message headers that require translation. This routine acquires a TABWA area to store information about the columns in the header it is processing. The routine checks the MSGBSHOW bit mask to find the columns, defined in the translation repository entry for the message, that should be displayed in the message. It then stores the starting column and the length of the body lines in the TABWA.

The second routine, MGFIHEAD, processes columnar message headers in language-independent form. Because the language-independent does not contain headers, this routine returns control to DMTMGFIAR with a null MSGLINE chain.

DMTMGFFM also contains two routines that process body lines of a columnar message. The first routine, MGFTBODY, formats a columnar message line that requires translation. This routine uses the columnar header information calculated by the MGFTHEAD routine to determine where each substitution value should be placed. The routine then calls DMTMGSUB to process each substitution.

The second routine, MGFIbody, processes the language independent form for each line in the body of a columnar message. This routine also checks the MSGBSHOW bit mask to determine which columns are needed for the message. It then calls DMTMGSUB to process any substitution values and places them in a MSGLINE.

Utility Formatting Routines

DMTMGFFM also contains several utility routines that assist in the message formatting process.

The MGFNEWLN routine allocates a MSGLINE by calling DMTQSAAB and adds it to the MSGLINE chain for the message. This MSGLINE is usually added to the end of the MSGLINE chain to ensure that the each line of the message is displayed in the correct order. For columnar message headers, however, MGFNEWLN adds the new MSGLINE to the start of the chain.

CHECKFLD determines the columns to be included in a columnar message by checking the MSGBSHOW bit mask. The MOVETEXT routine moves text into its correct location in the MSGLINE and updates the pointer to the end of the message text. If there is not enough space in the MSGLINE for the substitution text, the text is truncated.

Returning Control to DMTMGFIAR

When all lines of the message are processed, DMTMGFFM returns control to DMTMGFIAR with the following conditions:

R1

Points to a fullword field in the MSGWA, which in turns, points to the head of the MSGLINE chain. If the MSGLINE chain has already been formatted when DMTMGFFM is called, R1 points to a chain header. If the MSGLINE was not previously formatted, R1 points to the MWACCHED field.

R8

Points to the MSGBLOK

R9

Points to the MSGWA.

Processing Substitution Values

Any substitution values provided in the MSGBLOK must be formatted in the message. Each formatting routine in DMTMGF (excluding MGFIHEAD) calls DMTMGSUB with the following entry conditions:

R2

Points to the start of the available space in MSGLINE area for the message line being processed.

R3

Contains the length of the available space in the MSGLINE.

R5

Points to the contents of the message line, which is mapped by the MGRLINE DSECT in the translation repository. When processing the body of a columnar message, R5 points to a TABWA.

R6

Points to the message contents, which is mapped by the MCNMSG DSECT in the conversion repository.

R7

Points to additional translation information, which is mapped by the MGRMSG DSECT in the translation repository.

R8

Points to the MSGBLOK.

R9

Points to the MSGWA.

R11

Points, indirectly, to the information that is used for the substitution. The MGRSUB DSECT in the translation repository maps this information. For the language-independent form of a message, R11 contains the field number of the substitution.

DMTMGSUB formats one substitution and adds it to the message text accumulated in the MSGLINE. It then returns an updated MSGLINE pointer and the remaining length of the MSGLINE to DMTMGFFM.

Substitutions are processed in two ways. For translated messages, some leading text is transferred, followed by the substitution. The substitution may be padded with leading or trailing blanks or truncated according to the justification and output field width specified in the conversion repository.

For language-independent messages, the field is converted and then transferred, without justification. A three-digit prefix indicates the field length. However, dictionary terms are represented by the letter "D". They are followed by two digits that indicate the index number into a dictionary.

When converting a field, DMTMGSUB uses indirection chains to find the address and length of the data. It then calls the appropriate routine in DMTMGS to convert the data. DMTMGSUB then pads the converted data with the number of blanks specified by the output width and justification fields. Finally, DMTMGSUB appends this field to the text of the MSGLINE. If the language-independent form is specified on a CRI prefix, DMTMGSUB ignores the width and justification fields. Rather, it appends a 3-byte length prefix and the text to the MSGLINE.

Returning Control to DMTMGFFM

When a message substitution is complete, DMTMGSUB returns control to DMTMGFFM with the following conditions:

R2

Points to the next available byte in the MSGLINE.

R3

Contains the number of bytes remaining in the MSGLINE.

Message Repositories

Each message that RSCS issues is contained in a *conversion repository*. If issued in a national language, the message is also contained in a *translation repository*. Each repository file consists of fixed-length, 80-byte records, which are sequence numbered to support the CMS UPDATE facility. A compiler converts each source file into object code, which issues a message. [*z/VM: RSCS Networking Exit Customization*](#) contains more information about the structure of message repositories.

Conversion Repository

The conversion repository has the MCONV file type. Repository statements contain the following information about each message; other statements in the repository support symbolic references:

- One or more default routing codes
- If it is a private message (one that cannot be part of a SETMSG subscription)
- Conversion information for any message fields, including:
 - MSGBLOK parameter used to find the data
 - How to access the data using this parameter
 - How to convert and justify the data.

As [Figure 58 on page 135](#) shows, the MCONV exec transforms all data in the source file of the conversion repository into object form. The compiled conversion repository also contains an index based on the message numbers of all messages. DMTMGXEP uses this index to find the correct repository entry for a message number.

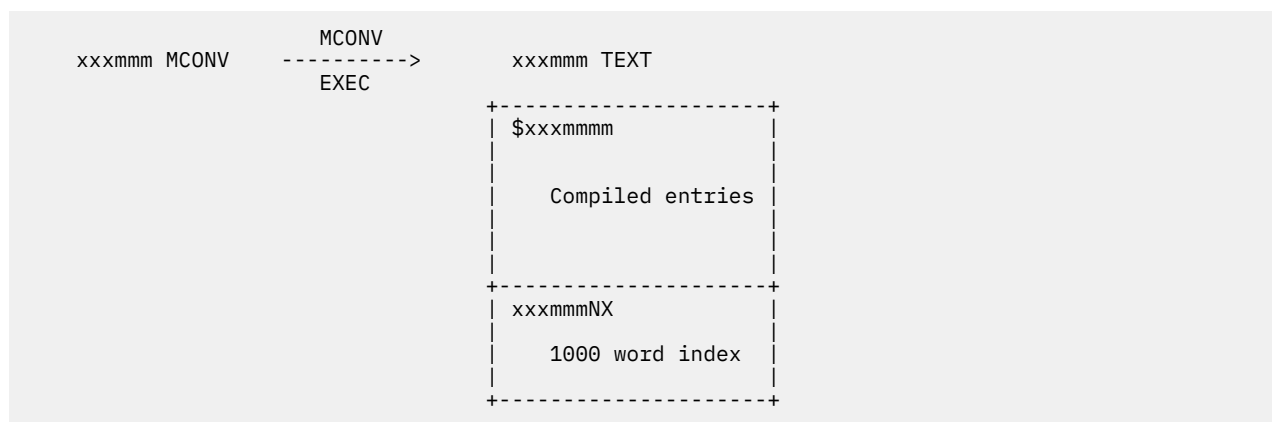


Figure 58. Compiling a Conversion Repository

Format of Compiled Repository Entries

Within the compiled repository (xxxxmm TEXT), each part of a message is mapped by DSECTs in the MSGCONV macro. The DSECTs map the format of the entire message, substitution fields, and special dictionary items.

The first 2 fullwords of MCNMSG contain general information about the message, including: its number, type (text or columnar), default routing and severity codes, header text, and the number of substitutions needed. As [Figure 59 on page 135](#) shows, MCNMSG also contains an index to the substitution fields, mapped by MCNFIELD.

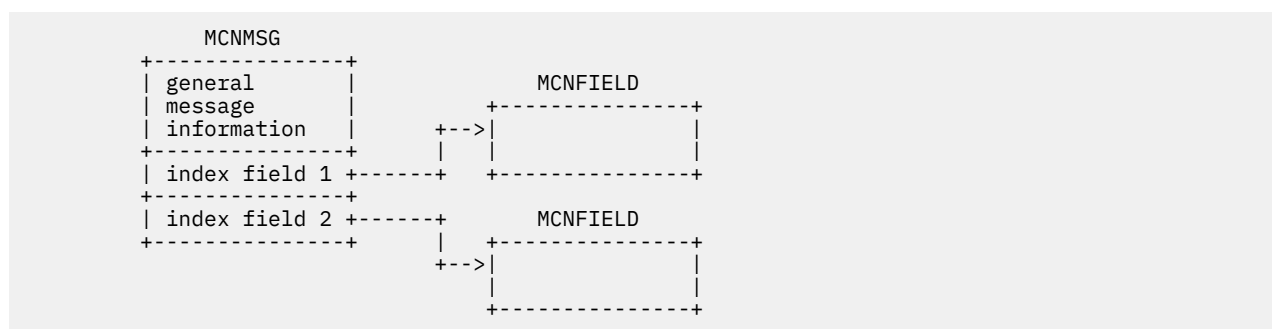


Figure 59. A Message in the Conversion Repository

MCNFIELD contains the parameter number, its input and output length, and flags that indicate the type of substitution and justification needed. Indirection locates the values for the substitution. If one indirection is needed, MCNFIELD points to the location of the information. If many indirections are needed to find

information, MCNFIELD contains a pointer to an indirection list. This list, in turn, points to the value for the substitution. Bit 0 in MCNFIELD indicates if an indirection list is to be used.

Translation Repository

The translation repository has a MSGS file type. This repository contains all message elements that can be translated, including numbered dictionary entries and TOD clock formats. For text messages, the translation repository contains fixed text and substitution values, which include a field reference, TOD, or dictionary items.

For columnar messages, the translation repository contains the following information:

- Substitution values needed in the message
- Bottom-level headings associated with the substitutions
- Higher-level groupings of headings.

Like the conversion repository, the message repository is also compiled into object form that contains the message formats and an index (see Figure 60 on page 136). The two names for the index allow translation repositories to be loaded dynamically or hard-linked into a load module.

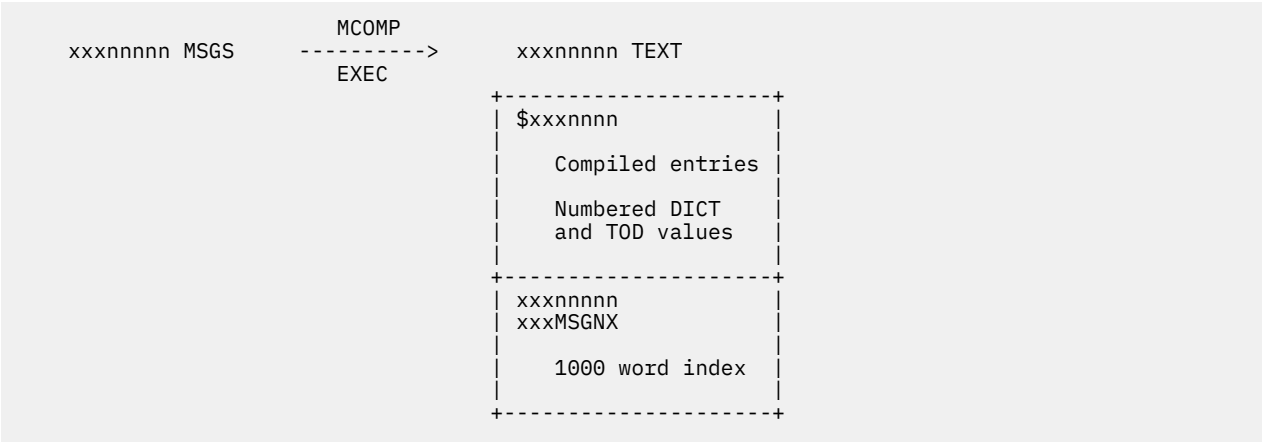


Figure 60. Compiling a Translation Repository

Format of Text Message Entries

DSECTs in the MSGTRANS macro map all of the translatable portions of the message in the compiled repository. This information includes: the format of the entire message, single lines, substitution values, and dictionary and TOD items.

The MGRMSG area contains information about the message format (text or columnar). For text messages, MGRMSG indicates the number of lines needed for the message and an index to the message lines, which are represented by MGRLINE (see Figure 61 on page 136).

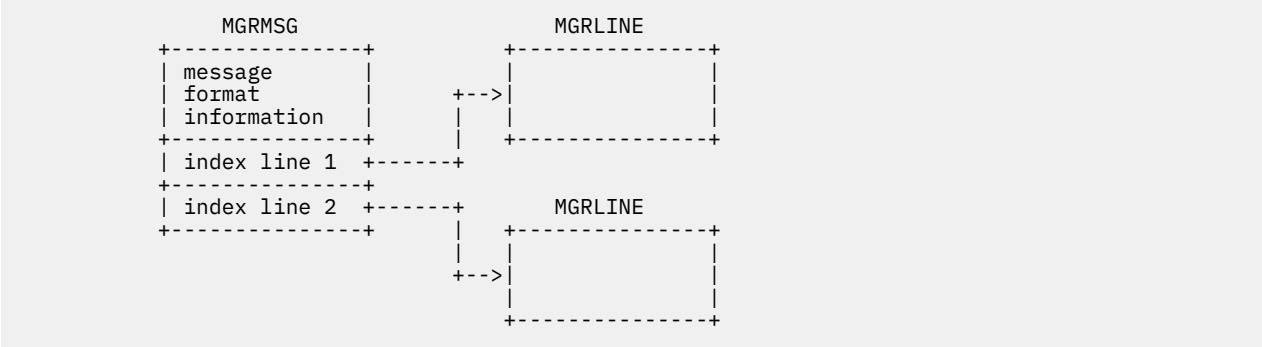


Figure 61. Structure of a Text Message

Each MGRLINE represents one line of the message. It contains the fixed message text and a pointer to an index for the substitutions. The index points to MGRSUB areas; each represents one substitution. If a dictionary item is needed for the substitution, the MGRSUB, in turn, points to a MGRDICT area. [Figure 62 on page 137](#) shows the how these areas interact to build a line of message text.

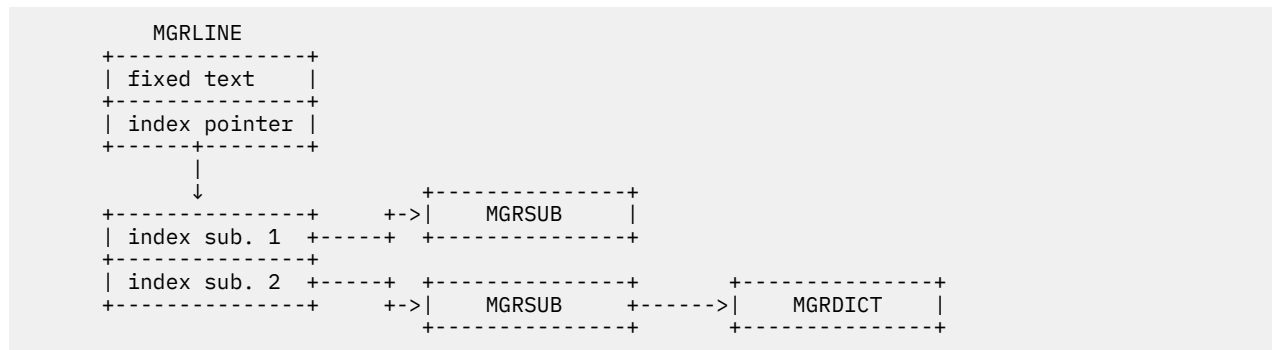


Figure 62. Building the Message Text

TOD and dictionary substitutions are encoded as pointers to literal or numbered items. If a TOD or dictionary item is numbered, its number will be greater than 1000.

Format of Columnar Message Entries

When building a columnar message, the MGRMSG area points to substitution and header indexes for each column of information to be displayed (see [Figure 63 on page 137](#)). MGRHEAD maps information about each column heading.

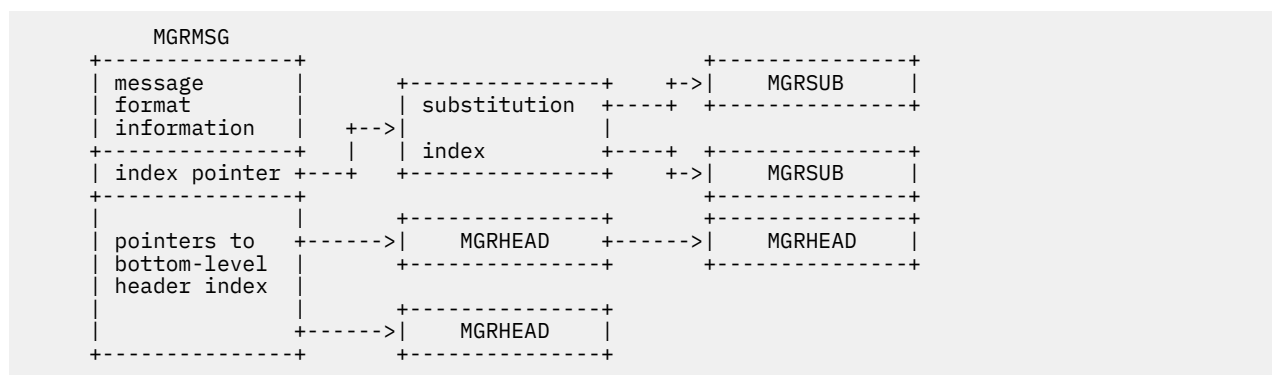


Figure 63. Building a Columnar Message

For example, message DMT696I, issued in response to the QUERY SYSTEM EXITS command, can produce the following bottom-level column headers:

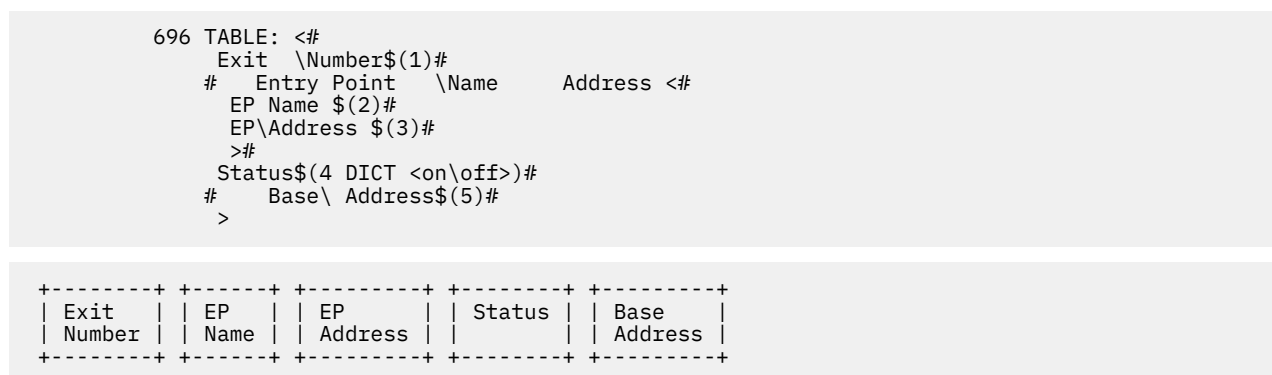


Figure 64. Bottom Level Column Headers

Message Repositories

When the entry point name and address are displayed in the response message, their column headers point to a higher level header. The columnar message response from the QUERY command produces the following display (the high-level column header is highlighted). The exit number and address values are substitution values. The “Status” column contains dictionary terms.

Exit Number	Entry Name	Point Address	Status	Base Address
11	EXITPT11	000A9D78	on	0000C1E0
29	EXITPT29	000C6DF6	off	0000C1D0
2 exits found				

Message Compilers

RSCS supplies two execs that compile the message repositories:

MCONV

Compiles conversion repositories from source files with the file type of MCONV.

MCOMP

Compiles translation repositories from source files with the file type of MSGS.

Each compiler transforms the syntax in the source file into object format. The compilers produce assembler code and invoke the system assembler to create a TEXT deck. The compilers also issue messages if processing errors occur.

You should compile the repositories if you alter the repositories supplied with RSCS. See [z/VM: RSCS Networking Exit Customization](#) for more information about the message compilers.

Each compiler contains the following processing sections:

- Lexical, listing, and error processing routines.

The lexical analysis routines open the input file, using the CMS UPDATE facility, read each token, and close the file.

The listing and error processing routines find and record any errors that may occur as the file is compiled.

- Assembler output routines, which generate assembler labels, opcodes, operands, and comment lines. The system assembler uses this information to create the processed TEXT file.
- Main compiler routines that verify the syntax of the messages in each repository. They also convert this data into the format mapped by macros in MSGCONV (for the conversion repository) and MSGTRANS (for the translation repository).

When processing the data, the compiler routines call the lexical routines to get input, the error routines to reflect errors, and assembler routines to generate the appropriate output.

Part 2. Diagnostic Aids

This part contains information to help you detect and diagnose problems that may occur in the RSCS virtual machine.

Chapter 13. Debugging Considerations

At various times, you may need to analyze a dump or trace to identify and correct a problem in RSCS. Errors may occur because of exit routines, programming errors, or hardware devices. This chapter describes some of the facilities you can use to diagnose and identify problems.

Abend Processing

This section describes facilities and techniques for gathering information from abend dumps. *z/VM: RSCS Networking Messages and Codes* describes the RSCS abend codes. *z/VM: CP Messages and Codes* contains information about GCS abend codes.

Console Abend Messages

To obtain reference material about an abend, such as messages, you should spool the RSCS console. You can then use the RSCS SCHEDULE command to close the console log at specified times, for example at midnight. For more information, see *z/VM: RSCS Networking Operation and Use*. See *z/VM: CP Commands and Utilities Reference* for information about the CP SPOOL command.

If an abend occurs, RSCS attempts to send messages to the operator's console. These messages can indicate the type of abend that occurred, the RSCS task that is involved, and the register contents at the time of the abend. Often, you can use this information to find the problem without reviewing additional dump information.

For example, if an OC5 abend occurs in the REX task, RSCS may issue the console message shown in Figure 65 on page 141.

```
16:05:40 DMTMAN010I RSCS Networking loaded at 00019000-000A4F90, CVT at 0001AFF8
COMMAND COMPLETE
16:05:44 DMTMAN090T ABEND S0C5 in supervisor task DMTREX -- task terminated
16:05:44 DMTMAN082I Program Status Word = FFE00005 C0C1C606
16:05:44 DMTMAN082I R0 - R3 = 01002000 A001B3B4 00040000 0001A7D8
16:05:44 DMTMAN082I R4 - R7 = 0001AAA8 0001AAA0 0001AAA8 0001ADC0
16:05:44 DMTMAN082I R8 - R11 = 000B8E88 0001AFF8 0001B000 0001C270
16:05:44 DMTMAN082I R12 - R15 = 0001B270 0000B058 4001B3C8 D7C1C600
16:05:44 DMTMAN082I RSCS was loaded from 00019000 to 000A4F90

DMKMSG057W BUB2      not receiving; disconnected
16:08:16
MSG FROM BUB2      : CSIABD226E Application 'RSCS' failed - System abend 0C5-0000

16:08:16 DMTMAN092T Supervisor failure -- RSCS Networking terminated
Ready;
```

Figure 65. Sample Console Abend Message

The first message, DMT010I, identifies the address at which the RSCS load module was loaded in virtual storage. This message appears in this example because RSCS had started to initialize; it may not be issued if an abend occurs in another task. The next message, DMT090T, identifies the type of abend that occurred.

The DMT082I messages that follow display the contents of the program status word (PSW) at the time of the abend. The PSW generally points to the instruction that immediately follows the last instruction that ran successfully. The DMT082I messages then display the register contents at the time of the abend, which can help you to isolate the error. The last DMT082I message contains the load address and virtual storage limit for the RSCS virtual machine. The DMT092T message displays the current status of the RSCS virtual machine.

Abend Dumps

If the console messages are not sufficient to isolate a problem, you should examine the dump that is associated with the error. Dumps are produced when GCS detects an error. They are sent to the user ID specified on the DUMP configuration file statement (see [z/VM: RSCS Networking Planning and Configuration](#)).

If you specify the VM operand on the DUMP statement, any resulting dumps will be in the VMDUMP format type, RSCSV2. You can then process the dump with the Dump Viewing Facility. See [z/VM: Dump Viewing Facility](#) for more information.

Reading Dumps

If you specify VM on the DUMP statement, you can process the RSCS dump with the Dump Viewing Facility. Chapter 14, “Examining Dumps,” on page 157 contains more information about using these facilities. For example, you can issue the DUMPSCAN DISPLAY subcommand to display various locations in the dump.

Subcommand	Function
dumpscan <i>nnnnn</i>	<i>nnnnn</i> is the problem number of the PRB <i>nnnnn</i> DUMP file.
display 298	Displays the address of the abend work area at location X'298'.
display <i>addr</i> .20	Displays the contents of the abend work area at address <i>addr</i> . This area contains the registers (displacement X'0' to X'3C') and the PSW (displacement X'40') at the time of the problem. This is the same information provided by the RSCS console dump.
display 5C4	Displays the GCS NUCON extension (SIE).
display <i>nnnn</i>	Displays the actual SIE address from the preceding operation.
display <i>xxxx</i>	Displays the address from the preceding operation plus X'14' bytes; this is the address of the first task block in dispatch queue.

At this point, you can follow the task block chain to find the task block for the abending task. You can then locate the state block for the task and determine the routine that was called when the abend occurred. See [z/VM: Group Control System](#) for information about verifying the contents of the task and state blocks.

System Abend Considerations

If a system abend occurs, the dump will contain an abend code that identifies the error. [z/VM: CP Messages and Codes](#) lists the abend codes for GCS.

RSCS also provides abend information in dumps. Register 12 is the base register for the modules within the RSCS load library. You should verify that this address is within the load range of the RSCS virtual machine. The console dump messages (see [Figure 65 on page 141](#)) contain this information.

Program Checks

Program checks always produce an abend; however, all abends are not caused by program checks. When a program check occurs, you should examine the PSW. The last byte of the first word of the PSW can identify the type of error that occurred. [z/Architecture Principles of Operation \(https://publibfp.dhe.ibm.com/epubs/pdf/a227832d.pdf\)](#) contains more information about the types of program checks that may occur. The contents of the registers at the time of the problem may also identify the cause of the program check.

Finding RSCS Data Areas

You can use the communication vector table (CVT) to locate other RSCS data areas in a dump. To find the address of the CVT, issue the QUERY SYSTEM LOADADDRESS command. As [Figure 65 on page 141](#) shows,

the command response identifies the location of the CVT. For more information about the CVT format, see [“CVT” on page 202](#).

GCS Considerations

If the RSCS dump is a VMDUMP-type dump, you can use Dump Viewing Facility commands to find information about tasks before the problem occurred. For example, you can issue the TACTIVE subcommand to display task blocks and state blocks.

However, if the dump cannot be processed or it is not in the VMDUMP format, you can still find information in the dump. The following sections describe how to find GCS information if you cannot use the TACTIVE subcommand.

Active Tasks

To find the address of the current active task for RSCS, find the address at location X'214' in the dump. After finding this address, you should then determine if the task was a parent task or a subtask.

To do so, verify the address at displacement X'88' into the active task address obtained from location X'214'. If you find a valid address, the active task is a subtask. To find its parent task, use this address from displacement X'88'. If no address is at this position, this is the parent task.

After the parent task is displayed, you can find the first state block for the task. The state block address in the task block points to the last program called by the task. It does not point to the first task that GCS gave control. The first 8 bytes of a state block contains the program name. The first state block in the chain contains the name “INIT”, which identifies the GCS program that gives control to RSCS. The second state block from the bottom of the chain will contain the DMTBPL module name.

Tracing State Blocks

State blocks are chained together; the task block points to the most active state block. A state block also contains pointers at the following displacements. By tracing the state blocks, you can follow the activity of an RSCS task.

X'10'

Points to the next state block in the chain.

X'14'

Points to the preceding state block in the chain.

The state blocks also contain the PSW (displacement X'08') and registers (displacement X'30' through X'6C'). However, these values refer to the task that is running under the preceding state block in the chain; they do not refer to this state block. The next state block, pointed to at displacement X'10', would contain this information about this state block.

Trace Data Format

If errors occur on a link, you may need to trace the transactions on the link. Errors may include lost data during transmission, protocol violations, or hardware errors. To start tracing transactions, issue the TRACE command for the link on which the problem occurs. See [z/VM: RSCS Networking Operation and Use](#) for more information about the TRACE command. Link tracing can also be turned on via the DEFINE and START commands. The following sections show examples of trace formats for several types of links.

Sample CTC Trace (RECORDS Option)

This example shows trace data from a networking channel-to-channel (CTC) link at line address 700. The link is receiving data from a remote node. The example shows the format of TRACE command when the RECORDS parameter is specified. The data from the transmission buffers is decompressed before being added to the trace. Each trace record is identified by its contents (NJE headers or data).

Debugging RSCS

RSCS LINE TRANSACTION LOG FOR LINK RSCS3 , TYPE NJE , LINE 700 ON 4/04/07 AT 9:07:57 EST

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=RECEIVE TIME= 9:07:57.202
NMR 015CA8C4 9A80 0000 0077043A D9E2C3E2 F1404040 40000041 00000000 00D9E2C3 E2F34040 4000C4D4
*....RSCS1 .....RSCS3 .DM*
link RSCS1 co* 0020 E3D5C3D9 F9F0F5C9 40E28987 95969540 96864093 89959240 D9E2C3E2 F1408396 *TNCR905I Signon of
size=4096 * 0040 94979385 A3856B40 82A48686 859940A2 89A9857E F4F0F9F6 *mplete, buffer
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:07:57.204
NMR 01201AE4 9A80 0000 0077043A D9E2C3E2 F3404040 40000041 00000000 00D9E2C3 E2F14040 4000C4D4
*....RSCS3 .....RSCS1 .DM*
link RSCS3 co* 0020 E3D5C3D9 F9F0F5C9 40E28987 95969540 96864093 89959240 D9E2C3E2 F3408396 *TNCR905I Signon of
size=4096 * 0040 94979385 A3856B40 82A48686 859940A2 89A9857E F4F0F9F6 *mplete, buffer
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.310
RIF 015D1634 9099
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=RECEIVE TIME= 9:08:32.357
PERM 015CA8C4 A099
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.360
JOBH 015D1634 99C0 0000 00CC0000 00C80000 05ADC1C1 00070101 FF000000 00000000 00000000 D9E2C3E2
*....H....AA.....RSCS*
*1453MAINT .....5.*
RSCS1 MAIN* 0020 F1F4F5F3 D4C1C9D5 E3404040 00000000 00000000 00000000 00000000 AADEF5B8
RSCS3 MAIN* 0040 F1800000 D9E2C3E2 F1404040 D4C1C9D5 E3404040 D9E2C3E2 F1404040 D4C1C9D5 *1...RSCS1 MAINT
RSCS3 MAIN* 0060 E3404040 D9E2C3E2 F3404040 D4C1C9D5 E3404040 D9E2C3E2 F3404040 D4C1C9D5 *T RSCS3 MAINT
*T .....MAIN* 0080 E3404040 00000000 00000000 00000000 00000000 00000000 00000000 D4C1C9D5
SYSPROG .....* 00A0 E3404040 40404040 40404040 40404040 E2E8E2D7 D9D6C740 00000000 00000000 *T
*.....* 00C0 00000000 00000000 0000000F
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.360
DSH 015D1634 99E0 0000 01000080 00700000 D9E2C3E2 F3404040 D4C1C9D5 E3404040 D7D9D6C6 C9D3C540 *.....RSCS3
MAINT PROFILE *
*EXEC ...A.....i...* 0020 C5E7C5C3 40404040 40404040 40404040 000000C1 0000000F 00420089 01000000
*STANDARD .....* 0040 E2E3C1D5 C4C1D9C4 40404040 40404040 00000000 00000000 00000000 00000000
*.....* 0060 00000000 00000000 A0000000 40404040 40404040 00B88700 00C14100 E2E8E2D7
EXEC .....* 0080 D9D6C740 D7D9D6C6 C9D3C540 40404040 C5E7C5C3 40404040 40404040 00320202 *ROG PROFILE
50 * 00A0 D9E2C3E2 F3404040 40D4C1C9 D5E34040 4040F5F0 40404040 40404040 40404040 *RSCS3 MAINT
* 00C0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
* 00E0 40404040 40404040 40404040 40404040 40404040 40404040 40404040
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.382
DSH 015D1634 99E0 0000 00300001 40404040 40404040 40404040 40404040 40404040 40404040
*....*
* 0020 40404040 40404040 40404040 00000000
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.382
REC 015D1634 9990 0000 028B40
*..*
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.382
REC 015D1634 9990 0000 7801C6C9 D3C57A40 D7D9D6C6 C9D3C540 40C5E7C5 C3404040 4040C1F1 40404040 *..FILE: PROFILE
EXEC A1 * 0020 40404040 40404040 40404040 E5D461C5 E2C140C3 9695A585 99A281A3 89969581 * VM/ESA
Conversations* 0040 9340D496 9589A396 9940E2A8 A2A38594 40404040 40404040 40404040 40404040 *1 Monitor
System * 0060 40404040 40D7C1C7 C540F0F0 F0F0F140 40404040 40404040 40 * PAGE
00001 *
-----
```

```
-----
TYPE TANKADDR CTLB DISP TANKDATA DIRECTION=SEND TIME= 9:08:32.382
REC 015D1634 9990 0000 021B40
-----
```

```

*..                                     *
-----
      .
      (additional data records)
      .
-----
TYPE TANKADDR CTLB DISP TANKDATA      DIRECTION=SEND      TIME= 9:08:32.397
REC 015D1634 9990 0000 0F097DC1 C3C3C5E2 E240F2F9 F140E97D      *..'ACCESS 291
Z'                                     *
-----
TYPE TANKADDR CTLB DISP TANKDATA      DIRECTION=SEND      TIME= 9:08:32.398
REC 015D1634 9990 0000 16097DE2 C5E340C6 C9D3C5D7 D6D6D340 E5D4E2E8 E27A7D      *..'SET FILEPOOL
VMSYS:'                                     *
-----
TYPE TANKADDR CTLB DISP TANKDATA      DIRECTION=SEND      TIME= 9:08:32.398
REC 015D1634 9990 0000 11097DE2 C5E340C3 D4E2E3E8 D7C540D9 E37D      *..'SET CMSTYPE
RT'                                     *
-----
TYPE TANKADDR CTLB DISP TANKDATA      DIRECTION=SEND      TIME= 9:08:32.413
JOBT 015D1634 99D0 0000 003C0000 002C0000 00C10000 00000000 00000000 00000000 00000000
*.....A.....*
*.....0020 0000000F 00000000 00000000 07070707 000C8900 00000000 00000163
*.....i.....*
-----
TYPE TANKADDR CTLB DISP TANKDATA      DIRECTION=RECEIVE     TIME= 9:08:32.631
FCMP 015CA8C4 C099
-----
TYPE TANKADDR CTLB DISP TANKDATA      DIRECTION=RECEIVE     TIME= 9:08:32.631
NMR 015CA8C4 9A80 0000 20770454 D9E2C3E2 F1404040 01D4C1C9 D5E34040 40D9E2C3 E2F34040 4000C4D4 *...RSCS1 .MAINT
RSCS3 .DM*
spooled to *
RSCS1(MAINT) 04/*
EST
*
0020 E3C1E7D4 F1F0F4C9 40C68993 85404DF1 F4F5F35D 40A29796 96938584 40A39640 *TAXM104I File (1453)
0040 D4C1C9D5 E3406060 40969989 87899540 D9E2C3E2 F14DD4C1 C9D5E35D 40F0F461 *MAINT -- origin
0060 F0F461F0 F740F0F9 7AF0F87A F3F140C5 E2E3
*04/07 09:08:31

```

Sample CTC Trace (ALL Option)

The following example shows a trace (TRACE ALL) of a networking CTC link at line address 700. This example shows how I and J sign-on records may appear in a trace.

```

-----
RSCS LINE TRANSACTION LOG FOR LINK RSCS3 , TYPE NJE , LINE 700 ON 4/04/07 AT 8:20:14 EST
-----
ADDR TYPE DISP SCSW=01884411 000A0400 90000001 SENSE=00 TIME= 8:20:14.168
C 000A03F8 CCW ---- 43200001 00000000
00000000 DATA 0000 00
*..                                     *
-----
ADDR TYPE DISP SCSW=00884007 015C9CA0 0C000000 SENSE=00 TIME= 8:20:14.172
S 015C9C98 CCW ---- 14200001 015C9D82
015C9D82 DATA 0000 07
*..                                     *
-----
ADDR TYPE DISP SCSW=00884007 015C9CB8 0C00002D SENSE=00 TIME= 8:20:14.175
W 015C9CA0 CCW ---- 01600002 000A0408
000A0408 DATA 0000 323D
*..                                     *
C 015C9CA8 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..                                     *
R 015C9CB0 CCW ---- 0220002F 015BA018
015BA018 DATA 0000 012D
*..                                     *
-----
ADDR TYPE DISP SCSW=00884007 015C9C98 0C000001 SENSE=00 TIME= 8:20:14.182
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*..                                     *
TW 015C9C60 CCW ---- 01600002 000A040A
000A040A DATA 0000 1070
*..                                     *
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..                                     *
TR 015C9C90 CCW ---- 0220002F 015BA018

```

Debugging RSCS

```

015BA018 DATA 0000 1002808F CFF0C929 D9E2C3E2 F3404040 01000000 00000010 00404040 40404040
*.....0I.RSCS3 .....*
0020 40404040 40404040 40009C00 0000
* .....*
-----
ADDR TYPE DISP SCSW=00884007 015C9C98 0C000F9D SENSE=00 TIME= 8:20:14.216
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*.....*
TW 015C9C60 CCW ---- 01600030 015DE008
015DE008 DATA 0000 1002808F CFF0D129 D9E2C3E2 F1404040 01FFFFFF FF000010 00404040 40404040
*.....0J.RSCS1 .....*
0020 40404040 40404040 40009C00 00000000
* .....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*.....*
TR 015C9C90 CCW ---- 02201002 015DA008
015DA008 DATA 0000 1002818F CF9A80FF 0077043A D9E2C3E2 F1404040 40000041 00000000 00D9E2C3
*..a.....RSCS1 .....RSC*
0020 E2F34040 4000C4D4 E3D5C3D9 F9F0F5C9 40E28987 95969540 96864093 89959240 *S3 .DMTNCR905I
Signon of link *
0040 D9E2C3E2 F14083D9 96949793 85A3856B 4082A486 86859940 A289A985 7EF4F0F9 *RSCS1 cRomplete,
buffer size=409*
0060 F6000010 26
*6....*
-----
ADDR TYPE DISP SCSW=00884007 015C9C98 0C000FFA SENSE=00 TIME= 8:20:14.240
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*.....*
TW 015C9C60 CCW ---- 01600065 015E0008
015E0008 DATA 0000 1002818F CF9A80FF 0077043A D9E2C3E2 F3404040 40000041 00000000 00D9E2C3
*..a.....RSCS3 .....RSC*
0020 E2F14040 4000C4D4 E3D5C3D9 F9F0F5C9 40E28987 95969540 96864093 89959240 *S1 .DMTNCR905I
Signon of link *
0040 D9E2C3E2 F34083D9 96949793 85A3856B 4082A486 86859940 A289A985 7EF4F0F9 *RSCS3 cRomplete,
buffer size=409*
0060 F6000010 26
*6....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*.....*
TR 015C9C90 CCW ---- 02201002 015DC008
015DC008 DATA 0000 1002828F CF001026
*..b.....*
-----
ADDR TYPE DISP SCSW=00884007 015C9C98 0C000FFA SENSE=00 TIME= 8:20:14.243
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*.....*
TW 015C9C60 CCW ---- 01600008 015C93A0
015C93A0 DATA 0000 1002828F CF001026
*..b.....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*.....*
TR 015C9C90 CCW ---- 02201002 015DC008
015DC008 DATA 0000 1002838F CF001003
*..c.....*
-----
ADDR TYPE DISP SCSW=00884007 015C9C68 0C000000 SENSE=00 TIME= 8:20:14.284
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*.....*
TW 015C9C60 CCW ---- 01200008 015C93B7
015C93B7 DATA 0000 1002838F CF001003
*..c.....*
-----
ADDR TYPE DISP SCSW=00884007 015C9C98 0C000FFA SENSE=00 TIME= 8:24:49.975
C 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*.....*
TR 015C9C90 CCW ---- 02201002 015DC008
015DC008 DATA 0000 1002848F CF001026
*..d.....*
-----
ADDR TYPE DISP SCSW=01000011 00000000 90000000 SENSE=00 TIME= 8:24:49.977
S 015C9C48 CCW ---- 14600001 015C9D82
-----
ADDR TYPE DISP SCSW=00884007 015C9CA0 0C000000 SENSE=00 TIME= 8:24:49.980
S 015C9C98 CCW ---- 14200001 015C9D82
015C9D82 DATA 0000 07
*.....*
-----

```

```

      ADDR  TYPE  DISP  SCSW=00884007 015C9C98 0C000FF7  SENSE=00  TIME= 8:24:49.985
W 015C9C60 CCW ---- 0160000B 015DE008
015DE008 DATA 0000 1002848F CF909900 001026
*..d...r....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..
TR 015C9C90 CCW ---- 02201002 015DC008
015DC008 DATA 0000 1002858F CFA09900 001026
*..e...r....*
-----
      ADDR  TYPE  DISP  SCSW=00884007 015C9C98 0C000F80  SENSE=00  TIME= 8:24:50.406
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*..
TW 015C9C60 CCW ---- 01600413 015E0008
015E0008 DATA 0000 1002858F CF99C0FF 00CC0000 00C80000 05ABC1C1 00070101 FF000000 00000000
*..e...r.....H....AA.....*
0020 00000000 D9E2C3E2 F1F4F5F1 D4C1C9D5 E3404040 00000000 00000000 00000000
*....RSCS1451MAINT
0040 00000000 AADEEBFF F374A400 00D9E2C3 E2F14040 40D4C1C9 D5E34040 40D9E2C3 *.....3.u..RSCS1
MAINT RSC*
0060 E2F14040 40D4C1C9 D5E34040 40D9E2C3 E2F34040 40D4C1C9 D5E34040 40D9E2C3 *S1 MAINT RSCS3
MAINT RSC*
0080 E2F34040 40D4C1FF C9D5E340 40400000 00000000 00000000 00000000 00000000 *S3
MA.INT .....*
00A0 00000000 0000D4C1 C9D5E340 40404040 40404040 4040E2E8 E2D7D9D6
*.....MAINT
SYSPRO*
.
. (additional data)
.
291 Z'.r.P.*
0380 405C40F2 F9F17D00 9990D00F 097DC1C3 C3C5E2E2 40F2F9F1 40E97D00 9990D716 * * 291'.r...'ACCESS
VMSYS:'.r.K..'SET*
03A0 097DE2C5 E340C6C9 D3C5D7D6 D6D340E5 D4E2E8E2 7A7D0099 90D21109 7DE2C5E3 *. 'SET FILEPOOL
03C0 40C3D4E2 E3E8D7C5 40D9E37D 0099D0FC 003C0000 002C0000 00C10000 00000000 * CMSTYPE
RT'.r.....A.....*
03E0 00000000 00000000 00000000 00000000 0000000F 00000000 00000000 07070707
*.....*
0400 000C8900 00000000 00000163 00998000 001026
*..i.....r.....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..
TR 015C9C90 CCW ---- 02201002 015DA008
015DA008 DATA 0000 1002868F CFC09900 9A80FF20 770454D9 E2C3E2F1 40404001 D4C1C9D5 E3404040
*..f...r.....RSCS1 .MAINT *
0020 D9E2C3E2 F3404040 00C4D4E3 C1E7D4F1 F0F4C940 C6899385 404DF1F4 F5F15D40 *RSCS3 .DMTAXM104I
File (1451) *
0040 A2979696 93858440 A396F340 D4C1C9D5 E3406060 40969989 87899540 D9E2C3E2 *spooled to3 MAINT --
origin RSCS*
0060 F14DD4C1 C9D5E35D 40F0F461 F0F461F0 F740F0F8 7AF2F47A F4F840C5 E2E30000 *1(MAINT) 04/04/07
08:24:48 EST...*
0080 1026
*..
*
-----
      ADDR  TYPE  DISP  SCSW=00884007 015C9C98 0C000FFA  SENSE=00  TIME= 8:24:50.500
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*..
TW 015C9C60 CCW ---- 01600008 015C93CE
015C93CE DATA 0000 1002868F CF001026
*..f.....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..
TR 015C9C90 CCW ---- 02201002 015DC008
015DC008 DATA 0000 1002878F CF001026
*..g.....*
-----
      ADDR  TYPE  DISP  SCSW=00884007 015C9C98 0C000FFA  SENSE=00  TIME= 8:24:50.571
S 015C9C48 CCW ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*..
TW 015C9C60 CCW ---- 01600008 015C93E5
015C93E5 DATA 0000 1002878F CF001003
*..g.....*
TC 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..
TR 015C9C90 CCW ---- 02201002 015DC008
015DC008 DATA 0000 1002888F CF001003
*..h.....*
-----
      ADDR  TYPE  DISP  SCSW=00884007 015C9C98 0C000FFA  SENSE=00  TIME= 8:25:00.983
C 015C9C80 CCW ---- 07600001 00000000
00000000 DATA 0000 00
*..
TR 015C9C90 CCW ---- 02201002 015DC008

```

Debugging RSCS

```
015DC008 DATA 0000 1002898F CF001026
*..i.....*
-----
      ADDR  TYPE DISP  SCSW=00884007 015C9C68 0C000000  SENSE=00  TIME= 8:25:01.000
S 015C9C48 CCW  ---- 14600001 015C9D82
015C9D82 DATA 0000 07
*..
TW 015C9C60 CCW  ---- 01200009 015C93FC
015C93FC DATA 0000 1002908F CFF0C20000
*.....0B..*
```

Sample SNANJE Trace

This example trace shows several VTAM requests on an SNANJE-type link.

```
-----
RSCS LINE TRANSACTION LOG FOR LINK RSCS2 , TYPE SNANJE , LU MBM3B ON 4/04/07 AT 8:30:27 EST
-----
      ADDR  TYPE DISP REQ TYPE=OPNDST R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.203
015C9A5C RPL 0000 00201770 800AC5B6 00000000 00000000 00001024 00800000 0006B898 00000000
*.....E.....q.....*
0020 015C9C1C 0100002E 20800000 00000000 00000000 00000000 10308450 00000000
*.*.....d&;...*
0040 80800000 40000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015C9C1C DATA --
-----
      ADDR  TYPE DISP REQ TYPE=SEND R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.211
015C9ACC RPL 0000 00202270 800ACA6E 015C9D94 00000000 00041024 80800000 0006B898 00000000
*.....>.*.m.....q.....*
0020 015C9DA8 0100002E 29800000 00000000 00000008 00000000 10309450 00000000
*.*.y.....m&;...*
0040 80800001 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008013 00000000 00000000 00000000
*.....*
015C9DA8 DATA 0000 08040400 0000C080
*.....*
-----
      ADDR  TYPE DISP REQ TYPE=RECEIVE R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.224
015C9B3C RPL 0000 00202370 800ACC68 015C9D98 00000000 00001024 00800000 0006B898 0B800000
*.....*.q.....q.....*
0020 015BF020 0100002E 29800000 00000000 00000008 0000002A 10309450 00000000 *.
$0.....m&;...*
0040 80800001 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008013 00000000 00000000 00000000
*.....*
015BF020 DATA 0000 08040400 0000C080
*.....*
-----
      ADDR  TYPE DISP REQ TYPE=SEND R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.226
015C9ACC RPL 0000 00202270 800AECA0 00000000 00000000 008A1024 80800000 0006B898 00000000
*.....*.q.....q.....*
0020 015C9DA8 0100002E 20800000 00000000 00000000 00000000 10309450 00000000
*.*.y.....m&;...*
0040 80800001 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015C9DA8 DATA --
-----
      ADDR  TYPE DISP REQ TYPE=RECEIVE R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.258
015C9B3C RPL 0000 00202370 800AD2D8 015C9D98 00000000 00001024 04800000 0006B898 03900000
*.....KQ.*.q.....q.....*
0020 015BF020 0100002E 29800000 00000000 0000002A 0000002A 10309450 00000000 *.
$0.....m&;...*
0040 80800002 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015BF020 DATA 0000 29F0C929 D9E2C3E2 F2404040 01000000 00000004 00404040 40404040 40404040
*.OI.RSCS2 .....*
0020 40404040 40001C00 0000
*.....*
-----
      ADDR  TYPE DISP REQ TYPE=RESETSR R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
```

```

8:30:27.259
015C9B3C RPL 0000 00202470 800AD498 00000000 00000000 00001024 04800000 0006B898 03900000
*.....Mq.....d.....q.....*
0020 015BF020 0100002E 20800000 00000000 0000002A 0000002A 10309450 00000000 *.
$0.....m&;...*
0040 80800002 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015BF020 DATA --
-----
ADDR TYPE DISP REQ TYPE=SEND R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.295
015C9ACC RPL 0000 00202270 800AD0B2 015C9D94 00000000 00041024 84800000 0006B898 00000000
*.....*.m.....d.....q.....*
0020 01214040 0100002E 29800000 00000000 0000002A 00000000 10309450 00000000
*..m&;...*
0040 80800002 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
01214040 DATA 0000 29F0D129 D9E2C3E2 F1404040 01FFFFFF FF000004 00404040 40404040 40404040
*.0J.RSCS1.....*
0020 40404040 40001C00 0000
*.....*
-----
ADDR TYPE DISP REQ TYPE=RECEIVE R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.326
015C9B3C RPL 0000 00202370 800ADF02 015C9D98 00000000 00001024 04800000 0006B898 03900000
*.....*.q.....d.....q.....*
0020 015CA1EA 0100002E 29800000 00000000 0000005A 00000400 90309450 00000000
*.*.....!.....m&;...*
0040 80800003 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015CA1EA DATA 0000 039A8057 09007704 3AD9E2C3 E2F18403 000041C5 003FD9E2 C3E2F240 404000C4
*.....RSCS1d....E..RSCS2..D*
0020 D4E3D5C3 D9F9F0F5 C940E289 87959695 40968640 93899592 40D9E2C3 E2F14083 *MTNCR905I Signon of
link RSCS1 c*
0040 96949793 85A3856B 4082A486 86859940 A289A985 7E04F1F0 F2F4 *complete, buffer
size=.1024 *
-----
ADDR TYPE DISP REQ TYPE=SEND R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:27.337
015C9ACC RPL 0000 00202270 800ADAB6 015C9D94 00000000 00041024 84800000 0006B898 00000000
*.....*.m.....d.....q.....*
0020 015D17EA 0100002E 29800000 00000000 0000005A 00000000 90309450 00000000
*.).....!.....m&;...*
0040 80800003 42000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015D17EA DATA 0000 039A8057 09007704 3AD9E2C3 E2F28403 000041C5 003FD9E2 C3E2F140 404000C4
*.....RSCS2d....E..RSCS1..D*
0020 D4E3D5C3 D9F9F0F5 C940E289 87959695 40968640 93899592 40D9E2C3 E2F24083 *MTNCR905I Signon of
link RSCS2 c*
0040 96949793 85A3856B 4082A486 86859940 A289A985 7E04F1F0 F2F4 *complete, buffer
size=.1024 *
-----
ADDR TYPE DISP REQ TYPE=CLSDST R15=00 R0=00 RTNCD-FDBK2=0000 SSENSEI-MI=0000 USENSEI=0000 TIME=
8:30:35.671
015C9A5C RPL 0000 00201F70 800AE0FE 00000000 00000000 00001024 00800000 0006B898 00000000
*.....q.....*
0020 015C9C1C 0100002E 20800000 00000000 00000000 00000000 10308450 00000000
*.*.....d&;...*
0040 80800000 40000000 00000000 00000000 00000000 00000000 00000000 00000000
*.....*
0060 80008012 00000000 00000000 00000000
*.....*
015C9C1C DATA --

```

Sample TCPNJE Trace

This example shows excerpts of a trace (TRACE ALL) on a TCPNJE-type link. This example includes trace data from IUCV calls for TCP/IP socket function calls (INITIALIZE, GETHOSTID, GETCLIENTID, GETHOSTBYNAME, SOCKET, SETSOCKOPT, BIND, CONNECT, SEND, and RECV).

RSCS LINE TRANSACTION LOG FOR LINK GDLVML00, TYPE TCPNJE , ON 8/07/06 AT 16:52:02 EDT

```

IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CD6 IPTRGCLS=00000000
IPBFADR1=0001A1A0 IPBFLN1F=00000014 IPSRCCLS=00000000 IPMSGTAG=00000000

```

Debugging RSCS

```
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CD6

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CD6 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=INITIALIZE DISP RETURN CODE=00000000 ERROR NUMBER=0000 CALLID=008C5CD600000000 TIME=16:52:02.660
000198AC ARG1 0000 C7C4D3E5 D4D3F0F0
*GDLVML00 *
0001672C ARG2 0000 00000031
*.... *
0001670C ARG3 0000 E3C3D7C9 D7404040
*TCPIP *
00016730 ARG4 0000 0000000A
*.... *
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5CDB IPTRGCLS=00070000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CDB

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CDB IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=GETHOSTID DISP RETURN CODE=09822848 ERROR NUMBER=0000 CALLID=008C5CDB00070000 TIME=16:52:02.664
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5CDC IPTRGCLS=001E0000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000030 RSVD=008C5CDC

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CDC IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=GETCLIENTID DISP RETURN CODE=00000000 ERROR NUMBER=0000 CALLID=008C5CDC001E0000 TIME=16:52:02.667
00016748 ARG1 0000 00000002
*.... *
000167DC ARG2 0000 00000002 C7C5C5D9 F3404040 C7C4D3E5 D4D3F0F0 40404040 40404040 40404040 *....GEER3
GDLVML00 *
0020 40404040 40404040
* *
-----
IPARML after IUCV CALL SOCKBLOK=006BFC00 SOCKCBLK=006BFB88
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=2817D6D1 IPTRGCLS=00030000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=006BFC28 IPBFLN2F=00000008 RSVD=2817D6D1

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=2817D6D1 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=GETHOSTBYNAM DISP RETURN CODE=0938D669 ERROR NUMBER=0000 CALLID=2817D6D100030000 TIME=16:52:02.670
00019960 ARG1 0000 C7C4D3E5 D4F74BC5 D5C4C9C3 D6E3E34B C9C2D44B C3D6D400
*GDLVM7.ENDICOTT.IBM.COM *
00019A60 ARG2 0000 00000018
*.... *
00019A66 ARG3 0000 384
*.. *
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=08 IPRCODE=00 IPMSGID=008C5CDD IPTRGCLS=00190000
IPBFADR1=0001A1A0 IPBFLN1F=00000010 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CDD

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CDD IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000001

FUNC=SOCKET DISP RETURN CODE=00000000 ERROR NUMBER=0000 CALLID=008C5CDD00190000 TIME=16:52:02.672
00016748 ARG1 0000 00000002
*.... *
0001674C ARG2 0000 00000001
*.... *
00016750 ARG3 0000 00000000
*.... *
00000000 ARG4
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CE5 IPTRGCLS=00170000
IPBFADR1=0001A1A0 IPBFLN1F=0000000C IPSRCCLS=00000000 IPMSGTAG=00000000
```



```
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CE5
```

```
IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CE5 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000
```

```
FUNC=SETSOCKOPT      DISP RETURN CODE=00000000 ERROR NUMBER=0000 CALLID=008C5CE500170000 TIME=16:52:02.700
00016738 ARG1 0000 00000000
```

```
*....
00016754 ARG2 0000 0000FFFF
*....
00016758 ARG3 0000 00000008
*....
000CD8CC ARG4 0000 00000001
*....
000CD8D0 ARG5 0000 00000004
*....
```

```
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CE6 IPTRGCLS=00020000
IPBFADR1=0001A1A0 IPBFLN1F=00000001 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CE6
```

```
IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CE6 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000
```

```
FUNC=BIND           DISP RETURN CODE=00000000 ERROR NUMBER=0000 CALLID=008C5CE600020000 TIME=16:52:02.703
00016738 ARG1 0000 00000000
```

```
*....
000167CC ARG2 0000 00020000 09822848 00000000 00000000
*....b.....
000CD8DC ARG3 0000 00000010
*....
```

```
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CE7 IPTRGCLS=00040000
IPBFADR1=0001A1A0 IPBFLN1F=00000001 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CE7
```

```
IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CE7 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000
```

```
FUNC=CONNECT        DISP RETURN CODE=00000000 ERROR NUMBER=0000 CALLID=008C5CE700040000 TIME=16:52:02.709
00016738 ARG1 0000 00000000
```

```
*....
000167AC ARG2 0000 000200B1 09822848 00000000 00000000
*....b.....
000CD8DC ARG3 0000 00000010
*....
```

```
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CF0 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000035 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CF0
```

```
IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CF0 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000
```

```
FUNC=SEND           DISP RETURN CODE=00000021 ERROR NUMBER=0000 CALLID=008C5CF000140000 TIME=16:52:02.715
00016738 ARG1 0000 00000000
```

```
*....
00016958 ARG2 0000 D6D7C5D5 40404040 C7C4D3C7 C5C5D940 09822848 C7C4D3E5 D4D3F0F0 09822848 *OPEN
GDLGEER .b..GDLVML00.b..*
0020 00
*
0001678C ARG3 0000 00000021
*....
000CD8C4 ARG4 0000 00000000
*....
```

```
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5CF3 IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000039 RSVD=008C5CF3
```

```
IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CF3 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=09822848
```

```
FUNC=RECV           DISP RETURN CODE=00000021 ERROR NUMBER=0000 CALLID=008C5CF300100000 TIME=16:52:02.719
00016738 ARG1 0000 00000000
```

Debugging RSCS

```
*....*
00016958 ARG2 0000 C1C3D240 40404040 C7C4D3E5 D4D3F0F0 09822848 C7C4D3C7 C5C5D940 09822848 *ACK
GDLVML00.b..GDLGEER .b..*
0020 00

*..*
00016784 ARG3 0000 00000021

*....*
000CD8C4 ARG4 0000 00000000

*....*
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CF4 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000020 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CF4

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CF4 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=SEND DISP RETURN CODE=0000000C ERROR NUMBER=0000 CALLID=008C5CF400140000 TIME=16:52:02.730
00016738 ARG1 0000 00000000

*....*
0001697C ARG2 0000 00000012 00000000 00000002

*.....*
000CD8D4 ARG3 0000 0000000C

*....*
000CD8C4 ARG4 0000 00000000

*....*
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5CF8 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000016 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5CF8

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5CF8 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=SEND DISP RETURN CODE=00000002 ERROR NUMBER=0000 CALLID=008C5CF800140000 TIME=16:52:02.735
00016738 ARG1 0000 00000000

*....*
000CD93C ARG2 0000 323D

*..*
0001678C ARG3 0000 00000002

*....*
000CD8C4 ARG4 0000 00000000

*....*
-----
.
. (Additional SEND and RECV Data)
.
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5D0E IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000024 RSVD=008C5D0E

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D0E IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=09822848

FUNC=RECV DISP RETURN CODE=0000000C ERROR NUMBER=0000 CALLID=008C5D0E00100000 TIME=16:52:02.819
00016738 ARG1 0000 00000000

*....*
00016988 ARG2 0000 0000003E 00000000 0000002E

*.....*
00016784 ARG3 0000 0000000C

*....*
000CD8C4 ARG4 0000 00000000

*....*
-----
IPARML after IUCV CALL SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5D0F IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000046 RSVD=008C5D0F

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D0F IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=09822848

FUNC=RECV DISP RETURN CODE=0000002E ERROR NUMBER=0000 CALLID=008C5D0F00100000 TIME=16:52:02.829
00016738 ARG1 0000 00000000

*....*
00017038 ARG2 0000 1002808F CFF0C929 C7C4D3E5 D4D3F0F0 01000000 00000010 00404040 40404040
```

```

*.....0I.GDLVML00.....*
          0020 40404040 40404040 40001C00 0000
*          .....*
          00016784 ARG3 0000 0000002E
*.....*
          000CD8C4 ARG4 0000 00000000
*.....*
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5D12 IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=0000001C RSVD=008C5D12

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D12 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=098C22848

FUNC=RECV          DISP RETURN CODE=00000004 ERROR NUMBER=0000 CALLID=008C5D1200100000 TIME=16:52:02.833
          00016738 ARG1 0000 00000000
*.....*
          00016990 ARG2 0000 00000000
*.....*
          00016784 ARG3 0000 00000004
*.....*
          000CD8C4 ARG4 0000 00000000
*.....*
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5D13 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000020 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5D13

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D13 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=SEND          DISP RETURN CODE=0000000C ERROR NUMBER=0000 CALLID=008C5D1300140000 TIME=16:52:02.853
          00016738 ARG1 0000 00000000
*.....*
          0001697C ARG2 0000 0000003E 00000000 0000002E
*.....*
          0001678C ARG3 0000 0000000C
*.....*
          000CD8C4 ARG4 0000 00000000
*.....*
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5D16 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000042 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5D16

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D16 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=SEND          DISP RETURN CODE=0000002E ERROR NUMBER=0000 CALLID=008C5D1600140000 TIME=16:52:02.857
          00016738 ARG1 0000 00000000
*.....*
          00698028 ARG2 0000 1002808F CFF0D129 C7C4D3C7 C5C5D940 01FFFFFF FF000010 00404040 40404040
*.....0J.GDLGEER .....*
          0020 40404040 40404040 40001C00 0000
*          .....*
          0001678C ARG3 0000 0000002E
*.....*
          000CD8C4 ARG4 0000 00000000
*.....*
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5D18 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000018 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5D18

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D18 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=SEND          DISP RETURN CODE=00000004 ERROR NUMBER=0000 CALLID=008C5D1800140000 TIME=16:52:02.862
          00016738 ARG1 0000 00000000
*.....*
          000CD8C4 ARG2 0000 00000000
*.....*
          0001678C ARG3 0000 00000004
*.....*
          000CD8C4 ARG4 0000 00000000
*.....*

```

Debugging RSCS

```
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5D21 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5D21

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D21 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=RECV      DISP RETURN CODE=0000000C ERROR NUMBER=0000 CALLID=008C5D1400100000 TIME=16:52:02.902
00016738 ARG1 0000 00000000
*....      *
00016988 ARG2 0000 0000000A 00000000 0000005A
*.....!      *
00016784 ARG3 0000 0000000C
*....      *
000CD8C4 ARG4 0000 00000000
*....      *
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5D23 IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000072 RSVD=008C5D23

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D23 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=09822848

FUNC=SEND      DISP RETURN CODE=0000000C ERROR NUMBER=0000 CALLID=008C5D2100140000 TIME=16:52:02.906
00016738 ARG1 0000 00000000
*....      *
0001697C ARG2 0000 00000076 00000000 00000066
*.....      *
0001678C ARG3 0000 0000000C
*....      *
000CD8C4 ARG4 0000 00000000
*....      *
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5D24 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000007A IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5D24

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D24 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=RECV      DISP RETURN CODE=0000005A ERROR NUMBER=0000 CALLID=008C5D2300100000 TIME=16:52:02.913
00016738 ARG1 0000 00000000
*....      *
00696008 ARG2 0000 1002818F CF9A80FF 00770431 C7C4D3C7 C5C5D940 00000041 00000000 00C7C4D3
*.a.....GDLGEER .....GDL*
0020 E5D4D3F0 F000E289 87959695 40968640 93899592 40C7C4D3 C7C5C5D9 40839694 *VML00.Signon of link
GDLGEER com*      0040 979385A3 856B40D0 82A48686 859940A2 89A9857E F4F0F9F6 0000      *plete, .buffer
size=4096..      *
00016784 ARG3 0000 0000005A
*...!      *
000CD8C4 ARG4 0000 00000000
*....      *
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5D26 IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000001C RSVD=008C5D26

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D26 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=09822848

FUNC=SEND      DISP RETURN CODE=00000066 ERROR NUMBER=0000 CALLID=008C5D2400140000 TIME=16:52:02.917
00016738 ARG1 0000 00000000
*....      *
00699030 ARG2 0000 1002818F CF9A80FF 0077043D C7C4D3E5 D4D3F0F0 40000041 00000000 00C7C4D3
*.a.....GDLVML00 .....GDL*
0020 C7C5C5D9 4000C4D4 E3D5C3D9 F9F0F5C9 40E28987 95969540 96864093 89959240 *GEER .DMTNCR905I
Signon of link *      0040 C7C4D3E5 D4D3F0D0 F0408396 94979385 A3856B40 82A48686 859940A2 89A9857E *GDLVML0.0 complete,
buffer size=*      0060 F4F0F9F6 0000
*4096..      *
0001678C ARG3 0000 00000066
*....      *
000CD8C4 ARG4 0000 00000000
```

```

*....*
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=48 IPRCODE=00 IPMSGID=008C5D30 IPTRGCLS=00140000
IPBFADR1=0001A1A0 IPBFLN1F=00000018 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000008 RSVD=008C5D30

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D30 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=RECV      DISP RETURN CODE=00000004 ERROR NUMBER=0000 CALLID=008C5D2600100000 TIME=16:52:02.947
00016738 ARG1 0000 00000000
*....*
00016990 ARG2 0000 00000000
*....*
00016784 ARG3 0000 00000004
*....*
000CD8C4 ARG4 0000 00000000
*....*
-----
IPARML after IUCV CALL      SOCKBLOK=0001A290 SOCKCBLK=0001A148
IPPATHID=0002 IPFLAGS1=88 IPRCODE=00 IPMSGID=008C5D35 IPTRGCLS=00100000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=0001A1E8 IPBFLN2F=00000024 RSVD=008C5D30

IPARML at INTERRUPT
IPPATHID=0002 IPFLAGS1=00 IPRCODE=07 IPMSGID=008C5D30 IPTRGCLS=00000000
IPBFADR1=00000000 IPBFLN1F=00000000 IPSRCCLS=00000000 IPMSGTAG=00000000
IPBFADR2=00000000 IPBFLN2F=00000000 RSVD=00000000

FUNC=SEND      DISP RETURN CODE=00000004 ERROR NUMBER=0000 CALLID=008C5D3000140000 TIME=16:52:02.952
00016738 ARG1 0000 00000000
*....*
000CD8C4 ARG2 0000 00000000
*....*
0001678C ARG3 0000 00000004
*....*
000CD8C4 ARG4 0000 00000000
*....*

```


Chapter 14. Examining Dumps

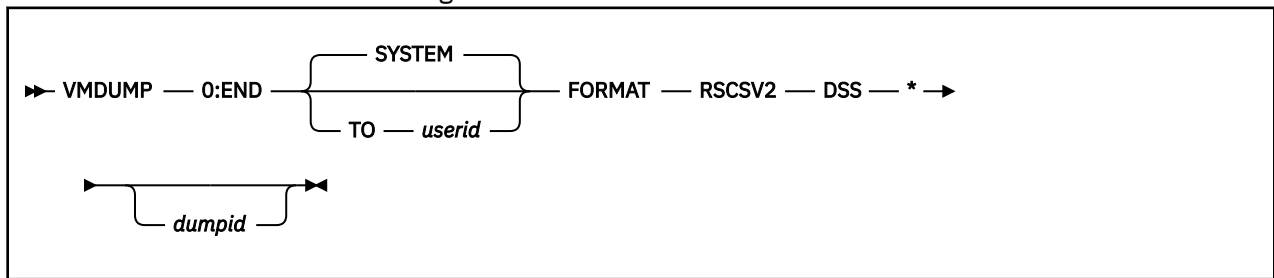
This chapter contains information to help you examine RSCS dumps. It also describes the subcommands that RSCS provides for use in DUMPSCAN sessions.

The Dump Viewing Facility is an online facility of z/VM that can help you diagnose and report system failures. You can use these facilities to interactively find and examine information from RSCS dumps.

Getting Dump Information

To process an RSCS dump with one of these facilities, the dump must be in a VMDUMP format with the type RSCSV2. Unless an enabled Exit 35 routine suppresses dumps, this dump is created automatically when an RSCS task abends.

You can also enter the VMDUMP command to create a dump. For more information on the VMDUMP command, see [z/VM: CP Commands and Utilities Reference](#). However, you only need to use this command if RSCS produces unpredictable results, but does not abend. The appropriate VMDUMP command format for RSCS is described in the following section.

**userid**

is the user ID to receive the dump.

SYSTEM

the default value, sends the dump to the user ID specified in the DUMP operand of the SYSTEM_USERIDs configuration statement.

***dumpid**

up to 100 characters that describe the dump.

Checking for a Compressed Load Map

When a dump is produced, you should check for a compressed load map. The compressed load map contains the names of the module, CSECT, and entry points. It also contains their load address; entry point addresses are sorted into ascending order.

If a compressed load map is present, you can also create various files associated with the dump (PRBnnnnnn DUMP, PRBnnnnnn REPORT, and SYMPTOM SUMMARY).

If a compressed load map is not produced, you should enter the MAP command to create this load map. You should also use this command if GCS has been updated after a compressed load map is generated. You must specify RSCSV2 as the name of the load map. For more information, see [*z/VM: Dump Viewing Facility*](#).

Using RSCS-Supplied Subcommands

You can use the DUMPSCAN command to interactively locate, display, or print data in the dump. If you do not specify any required parameters, you will be prompted for information. To save the display output from the processing session for later use, you should also enter the CP SPOOL command (see [z/VM: CP Commands and Utilities Reference](#)).

When you enter these processing environments, you can use the common subcommands for each facility and the subcommands provided by RSCS. The RSCS-supplied subcommands, described in the following sections, let you view data areas, tables, and queues that are specific to RSCS.

CVT



Purpose

The CVT subcommand displays the contents of the communications vector table.

Sample Output

Figure 66 on page 158 shows sample output from the CVT subcommand. In this example, *h* is hexadecimal data and *e* is EBCDIC data.

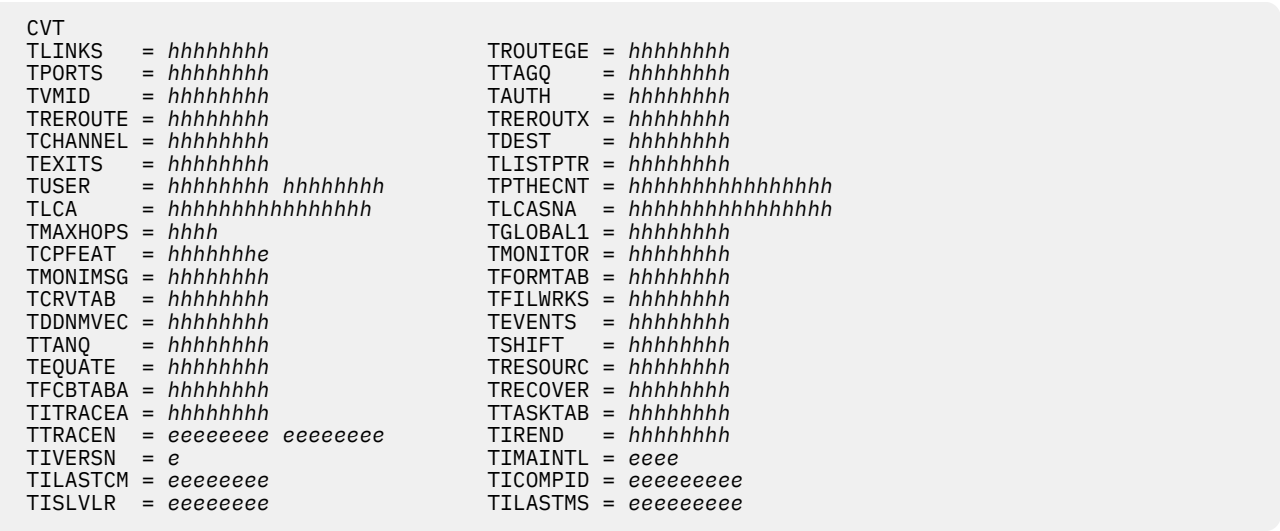


Figure 66. Output of the CVT Subcommand

Messages

- DMT985I Page 'nnnnnnnn' not found in dump

DWA



Purpose

The DWA subcommand formats and displays the dynamic work area (DWA) for any RSCS link. The DWA contains all the storage areas that are modified when the link is run.

Operands

linkid

is the 1- to 8-character link identifier of the link for which you want to display the DWA.

Sample Output

Figure 67 on page 159 shows the output from the DWA subcommand. In this example:

linkid

identifies the link

aaaaaaaa

is the address of the DWA in hexadecimal

000, 010, 020...

are displacement values

hhhhhhhh

is the DWA data in hexadecimal

eeeeee ...

is the EBCDIC representation of the DWA.

```
The DWA for link 'linkid' is:
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          .
          .
          .
The Secondary DWA or NDWA for link 'linkid' is:
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          .
          .
          .
End of DWA for link 'linkid'
```

Figure 67. Output of DWA Subcommand

Messages

- DMT982E Link 'linkid' not found
- DMT985I Page 'nnnnnnnn' not found in dump
- DMT991E Invalid command format -- unable to execute subcommand
- DMT992I No DWA for link 'linkid' available in dump
- DMT993I No secondary DWA or NDWA for link 'linkid' available in dump

IOTABLE

►► IOTABLE — linkid — rcb ◄◄

Purpose

The IOTABLE subcommand formats and displays a link's RSCS I/O table (IOTABLE) chain for a specified record control byte (RCB). The IOTABLE control block defines an I/O request to write output to line or spool.

Operands

linkid

is the 1- to 8-character link identifier of the link for which you want to display IOTABLE information.

rcb

is the hexadecimal record control byte for the specified IOTABLE chain.

Sample Output

Figure 68 on page 160 shows the output from the IOTABLE subcommand. In this example:

linkid

identifies the link

rcb

is the stream identifier of the IOTABLE being displayed

aaaaaaaa

is the address of the IOTABLE in hexadecimal

000, 010, 020...

are displacement values

hhhhhhhh

is the IOTABLE data in hexadecimal

eeeeee ...

is the EBCDIC representation of the IOTABLE.

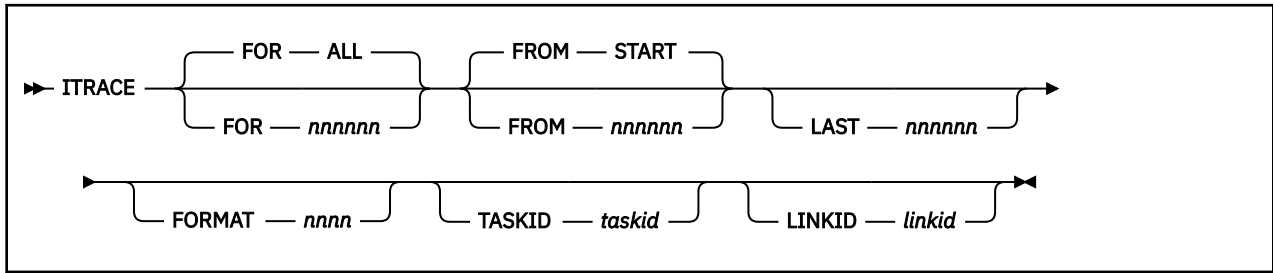
```
IOTABLE chain for link 'linkid', RCB 'rcb':
Begin IOTABLE element for link 'linkid', RCB 'rcb':
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          .
          .
          .
End IOTABLE element for link 'linkid', RCB 'rcb'
Begin IOTABLE element for link 'linkid', RCB 'rcb':
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          .
          .
          .
End IOTABLE element for link 'linkid', RCB 'rcb'
```

Figure 68. Output of the IOTABLE Subcommand

Messages

- DMT982E Link '*linkid*' not found
- DMT985I Page '*nnnnnnnn*' not found in dump
- DMT989I RCB '*rcb*' for link '*linkid*' not found in dump
- DMT991E Invalid command format -- unable to execute subcommand
- DMT992I No DWA for link '*linkid*' available in dump
- DMT993I No secondary DWA or NDWA for link '*linkid*' available in dump
- DMT994E Requested link '*linkid*' is not a networking link
- DMT995I IOTABLE for link '*linkid*', RCB '*rcb*' not found in dump

ITRACE



Purpose

The ITRACE subcommand formats and displays the RSCS internal wrap-around trace table. The contents of the trace table will vary, depending on the options you selected when you entered the RSCS ITRACE command.

Operands

FOR

indicates the number of trace entries to display from a starting point, which is specified by the FROM operand.

nnnnnn

is the number of the trace table entries to process. This value may range from 1 to 999999, depending on the value specified on the SIZE operand of the ITRACE command or configuration file statement.

ALL

(the default) displays all of the trace table entries.

FROM

displays the trace table entries starting from a specified entry number.

nnnnnn

is the number of the trace entry, which may range from 1 to 999999.

START

(the default) displays the trace table entries, beginning with the first entry in the table.

LAST **nnnnnn**

displays the last entries in the trace table. The *nnnnnn* value is the number of entries, from the last entry, that you want to display. This value may range from 1 to 262000, depending on the value specified on the SIZE operand of the ITRACE command or statement.

FORMAT **nnnn**

displays ITRACE records that were created by specifying the TYPE=*nn* operand of the ITRACE macro.

TASKID **taskid**

displays ITRACE records created by a specific task; the *taskid* is the GCS task ID number.

LINKID **linkid**

displays ITRACE records created by a specific RSCS link; the *linkid* is the 1- to 8-character identifier of the requested link.

Usage Notes

1. If you do not specify any options, ITRACE displays all of the entries in the trace table, starting with the first entry.
2. You cannot use the ITRACE subcommand on dumps created by the ITRACE TABLE DUMP YES command because RSCS low storage is not included in these types of dumps.

Sample Output

1. The following example is created by issuing the command:

```
itrace format 1a linkid rscs3
```

The ITRACE TYPE=1A call traces the TIB and TANK areas on NJE Request_Initiate_Function transmissions. The TIB control block is traced beneath the eye catcher, RIF TIB; the data buffer follows the eye catcher, RIF TANK. The first control block listed is the ITRACE header followed by the ITRACE record.

```
ITRACE for ALL      from START  format  1A taskid ALL  linkid ...RSCS3  .
0121E000 0000 AB14036A 3FCD1802 0121E040 0392CFFF .....k..
0010 0151CBE0 0151CD00 01217F00 01217E00 ....."....=.
0020 01217D00 270F0000 00000000 00000000 .....'.
0030 00000000 00000000 00000000 00000000 .....

01271080 0000 001A000A C4D4E3D5 C5E34040 D3C9D5D2 .....DMTNET LINK
0010 40D9E2C3 E2F34040 40404040 40404040 RSCS3
0020 AB1405AB 51FFE701 01204380 000005E0 .....X.....
0030 01271660 01270FA0 001A0014 0018027C .....-.....@
0040 03CDA610 00190003 03CE502D 00000000 .....w.....&;...
0050 00000000 00000000 00000000 00000000 .....
0060 00000000 00000000 00000000 00000000 .....

.
. (additional trace records)
.
0200 00000000 00000000 00000000 00000000 .....
0210 00000000 00000000 00000000 00000000 .....
0220 00180280 0220D9C9 C640E3C9 C2404040 .....RIF TIB
0230 00000000 03CE5020 03CE5030 00000000 .....&;..&;...
0240 00000000 00000000 00000000 00000000 .....
0250 00990000 00000000 0000D9E2 C3E2F240 ....r.....RSCS2
0260 404040C3 D4E2F140 40404040 F0F14040 CMS1 ... 01
0270 40404040 40404040 40404040 40404040
0280 40404040 40404040 40404040 40404040
0290 40404040 40404040 40404040 40404040
0300 40404040 40404040 40404040 40404040
0310 40404040 40404040 40404040 40404040
0320 40404040 40400000 00000000 00000000 .....
0330 00000000 00000000 00000000 00000000 .....
0340 00000000 00000000 00000000 00000000 .....
0350 00000000 00000000 00000000 00000000 .....
0360 03CCE368 01000102 00000000 00000000 .....T.....
0370 00000204 F0F9F9F1 0000049D 00000000 .....0991.....
0380 00000000 00000000 00000000 00000000 .....
0390 00000000 00000000 00000000 00000000 .....
03A0 00000000 00000000 00000000 00000000 .....
03B0 00000000 00000000 00000000 00000000 .....
03C0 00000000 00000000 00000000 00000000 .....
03D0 00000000 00000000 00000000 00000000 .....
03E0 0392E008 03CEA000 012165B0 03CEA0A8 ....k.....y
03F0 00000017 03CDA63A 03CDA63A 00880000 .....w...w..h..
0400 03000088 C8080000 D9E2C3E2 F2404040 .....hH...RSCS2
0410 40C3D4E2 F1404040 4040F0F1 40404040 CMS1 ... 01
0420 40404040 40404040 40404040 40404040
0430 40404040 40404040 40404040 40404040
0440 40404040 40404040 40404040 40404040
0450 40404040 40404040 40404040 40404040
0460 40404040 40404040 40404040 40404040
0470 40404040 40404040 40404040 40404040
0480 40404040 40404040 40404040 40404040
0490 00000000 00000000 00000000 00000000 .....
04A0 00000000 00000000 00000000 00000000 .....
04B0 00190110 04B0D9C9 C640E3C1 D5D24040 .....RIF TANK
04C0 90990000 00000000 00000000 00000000 ....r.....
04D0 00000000 00000000 00000000 00000000 .....

.
. (additional trace records)
.
05B0 00000000 00000000 00000000 00000000 .....
05C0 00000000 00000000 00000000 00000000 .....
05D0 00000000 00000000 00000000 00000000 .....
```

```
End ITRACE for ALL      from START  format  1A taskid ALL  linkid ...
RSCS3  .
```

2. The next example is produced by the following command:

```
itrace format 15 taskid 0004
```

This example shows ITRACE data from the Spool Manager task; the TYPE=15 option of the ITRACE macro was specified.

```
ITRACE for ALL      from START  format  15 taskid 0004 linkid ALL ...
0121E000 0000 AB14036A 3FCD1802 0121E040 0392CFFF .....k..
          0010 0151CBE0 0151CD00 01217F00 01217E00 .....".=
          0020 01217D00 270F0000 00000000 00000000 .....'.
          0030 00000000 00000000 00000000 00000000 .....

013ED880 0000 00150004 C4D4E3C1 E7D44040 E2E8E2E3 .....DMTAXM SYST
          0010 C5D440C1 E7D44040 40404040 40404040 EM AXM
          0020 AB140718 DA80C401 01204CE0 00000280 .....D...<....
          0030 013EDB00 013ED5E0 0015003C 00110004 .....N.....
          0040 03CCA6FC 001500A0 00068B58 001D0004 .....w.....
          0050 03CCA700 001F0004 03CCA704 00200004 .....x.....
          0060 03CCA708 00210001 03CCA70C 00220003 .....x.....x....
          0070 03CCA6F8 00000000 00000000 00000000 .....w8.....
          0080 00000000 00000000 00000000 00000000 .....
          0090 001D0010 0090D9E7 40D7C1D9 D4404040 .....RX PARM
          00A0 00068B58 00000000 00000000 00000000 .....
          00B0 00000000 00000000 00000000 00000000 .....
          00C0 00000000 00000000 00000000 00000000 .....
          00D0 001F0010 00D0D9E8 40D7C1D9 D4404040 .....RY PARM
          00E0 00000000 00000000 00000000 00000000 .....
          00F0 00200010 00F0D9E8 4EF140D7 C1D9D440 .....ORY+1 PARM
          0100 94000FFE 00000000 00000000 00000000 ...m.....
          0110 00210010 0110C4C9 C1C740C3 C3404040 .....DIAG CC
          0120 10000000 00000000 00000000 00000000 .....
          0130 00110010 0130C4C9 C1C740D9 C3404040 .....DIAG RC
          0140 94000FFE 00000000 00000000 00000000 ...m.....
          0150 00220010 0150C4C9 C1C740C3 D6C4C540 .....&DIAG CODE
          0160 140FFE00 00000000 00000000 00000000 .....
          0170 00150100 0170C4C9 C1C740D7 D3C9E2E3 .....DIAG PLIST
          0180 00000000 00891C00 D9E2C3E2 F1404040 .....i..RSCS1
          0190 C3D4E2F1 40404040 00000015 008403FC CMS1 .....d..
          01A0 00410000 00000000 E3C5E2E3 40404040 ... ..TEST
          01B0 40404040 C6C9D3C5 40404040 40404040 FILE
          01C0 F0F561F1 F661F0F7 F1F47AF0 F87AF2F7 ...05/16/0714:08:27
          01D0 00000000 0001C120 C3D4E2E3 E2E34040 .....A.CMSTST
          01E0 40404040 01220010 E2E3C1D5 C4C1D9C4 .....STANDARD
          01F0 E2E3C1D5 C4C1D9C4 00000000 00000000 ...STANDARD.....
          0200 40404040 40404040 00000000 00000000 .....
          0210 000008D8 00000000 00000000 00000000 .....Q.....
          0220 00000000 00000000 00000000 00000000 .....
          0230 00000000 00000000 00000000 00000000 .....
          0240 00000000 00000000 00000000 00000000 .....
          0250 00000000 00000000 00000000 00000000 .....
          0260 00000000 00000000 00000000 00000000 .....
          0270 00000000 00000000 00000000 00000000 .....

End ITRACE for ALL      from START  format  15 taskid 0004 linkid ...ALL .
```

3. The following example shows the output when the ITRACE table is dumped before wrapping. The ITRACE subcommand cannot be used because the RSCS load address is not included. Only commands supported by the Dump Viewing Facility can be used to view the data. Register 9 at the time of the dump contains the address of the start of the ITRACE table. The first 64 bytes is the ITRACHDR, which contains the pointer to the current entry (last used). The ITRRCBCK field points to the previous entry.

```
HCSOSS200I PROCESSING FILE RSCSITRA DUMP0030 B1      10/17/06    ... 17:03:43
HCSOSS401I READY, DUMP TYPE IS VM
DMTYDS987I RSCS LOAD ADDRESS INVALID
----> regs
REGS
CPU ADDRESS - 0000                                PREFIX REGISTER - 00000000
GENERAL REGS 0 - 15
          002BCE00 002BCF80 002B9040 002B92E0 00019300 00077003 ...00000074 01204E70
          00036710 002B9000 000195C8 000195C8 00076860 012017E0 ...800768B2 00000000
CONTROL REGS 0 - 15
          000008E2 00000000 00000000 00000000 00000000 00000000 ...FF000000 00000000
          00000002 00000000 00000000 00000000 00000000 00000000 ...D2000000 00000000
ACCESS REGS 0 - 15
          00000000 00000000 00000000 00000000 00000000 00000000 ...00000000 00000000
          00000000 00000000 00000000 00000000 00000000 00000000 ...00000000 00000000
```

```

FLOATING POINT REGS  0 - 6
00000000 00000000 00000000 00000000 00000000 00000000 ...00000000 00000000

TOD CLOCK          AAD77A48 A4759402          PSW 00EC1000 ...80076A1C
CLOCK COMPARATOR  AAD831CA 4EAFB000
CPU TIMER          FFFFFC6A EB724900

----> d 2b9000
DISPLAY 2B9000
002B9000 AAD776DF 28A95A02 002B9040 002BCFFF EE ...*.P...Z.....*
002B9010 002BCE00 002B9040 01217C00 01217B00 *......@...#.*
002B9020 01217A00 00044000 00000000 00000000 *...:.....*
002B9030 00000000 00000000 00000000 00000000 ...*.....*
002B9040 00080003 C4D4E3D9 C5E74040 E2E8E2E3 ...*....DMTREX SYST*
002B9050 C5D440D9 C5E74040 40404040 40404040 *EM REX ...*
002B9060 AAD77A42 F871A101 01204E70 000002A0 ...*.P:.8.....+....*
002B9070 002B92E0 00000000 00080014 00100030 ...*.k.....*
002B9080 012015D4 00150100 01217B00 00000000 ...*...M.....#.....*
002B9090 00000000 00000000 00000000 00000000 ...*.....*
002B90A0 00000000 00000000 00000000 00000000 ...*.....*
002B90B0 00000000 00000000 00000000 00000000 ...*.....*
002B90C0 00000000 00000000 00000000 00000000 ...*.....*
002B90D0 00100030 0090D8E2 C140C1D5 C3C84040 ...*....QSA ...ANCH *
002B90E0 00020718 0000000C 01201570 01201640 ...*.....*
002B90F0 800768B2 0001B9B4 00000100 012015B8 ...*.....*
002B9100 01217B00 00000100 00020748 00000000 ...*...#.....*
002B9110 001501C0 00D0D8E2 C140C1C4 C4D94040 ...*....QSA ...ADDR *
002B9120 00000000 00000000 00000000 00000000 ...*.....*
002B9130 00000000 00000000 00000000 00000000 ...*.....*
002B9140 00000000 00000000 00000000 00000000 ...*.....*
002B9150 00000000 00000000 00000000 00000000 ...*.....*
002B9160 00000000 00000000 00000000 00000000 ...*.....*
002B9170 00000000 00000000 00000000 00000000 ...*.....*
002B9180 00000000 00000000 00000000 00000000 ...*.....*
002B9190 00000000 00000000 00000000 00000000 ...*.....*
002B91A0 00000000 00000000 00000000 00000000 ...*.....*
002B91B0 00000000 00000000 00000000 00000000 ...*.....*
002B91C0 00000000 00000000 00000000 00000000 ...*.....*

----> d 2bce00
DISPLAY 2BCE00
002BCE00 00070003 C4D4E3D9 C5E74040 E2E8E2E3 EE ...*....DMTREX SYST*
002BCE10 C5D440D9 C5E74040 40404040 40404040 *EM REX ...*
002BCE20 AAD77A48 9A5AEC02 01204E70 00000180 ...*.P:.....+....*
002BCE30 00000000 002BCD20 0007000C 000700D0 ...*.....*
002BCE40 00036710 00000000 00000000 00000000 ...*.....*
002BCE50 00000000 00000000 00000000 00000000 ...*.....*
002BCE60 00000000 00000000 00000000 00000000 ...*.....*
002BCE70 00000000 00000000 00000000 00000000 ...*.....*
002BCE80 00000000 00000000 00000000 00000000 ...*.....*
002BCE90 000700D0 0090D4E2 C740D7C1 D9D4E240 ...*....MSG ...PARMS *
002BCEA0 C9C3D8E7 02A50000 00000000 40510007 ...*ICQX.v.....*
002BCEB0 D9E2C3E2 F1404040 40404040 40404040 *RSCS1 ...*
002BCEC0 40404040 40400000 00036800 02000000 ...*.....*
002BCED0 40404040 40404040 FD000000 00000000 ...*.....*
002BCEE0 00000000 00000000 00000000 00000000 ...*.....*
002BCEF0 01216088 40404040 40404040 40404040 ...*...-h ...*
002BCF00 40404040 40404040 40404040 40404040 ...*.....*
002BCF10 40404040 40404040 40404040 40404040 ...*.....*
002BCF20 40404040 40404040 40404040 40404040 ...*.....*
002BCF30 40404040 40404040 40404040 40404040 ...*.....*
002BCF40 40404040 40404040 40404040 40404040 ...*.....*
002BCF50 40404040 40404040 40404040 40404040 ...*.....*
002BCF60 40404040 40404040 40404040 40404040 ...*.....*
002BCF70 00000000 00000000 00000000 00000000 ...*.....*
002BCF80 00000000 00000000 00000000 00000000 ...*.....*
002BCF90 00000000 00000000 00000000 00000000 ...*.....*
002BCFA0 00000000 00000000 00000000 00000000 ...*.....*
002BCFB0 00000000 00000000 00000000 00000000 ...*.....*
002BCFC0 00000000 00000000 00000000 00000000 ...*.....*
DMTYDS987I RSCS LOAD ADDRESS INVALID

```

Messages

- DMT984E There are only 'nnnnnn' records in the ITRACE table
- DMT985I Page 'nnnnnnnn' not found in dump
- DMT990I Internal trace table is empty
- DMT991E Invalid command format -- unable to execute subcommand

LINKS



Purpose

The LINKS subcommand displays the RSCS link queue.

Operands

linkid

is the 1-to 8-character identifier for a link in the queue.

ALL

the default value, displays the characteristics of all the links within the link table.

Sample Output

Figure 69 on page 165 shows the output from the LINKS subcommand for each link in the RSCS link queue. In this example:

linkid

is the link name or the local node name

aaaaaaaa

is the address of the entry in hexadecimal

000, 010, 020...

are displacement values

hhhhhhhh

is the link entry in hexadecimal

eeeeee ...

is the EBCDIC representation of the link entry.

```

THE LINK TABLE ENTRY FOR LINK 'linkid' IS:
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          .
          .
          .
END OF LINK TABLE ENTRY FOR LINK 'linkid'
  
```

Figure 69. Output of the LINKS Subcommand

Messages

- DMT982E Link '*linkid*' not found
- DMT985I Page '*nnnnnnnn*' not found in dump

NDWA



Purpose

NDWA formats and displays the network dynamic work area (NDWA) for a specific networking link. The NDWA contains all the storage areas that a networking link driver can modify when it is called.

Operands

linkid

is the 1- to 8-character link identifier of the link for which you want to display the NDWA.

Messages

- DMT982E Link '*linkid*' not found
- DMT985I Page '*nnnnnnnn*' not found in dump
- DMT991E Invalid command format -- unable to execute subcommand
- DMT993I No secondary DWA or NDWA for link '*linkid*' available in dump
- DMT994E Requested link '*linkid*' is not a networking link

RIB



Purpose

The RIB subcommand displays the contents of receiving information blocks (RIBs). The networking link drivers use these data areas to receive data streams from remote systems.

Operands

linkid

is the 1- to 8-character link identifier of the link for which you want to display RIB information.

rcb

is the hexadecimal record control byte for the specified RIB chain.

Sample Output

Figure 70 on page 167 shows the output from the RIB subcommand. In this example:

linkid

identifies the link

rcb

is the stream identifier of the RCB displayed

aaaaaaaa

is the address of the RIB in hexadecimal

000, 010, 020...

are displacement values

hhhhhhhh

is RIB data in hexadecimal

eeeeee ...

is the EBCDIC representation of the RIB.


```
RIB for link 'linkid', RCB 'rcb':
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          .
          .
          .
End of RIB for link 'linkid', RCB 'rcb'
```

Figure 70. Output of RIB Subcommand

Messages

- DMT982E Link 'linkid' not found
- DMT985I Page 'nnnnnnnn' not found in dump
- DMT989I RCB 'rcb' for link 'linkid' not found in dump
- DMT993I No secondary DWA or NDWA for link 'linkid' available in dump
- DMT994E Requested link 'linkid' is not a networking link

ROUTES



Purpose

The ROUTES subcommand displays the contents of the RSCS routing table.

Sample Output

Route Group Name	Link ID	LINKTABL Address	Type of Route
<i>groupid</i>	<i>linkid</i>	<i>aaaaaaaa</i>	primary
.....	<i>linkid</i>	<i>aaaaaaaa</i>	primary
.....	<i>linkid</i>	<i>aaaaaaaa</i>	secondary
.....	<i>linkid</i>	<i>aaaaaaaa</i>	secondary
<i>groupid</i>	<i>linkid</i>	<i>aaaaaaaa</i>	primary
.....	<i>linkid</i>	<i>aaaaaaaa</i>	primary
.....	<i>linkid</i>	<i>aaaaaaaa</i>	secondary
.....	<i>linkid</i>	<i>aaaaaaaa</i>	secondary

Figure 71. Output of the ROUTES Subcommand

In this example:

- groupid**
identifies the routing group associated with this link entry
- linkid**
identifies the link
- aaaaaaaa**
is the address of the LINKTABL entry for this link
- primary**
is a primary link in the routing group
- secondary**
is the alternate link for the routing group.

Messages

- DMT981I ROUTEGRP table is empty

- DMT985I Page 'nnnnnnnn' not found in dump

TAGQUE



Purpose

The TAGQUE subcommand displays tag queue information for a specified link. The information is displayed in dump format.

You can display data from the input and output queues or both. If an inactive file is enqueued on a link, its tag shadow element is displayed before the actual tag element.

Operands

linkid

is the 1- to 8-character link identifier of the tag queue to be displayed.

ALL

displays the input (including tag shadow elements for any inactive files) and output queues for the specified link. ALL is the default value.

Input

displays the input tag queue, including tag shadow elements, for the specified link.

Output

displays the output tag queue for the specified link.

Sample Output

Figure 72 on page 169 shows the possible output from the TAGQUE subcommand for each link in the tag queue. This example shows the information you will see when the tag queue is empty, when there are tag shadow elements, and when there are tag queue elements. The following symbols are used in these examples:

status

is the status of the link, either ACTIVE or INACTIVE

type

is the type of link, either INPUT or OUTPUT

linkid

identifies the link

aaaaaaaa

is the address of the entry in hexadecimal

000, 010, 020...

are displacement values

hhhhhhh

is the link entry in hexadecimal

eeeeee ...

is the EBCDIC representation of the link entry.

```

DMTYTG980I Link 'linkid' 'status' 'type' TAG queue is empty
TAG shadow element for 'status' 'type' TAG on link 'linkid':
aaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          020 hhhhhhhh hhhhhhhh                eeeee ...
          .
          .
          .
End of TAG shadow element for 'status' 'type' TAG on link 'linkid'
'status' 'type' TAG on link 'linkid':
aaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeee ...
          .
          .
          .
End of 'status' 'type' TAG on link 'linkid'

```

Figure 72. Output of the TAGQUE Subcommand

Messages

- DMT980I Link 'linkid' 'status' 'type' TAG queue is empty
- DMT982E Link 'linkid' not found
- DMT985I Page 'nnnnnnnn' not found in dump
- DMT986E Invalid TAGQUE parameter 'parameter' found, 'ALL' assumed

TIB



Purpose

The TIB subcommand displays the contents of transmitting information blocks (TIBs), which are data areas the networking link drivers use to transmit data streams to remote systems.

The TIB subcommand is valid only for GATEWAY-type, LISTPROC-type, NJE-type, NOTIFY-type, SNANJE-type, and TCPNJE-type links.

Operands

linkid

is the 1- to 8-character link identifier of the link for which you want to display TIB information.

rcb

is the hexadecimal record control byte for the specified TIB chain.

Sample Output

Figure 73 on page 170 shows the output from the TIB subcommand. In this example:

linkid

is the link name for which the TIBs are being displayed

rcb

is the stream identifier of the TIB being displayed

aaaaaaaa

is the address of the TIB in hexadecimal

000, 010, 020...

are displacement values

hhhhhhhh

is the TIB data in hexadecimal

eeeeee ...

is the EBCDIC representation of the TIB.

```

TIB for link 'linkid', RCB 'rcb':
aaaaaaaa 000 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          010 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          020 hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh eeeeeee ...
          .
          .
          .
End of TIB for link 'linkid', RCB 'rcb'

```

Figure 73. Output of TIB Subcommand

Messages

- DMT982E Link '*linkid*' not found
- DMT985I Page '*nnnnnnnn*' not found in dump
- DMT989I RCB '*rcb*' for link '*linkid*' not found in dump
- DMT994E Requested link '*linkid*' is not a networking link
- DMT996I TIB for link '*linkid*', RCB '*rcb*' not found in dump

Chapter 15. Solving Problems in RSCS Interchange

This chapter contains helpful information for debugging and tracing problems in RSCS Interchange.

Using REXX Traces

You can use a REXX trace for analysis of problems within the program code of the RSCS Interchange server. [z/VM: REXX/VM Reference](#) may be helpful when you want to trace server activity.

By using the RSCS Interchange DEBUG command (see [z/VM: RSCS Networking Operation and Use](#)), you can analyze the code on the server. For example, [Figure 74 on page 171](#) shows an example of the output that may be displayed when you enter the command:

```
debug command all
```

```
ACHAMA801I Requested tracing set
debug all off
2140 **      ArgumentsMixedCase=Arguments
2141 **      Upper UserCmd Arguments
2142 **      call ExitPoint Cmd, 16, CmdUser UserCmd Arguments
4464 **      ExitPoint:
4465 **      arg Name, Maxrc, Args
4466 **      if words(Exit.Name) = 0
4467 **      then
**      return 0
:
2178 **      when UserCmd="DEBUG"
**      then
2179 **      call ProcessDebug Arguments
3301 **      ProcessDebug:
3302 **      parse upper arg Area RxTrace TOKywd ToUser
3303 **      /*! Validate area */
3304 **      select
3305 **      when abbrev('ALL',Area,1)
**      then
:
4623 **      /*! Edit message using Diag X'5C' and issue
*/
4624 **      msgtext = diag(5c,msgtext)
4625 **      if ~verify('C',routcode)
4627 **      if ~verify('T',routcode)
4628 **      then
**      say msgtext
ACHAMA801I Requested tracing set
4629 **      if ~verify('A',routcode)
4637 **      if ~verify('L',routcode)
4639 **      if (~verify('U',routcode)) & (touser ~= '')
4641 **      return
2208 **      return
3380 **      return 1
2197 **      return /* end of processing the command */
```

Figure 74. Sample DEBUG Output

Using a Log File

You also can use a log file to review server activity. [Figure 75 on page 172](#) is an example of a log file.

```
yyyymmdd 11:23:52 ACHAMA000I RSCS Data Interchange Manager Function Level nnn-0000 ready
yyyymmdd 11:26:40 ACHAMA501I Temporary nickname XX000004 added for SMTP address MATT@HURRAH.TCPIP.COL.EDU
yyyymmdd 11:36:20 ACHAMA501I Temporary nickname XX000005 added for SMTP address DOUG@HURRAH.TCPIP.COL.EDU
yyyymmdd 11:42:00 ACHAMA100I RSCS Interchange Terminated
yyyymmdd 11:42:22 ACHAMA000I RSCS Data Interchange Manager Function Level nnn-0000 ready
yyyymmdd 12:09:16 ACHAMA148I Sent file to ABC(JONES) from SMTP user matt@hurrah.tcpip.col.edu
yyyymmdd 12:10:06 ACHAMA005I Location *(SMTP) executing: MSG ABC I1 * UNABLE TO DELIVER MAIL TO: <U2@XYZ>
yyyymmdd 12:11:03 ACHAMA147I Sent file 7162 from ABC(JONES) through SMTP to MATT@HURRAH.TCPIP.COL.EDU
yyyymmdd 14:19:25 ACHAMA005I Location *(DOUG) executing: Q NICK ALL
yyyymmdd 14:22:26 ACHAMA005I Location *(DOUG) executing: NICK ADD MARK USER2ABC.TCPIP.COL.EDU
yyyymmdd 14:23:39 ACHAMA005I Location *(DOUG) executing: Q NICK ALL
yyyymmdd 14:24:26 ACHAMA005I Location *(DOUG) executing: CHANGE MARK USER2@ABC.TCPIP.COL.EDU
yyyymmdd 14:24:42 ACHAMA005I Location *(DOUG) executing: Q NICK ALL
yyyymmdd 14:29:24 ACHAMA005I Location *(DOUG) executing: NICK ADD PAUL USER1@ABC.TCPIP.COL.EDU
yyyymmdd 14:43:49 ACHAMA147I Sent file 7289 from XYZ(DOUG) through SMTP to USER1@ABC.TCPIP.COL.EDU
yyyymmdd 14:48:47 ACHAMA147I Sent file 7298 from ABC(DRB) through SMTP to USER1@ABC.TCPIP.COL.EDU
yyyymmdd 14:50:50 ACHAMA005I Location *(DOUG) executing: Q NICK ALL
yyyymmdd 14:51:54 ACHAMA005I Location *(DOUG) executing: NICK ADD DRB DRB@ABC.TCPIP.COL.EDU
yyyymmdd 15:06:30 ACHAMA005I Location *(DOUG) executing: Q NICK ALL
yyyymmdd 15:07:57 ACHAMA147I Sent file 7311 from ABC(DRB) through SMTP to DRB@ABC.TCPIP.COL.EDU
yyyymmdd 15:11:39 ACHAMA005I Location *(DOUG) executing: Q SYSTEM
yyyymmdd 15:17:36 ACHAMA005I Location *(DOUG) executing: Q NICK ALL
yyyymmdd 15:21:39 ACHAMA147I Sent file 7327 from ABC(JONES) through SMTP to JONES@ABC.TCPIP.COL.EDU
yyyymmdd 15:51:55 ACHAMA147I Sent file 7358 from ABC(DRB) through SMTP to DRB@ABC.TCPIP.COL.EDU
yyyymmdd 15:51:59 ACHAMA005I Location *(SMTP) executing: MSG ABC I2 * MAIL DELIVERED TO: <U1@TCPIP.COL.EDU>
yyyymmdd 15:55:19 ACHAMA147I Sent file 7365 from ABC(DRB) through SMTP to DOUG@HURRAH.TCPIP.COL.EDU
yyyymmdd 16:00:28 ACHAMA108E Invalid destination address on file 7367
yyyymmdd 16:04:18 ACHAMA147I Sent file 7370 from ABC(USER1) through SMTP to DOUG@HURRAH.TCPIP.COL.EDU
yyyymmdd 18:02:28 ACHAMA147I Sent file 7416 from ABC(USER1) through SMTP to DRB@ABC.TCPIP.COL.EDU
yyyymmdd 18:02:34 ACHAMA147I Sent file 7418 from ABC(RSCS) through SMTP to MATT@HURRAH.TCPIP.COL.EDU
yyyymmdd 18:02:41 ACHAMA147I Sent file 7417 from ABC(RSCS) through SMTP to MATT@HURRAH.TCPIP.COL.EDU
```

Figure 75. Sample RSCS Interchange Server Log File

Using Incoming and Outgoing Mail Files

If a mail file is handled correctly, the original incoming mail file, MAIL MAIL A, is read in by the Interchange server and converted to MAIL NOTE A for the outgoing note. These files exist on the disk, until the next file arrives, and can be used for problem determination.

Using RSCS Diagnosis Commands

Any problems with RSCS Interchange code residing in the RSCS machine should be diagnosed with RSCS diagnosis aids.

Part 3. Reference Directories

This part contains directories to the RSCS modules and important data areas. Use this section to identify the structure and contents of the data areas described in [Part 1, “Functional Overview,” on page 1](#) and [Part 2, “Diagnostic Aids,” on page 139](#).

Chapter 16. Module Directory

This chapter contains reference information about the RSCS modules and entry points. The modules are listed alphabetically; however, their executable entry points are listed in the order they appear in the module.

RSCS Modules

The following tables describe the modules that are part of the RSCS load library.

DMTAPT

DMTAPT, a printer link driver task, contains routines that are used by ASCII printer and plotter exit routines to communicate with various ASCII output devices. DMTAPT and the exit routines, which are supplied by the installation, create ASCII-type links. See [“ASCII Printer and Plotter Link Driver” on page 82](#) and [z/VM: RSCS Networking Exit Customization](#) for more information.

Entry Point	Attribute	Description
DMTAPTEP	Reentrant	Provides the interface between ASCII printer and plotter exit routines and RSCS.

DMTAST

DMTAST, the auto-start task, manages all BSC auto-start links on all auto-dial ports defined to RSCS. It also manages the inactivity timeout and RETRY functions for all links that specify the ITO or RETRY parameters, respectively. See [“Auto-Start Task” on page 52](#) for more information.

Entry Point	Attribute	Description
DMTASTEP	Serially reusable	Main entry point of the auto-start task.
DMTASTTX	Serially reusable	Tells the auto-start task when it should check for activity on an auto-start link; also called the STIMER interrupt exit.

DMTAXA

DMTAXA contains the transmission algorithms supplied by IBM; for compatibility, it also contains the initial code for additional internal transmission algorithms. The spool manager task calls a transmission algorithm when a networking link requests to process a file. See [z/VM: RSCS Networking Exit Customization](#) for more information.

Entry Point	Attribute	Description
DMTAXAG0	Serially reusable	Contains transmission algorithm 0.
DMTAXAG1	Serially reusable	Contains transmission algorithm 1.
DMTAXAG2- DMTAXAGF	Serially reusable	Contains entry code for optional transmission algorithms 2 through F.

DMTAXM

DMTAXM, the spool manager task, controls the interface between RSCS link driver tasks and the spool system. DMTAXM also enqueues files for transmission on links, processes spool file-related commands, and calls transmission algorithms. See [“Spool Manager Task” on page 46](#) for more information.

Entry Point	Attribute	Description
DMTAXMEP	Serially reusable	Initializes the spool manager task and monitors ECBs to determine when files and commands need to be processed.
DMTAXMGE	Reentrant	Notifies the spool manager task when RSCS receives an I/O interrupt from GCS because a file has arrived in RSCS's virtual reader.
DMTAXMRQ	Reentrant	Processes requests from link driver tasks to open or close an input or output file.
DMTAXMSE	Serially reusable	Traps specification exceptions when the AXM task determines if the system on which RSCS is running supports certain features (for example, Diagnose codes).

DMTAXM calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 2	Serially reusable	Creates an accounting record for each spool file RSCS receives from a local user.
Exit 3	Serially reusable	Creates or modifies an accounting record for each file RSCS sends on a link.
Exit 4	Serially reusable	Creates an accounting record for each file RSCS purges.
Exit 5	Serially reusable	Creates or modifies an accounting record for each file RSCS receives on a link.
Exit 6	Serially reusable	Ensures a file's TAG priority is used correctly.
Exit 21	Serially reusable	Establishes criteria to accept or reject an incoming spool file.
Exit 24	Serially reusable	Examines or modifies a CP command executed by the spool manager task.
Exit 25	Serially reusable	Examines or modifies the return code from a CP command executed by the spool manager task.
Exit 31	Serially reusable	Changes the sort priority of a file's TASHADOW elements.
Exit 34	Serially reusable	Determines if the spool manager task should execute or modify a command element.

DMTBOX

DMTBOX contains the data to print the VM-style separator page; it does not contain any executable entry points.

DMTBPL

DMTBPL contains routines, which are called by many RSCS tasks and user exit routines, to attach and load links, transmission algorithms, and exit routines. If errors occur in this process, the task or exit routine that called DMTBPL will not abend.

Entry Point	Attribute	Description
DMTBPLEP	Reentrant	Calls DMTBPLLX to load the link driver or exit routine code; it then transfers control to the link driver task or exit routine.
DMTBPLAL	Reentrant	Attaches a stub task at DMTBPLEP, which transfers control to the link driver task or exit routine.
DMTBPLLX	Reentrant	Loads the specified link driver task or exit routine.

DMTCMA

DMTCMA works with DMTCMX to process commands. Each entry point runs a specific RSCS command. See [z/VM: RSCS Networking Operation and Use](#) for information about each command.

Entry Point	Attribute	Description
DMTCMAPO	Serially reusable	Processes the PORT command.
DMTCMAXT	Serially reusable	Processes the EXIT command.
DMTCMARO	Serially reusable	Processes the ROUTE command.
DMTCMASL	Serially reusable	Processes the SLOWDOWN command.

DMTCMB

DMTCMB works with DMTCMX to process RSCS commands, which are executed by the following entry points. See [z/VM: RSCS Networking Operation and Use](#) for information about each command.

Entry Point	Attribute	Description
DMTCMBSC	Serially reusable	Executes the SCHEDULE command.
DMTCMBSH	Serially reusable	Executes the SHIFT command.
DMTCMBRE	Serially reusable	Executes the RESETCOUNTERS command.
DMTCMBDE	Serially reusable	Executes the DEST command.

DMTCMQ

DMTCMQ is the primary module to process the QUERY command. Each entry point processes an operand specified the QUERY SYSTEM command. See [z/VM: RSCS Networking Operation and Use](#) for information about the QUERY command.

Entry Point	Attribute	Description
DMTCMQEP	Reentrant	Initially processes the QUERY command and calls the appropriate entry point to run the specific command.
DMTCMQLV	Serially reusable	Processes QUERY SYSTEM LEVEL.
DMTCMQLO	Serially reusable	Processes QUERY SYSTEM LOCAL.
DMTCMQLA	Serially reusable	Processes QUERY SYSTEM LOADADDRESS.
DMTCMQNE	Serially reusable	Processes QUERY SYSTEM NETWORK.
DMTCMQTC	Serially reusable	Processes QUERY SYSTEM TCPIP.
DMTCMQSQ	Serially reusable	Processes QUERY SYSTEM QUEUES.

Entry Point	Attribute	Description
DMTCMQDE	Serially reusable	Processes QUERY SYSTEM DESTS.
DMTCMQEX	Serially reusable	Processes QUERY SYSTEM EXITS.
DMTCMQSH	Serially reusable	Processes QUERY SYSTEM SHIFT.
DMTCMQSO	Serially reusable	Processes QUERY SYSTEM OPTIONS.
DMTCMQPO	Serially reusable	Processes QUERY SYSTEM PORTS.
DMTCMQRR	Serially reusable	Processes QUERY SYSTEM REROUTES.
DMTCMQSL	Serially reusable	Processes QUERY SYSTEM SLOWDOWN.
DMTCMQSZ	Serially reusable	Processes QUERY SYSTEM ZONE.

DMTCMX

DMTCMX, which operates as part of the communications task, is the primary command processor. DMTCMX receives command input strings and processes some RSCS commands at its entry points. DMTCMX calls DMTCMA, DMTCMQ, DMTCMY, and DMTCMZ to process other RSCS commands

Entry Point	Attribute	Description
DMTCMXEP	Serially reusable	Primary command processor.

DMTCMX calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 19	Serially reusable	Determines if RSCS should process a command.
Exit 29	Serially reusable	Examines or processes a command that RSCS does not recognize.

DMTCMY

DMTCMY works with DMTCMX to process RSCS commands; each entry point executes a specific command. See [z/VM: RSCS Networking Operation and Use](#) for information about RSCS commands.

Entry Point	Attribute	Description
DMTCMYBA	Serially reusable	Executes the BACKSPACE command.
DMTCMYDE	Serially reusable	Executes the DEFINE command.
DMTCMYDL	Serially reusable	Executes the DELETE command.
DMTCMYDR	Serially reusable	Executes the DRAIN command.
DMTCMYFO	Serially reusable	Executes the FORCE command.
DMTCMYFR	Serially reusable	Executes the FREE command.
DMTCMYFW	Serially reusable	Executes the FWDSPACE command.
DMTCMYHO	Serially reusable	Executes the HOLD command.
DMTCMYRE	Serially reusable	Executes the READY command.
DMTCMYSE	Serially reusable	Executes the SET and SETMSG commands.
DMTCMYST	Serially reusable	Executes the START command.

Entry Point	Attribute	Description
DMTCMYTR	Serially reusable	Executes the TRACE command.

DMTCMY calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 33	Serially reusable	Processes UPARM values on the DEFINE command.

DMTCMZ

DMTCMZ works with DMTCMX to process various RSCS commands; each entry point in the module processes a specific command. See [z/VM: RSCS Networking Operation and Use](#) for information about each command.

Entry Point	Attribute	Description
DMTCMZCP	Serially reusable	Processes the CP command and sends the associated command to the control program for processing.
DMTCMZCQ	Serially reusable	Processes the CPQ command.
DMTCMZDA	Serially reusable	Processes the DISABLE command for telecommunication ports.
DMTCMZDI	Serially reusable	Processes the DISCONNECT command to disconnect the RSCS virtual machine.
DMTCMZEN	Serially reusable	Processes the ENABLE command for telecommunication ports.
DMTCMZEX	Serially reusable	Receives the EXEC command and passes it to DMTEXE for processing.
DMTCMZIT	Serially reusable	Processes the GTRACE command.
DMTCMZLO	Serially reusable	Processes the LOOPING command.
DMTCMZRC	Serially reusable	Processes the RECONNECT command.
DMTCMZNE	Serially reusable	Processes the NETWORK command.
DMTCMZRE	Serially reusable	Processes the REORDER command.
DMTCMZRR	Serially reusable	Processes the REROUTE command.
DMTCMZSH	Serially reusable	Processes the SHUTDOWN command.
DMTCMZSO	Serially reusable	Processes the STOP command.
DMTCMZTC	Serially reusable	Processes the TCPIP command.

DMTCOM

DMTCOM contains utility routines that are used by most RSCS tasks and exit routines. For more information, see [“General Purpose Routines”](#) on page 109.

Entry Point	Attribute	Description
DMTCOMLK	Reentrant	Finds LINKTABL entries.
DMTCOMLU	Reentrant	Finds LINKTABL entries for LUNAME

Entry Point	Attribute	Description
DMTCOMLD	Reentrant	Finds LINKTABL entries for DEF LUNAME
DMTCOMLC	Reentrant	Finds LINKTABL entries for communications identifier (LCID)
DMTCOMGG	Reentrant	Finds the ROUTEGRP entries for a group name.
DMTCOMGN	Reentrant	Finds NODE and ROUTEGRP entries for a node name.
DMTCOMNQ	Reentrant	Enqueues an element from a task's general purpose queue.
DMTCOMDQ	Reentrant	Dequeues an element from a task's general purpose queue.
DMTCOMTE	Reentrant	Converts z/Architecture TOD clock values to EBCDIC.
DMTCOMTS	Reentrant	Converts EBCDIC TOD clock values into z/Architecture values.
DMTCOMDG	Reentrant	Validates and converts EBCDIC values into decimal.
DMTCOMHG	Reentrant	Validates and converts EBCDIC values into hexadecimal.
DMTCOMDV	Reentrant	Validates and converts phone numbers.
DMTCOMTG	Reentrant	Scans the origin user tag of a file.
DMTCOMLS	Reentrant	Tests for changes in the state of a link.
DMTCOMCL	Reentrant	Processes PARM values for a deactivating link.
DMTCOMGD	Reentrant	Finds a link driver entry point name using its type.
DMTCOMSM	Reentrant	Sends messages on a link.
DMTCOMFI	Reentrant	Disk file interface routine; reads records from files and returns then to the calling task.

DMTCQC

DMTCQC compiles filter programs to display specific data area information for some RSCS commands, including QUERY, CHANGE, and PURGE. DMTCQC produces a compiled subroutine, which the command processing module uses when running a command. The entry points in DMTCQC perform the specific functions to compile the filter.

Entry Point	Attribute	Description
DMTCQCIN	Reentrant	Initializes compiler.
DMTCQCIS	Reentrant	Initializes compiler for short programs.
DMTCQCEN	Reentrant	Finishes compiling with a BR R14.
DMTCQCPT	Reentrant	Loads and allocates register with pointer.
DMTCQCGE	Reentrant	Provides a generic test.
DMTCQCRR	Reentrant	Provides a single RROUTE-TO operand.
DMTCQCLC	Reentrant	Provides a link class filter.

Entry Point	Attribute	Description
DMTCQCFL	Reentrant	Tests a flag (boolean AND mode).
DMTCQCFO	Reentrant	Tests a flag (boolean OR mode).
DMTCQCFW	Reentrant	Compares against fullword.
DMTCQCHW	Reentrant	Compares against halfword.
DMTCQCFR	Reentrant	Compares register against fullword.
DMTCQCZE	Reentrant	Tests field for zero.
DMTCQCPR	Reentrant	Tests for a primary link.
DMTCQCAL	Reentrant	Tests for an alternate link.
DMTCQCCH	Reentrant	Tests for a generic in a chain of items.
DMTCQCGF	Reentrant	Tests for a group filter.

DMTCQX

DMTCQX is a logical extension of the DMTCMQ. It contains entry points that process the QUEUES and FILE operands of the QUERY command. DMTCQX also contains routines that propagate these commands.

Entry Point	Attribute	Description
DMTCQXQU	Serially reusable	Processes QUERY QUEUES.
DMTCQXFI	Serially reusable	Processes QUERY FILES.
DMTCQXLK	Serially reusable	Processes QUERY (SYSTEM) LINKS.
DMTCQXPR	Serially reusable	Performs command propagations.
DMTCQXFL	Serially reusable	Checks for flooding conditions.
DMTCQXSC	Serially reusable	Processes QUERY SYSTEM SCHEDULE.
DMTCQXCO	Serially reusable	Processes QUERY SYSTEM COUNTS.

DMTCQY

DMTCQY, a logical extension of DMTCMQ, processes the GROUP, SYSTEM, and NODE operands of the QUERY command. See [z/VM: RSCS Networking Operation and Use](#) for more information about the QUERY command.

Entry Point	Attribute	Description
DMTCQYGG	Serially reusable	Processes QUERY GROUP groupid <option>.
DMTCQYGH	Serially reusable	Processes QUERY GROUP groupid HIERARCHY.
DMTCQYSG	Serially reusable	Processes QUERY SYSTEM GROUPS.
DMTCQYNO	Serially reusable	Processes QUERY NODE nodeid <option>.
DMTCQYNH	Serially reusable	Processes QUERY NODE nodeid HIERARCHY.
DMTCQYSN	Serially reusable	Processes QUERY SYSTEM NODES.
DMTCQYSE	Serially reusable	Processes QUERY SYSTEM SET.
DMTCQYSM	Serially reusable	Processes QUERY SYSTEM SETMSG.

DMTCQZ

DMTCQZ, an extension of DMTCMQ, processes invocations of the QUERY ITRACE command. See [z/VM: RSCS Networking Operation and Use](#) for more information about the QUERY command.

Entry Point	Attribute	Description
DMTCQZIT	Serially reusable	Processes QUERY SYSTEM ITRACE.

DMTCVT

DMTCVT calls the CVT macro to build the communications vector table (CVT) when RSCS initializes. It does not contain executable entry points.

DMTDDL

DMTDDL converts data records, passed to it by the calling routine, into NETDATA format, which is used to generate a note. DMTDDL is called by DMTCNOT and can also be called by an exit routine. See [“NETDATA Conversion Routine”](#) on page 115 and [“NOTIFY Link Driver”](#) on page 101 for more information.

Entry Point	Attribute	Description
DMTDDLEP	Reentrant	Converts individual records into the NETDATA format.

DMTDUP

DMTDUP, the auto-answer (dial-up) task, monitors all dial-up ports for incoming sign-on requests from remote workstations or BSC NJE nodes. See [“Auto-Answer Tasks”](#) on page 62 for more information.

Entry Point	Attribute	Description
DMTDUPEP	Reentrant	Main entry point for the auto-answer task.

DMTDUP calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 7	Reentrant	Creates an accounting record when a sign-on time out expires for an auto-answer port.
Exit 8	Reentrant	Creates an accounting record when an auto-answer port receives unrecognizable data.
Exit 9	Reentrant	Validates the sign-on card from an auto-answer port.
Exit 10	Reentrant	Processes a sign-on card that was rejected by a link associated with an auto-answer port.

DMTEND

DMTEND identifies the end of the RSCS load module and the RSU level; it does not contain executable entry points.

DMTEQU

DMTEQU contains the RSCS equate table, which relates each link driver and system task with its task entry point address. The equate table also describes the characteristics of each RSCS task. DMTEQU does not contain executable entry points.

DMTEVE

DMTEVE, the event scheduler task, manages all events scheduled for the RSCS virtual machine. See [“Event Manager Task” on page 53](#) for more information.

Entry Point	Attribute	Description
DMTEVEEP	Serially reusable	Main entry point of the RSCS event scheduler task.

DMTEXE

DMTEXE, the exec processor task, calls the GCS command processor to run any RSCS execs.

Entry Point	Attribute	Description
DMTEXEEP	Serially reusable	Main entry point of the RSCS exec processor task.

DMTGPI

DMTGPI, the GATEWAY link driver task, contains routines that are called when a gateway program issues an RSCS gateway service macro. See [“GATEWAY Link Driver” on page 73](#) and [z/VM: RSCS Networking Exit Customization](#) for more information.

Entry Point	Attribute	Description
DMTGPIEP	Reentrant	Initializes networking data areas and prepares the call to the gateway program.
DMTGPICM	Reentrant	Processes commands for the link driver.
DMTGPIOP	Reentrant	Processes NJEOPEN requests, issued by the gateway exit routine, to open an NJE file or job.
DMTGPICL	Reentrant	Processes NJECLOSE requests, issued by the gateway exit routine, to close an NJE file or job.
DMTGPIGT	Reentrant	Processes NJEGET requests, issued by the gateway exit routine, to acquire an NJE file or job.
DMTGPIPT	Reentrant	Processes NJEPUT requests to place an NJE record into a file.
DMTGPIAB	Reentrant	Processes NJEABORT requests to stop sending an NJE record.
DMTGPIRJ	Reentrant	Processes NJERJECT requests to reject a file that has been received.
DMTGPICN	Reentrant	Processes NJECONCT requests, which tell the gateway exit routine to begin processing.
DMTGPIDS	Reentrant	Processes NJEDSCON requests, which mark the GATEWAY-type link as ACTIVE.

DMTGPI calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 37	Reentrant	Examines job headers before a store-and-forward file is sent on the link.

Exit Point	Attribute	Description
Exit 38	Reentrant	Examines dataset headers before a store-and-forward file is sent on the link.
Exit 39	Reentrant	Examines job trailers before a store-and-forward file is sent on the link.
Exit 40	Reentrant	Examines records, other than NJE headers, as RSCS receives a file on the link.

DMTHAS

DMTHAS contains routines that build and maintain hash tables, which contain information about various RSCS data structures. RSCS tasks call DMTHAS routines to find, add, or delete entries in the hash tables.

Entry Point	Attribute	Description
DMTHASHB	Reentrant	Builds the initial RSCS hash table.
DMTHASHC	Reentrant	Deallocates a hash table after it is used.
DMTHASHA	Reentrant	Adds entries to the hash table.
DMTHASHD	Reentrant	Deletes entries from the hash table.
DMTHASHF	Reentrant	Locates entries in the hash table.
DMTHASHG	Reentrant	Locates hash table entries that may contain generic values (asterisks).
DMTHASHS	Reentrant	Updates counters for the hash table, which include the number of chain anchors in use and the length of the longest chain.

DMTIOT

DMTIOT processes all RSCS I/O requests to GCS. See [“I/O Interface Routines” on page 114](#) for more information.

Entry Point	Attribute	Description
DMTIOTST	Reentrant	Performs OPEN and START processing on I/O requests.
DMTIOTHD	Reentrant	Performs HALT and CLOSE processing on I/O requests.
DMTIOTGE	Serially reusable	Contains the general I/O exit routine to GCS.

DMTITR

DMTITR copies trace records into the RSCS internal wrap around trace table. DMTITREP, the only executable entry point, is only called by expansions of the ITRACE macro.

Entry Point	Attribute	Description
DMTITREP	Reentrant	Obtains logical record formats from the trace table and formats data supplied by the calling routine into trace records.

DMTIRW

DMTIRW contains prototype AUTHBLOK, FORM, LINKTABL, PORT, and ROUTE tables, which are used by DMTIRX when RSCS initializes. It does not contain executable entry points.

DMTIRX

DMTIRX processes each statement in the configuration file when RSCS initializes. It then uses this information to build the data areas that describe the RSCS virtual machine and network structure.

Entry Point	Attribute	Description
DMTIRXEP	Serially reusable	Processes the configuration file; initializes and builds data areas tables.

DMTLAX

DMTLAX determines if a valid port address is specified on a link and if the link is available.

Entry Point	Attribute	Description
DMTLAXEP	Serially reusable	Validates the port address specified on a link and allocates if it is available.

DMTLCR

DMTLCR contains routines that are used by the list processor task, DMTLIS, and other networking link driver tasks. See [“Using List Processor Routines” on page 100](#) for more information.

Entry Point	Attribute	Description
DMTLCRNG	Reentrant	Determines if the data set header currently be processed should be included in a transmission.
DMTLCRDP	Reentrant	Scans and formats an existing data set header for later use by the list processor task.
DMTLCRTS	Reentrant	Scans the origin user tag in a file’s distribution list.
DMTLRCRD	Reentrant	Copies a data set header into an AUXTANK area.

DMTLIS

DMTLIS is the primary module of the list processor task. See [“List Processor” on page 99](#) for more information.

Entry Point	Attribute	Description
DMTLISEP	Reentrant	Main entry point for the RSCS list processor task.

DMTLOG

DMTLOG generates and processes output trace logs for RSCS link driver tasks.

Entry Point	Attribute	Description
DMTLOGEP	Reentrant	Writes trace records in to the specified output spool device.
DMTLOGCL	Reentrant	Closes the output spool device to which the trace records are written.

DMTLPD

DMTLPD is the primary module for the LPD link driver task, which receives datastreams from a TCP/IP UFT client in a TCP/IP network for distribution to a destination within the RSCS network. See [“Line Printer Daemon \(LPD\) Link Driver”](#) on page 88 for more information.

Entry Point	Attribute	Description
DMTLPDEP	Reentrant	Main entry point of the LPD link driver task.

DMTLPR

DMTLPR is the primary module for the LPR link driver task, which provides an interface between RSCS and a TCP/IP line printer daemon. See [“Line Printer Remote \(LPR\) Link Driver”](#) on page 90 for more information.

Entry Point	Attribute	Description
DMTLPREP	Reentrant	Main entry point of the LPR link driver task.

DMTMAN

DMTMAN, the console input task, accepts commands entered from the console and passes them to DMTREX for processing. DMTMAN also contains ESTAE exit routine that are called when RSCS tasks abend.

Entry Point	Attribute	Description
DMTMANEP	Serially reusable	Receives commands entered at the RSCS console or by an exec; attaches DMTREX when an INIT command is issued.
DMTMANEX	Reentrant	Common ESTAE exit routine for all RSCS tasks.
DMTMANSE	Reentrant	Contains the ESTAE exit routine for all system tasks.
DMTMANDE	Reentrant	Contains the ESTAE exit routine for all link driver tasks.
DMTMANPE	Reentrant	Contains the ESTAE exit routine for all auto-answer tasks.

DMTMAN calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 35	Serially reusable	Determines if RSCS should request a dump when a task abends.

DMTMGF

DMTMGF, part of the message builder, formats each line of a message.

Entry Point	Attribute	Description
DMTMGFFM	Reentrant	Formats each line of a message in the specified language.

DMTMGI

DMTMGI issues a message, after it is formatted by DMTMGF, to all destinations specified by its routing code.

Entry Point	Attribute	Description
DMTMGIAR	Reentrant	Sends messages to their specified destinations and subscriptions.

DMTMGI calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 28	Reentrant	Changes the language in which RSCS issues a message.

DMTMGS

DMTMGS performs any substitutions needed within a line of message text.

Entry Point	Attribute	Description
DMTMGSUB	Reentrant	Processes each substitution in message text.

DMTMGX

DMTMGX, which is called by any RSCS task, is the main message building module. It formats the message work area (MSGWA) with information about the requested message and calls DMTMGI to start the process of formatting and issuing the message. See [Chapter 12, “Message Processing,” on page 127](#) for more information about building and issuing messages.

Entry Point	Attribute	Description
DMTMGXEP	Reentrant	Prepares the MSGBLOK and MSGWA for use by DMTMGI.

DMTMGX calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 27	Reentrant	Modifies or suppresses an RSCS message.

DMTMPT

DMTMPT contains general parsing routines that are used by RSCS and user exit routines.

Entry Point	Attribute	Description
DMTMPTGP	Reentrant	Parses a token from an input string.
DMTMPTBP	Reentrant	Reads an input parameter and, if it is a keyword, branches to the appropriate processing routine.
DMTMPTCK	Reentrant	Determines if a keyword has been parsed.
DMTMPTGD	Reentrant	Converts decimal values into binary.

DMTNCR

DMTNCR is called by the networking link drivers to initialize storage and process commands and sign-on records.

Entry Point	Attribute	Description
DMTNCRIN	Reentrant	Initializes and obtains storage for networking data areas (NDWA, RIB, and TIB).
DMTNCRSG	Reentrant	Processes incoming sign-on records.
DMTNCRTC	Reentrant	Counts successful time outs on a link.
DMTNCREC	Reentrant	Counts the number of error that occur on a link.
DMTNCRTO	Reentrant	Counts the number of time outs that occur on a link.
DMTNCRCD	Reentrant	Processes commands issued for networking link drivers.

DMTNET

DMTNET is the primary module for the BSC and CTC link driver tasks. It performs initialization functions and processes I/O requests for NJE-type link driver tasks. For more information, see [“BSC and CTC Link Driver”](#) on page 71.

Entry Point	Attribute	Description
DMTNETEP	Reentrant	Main entry point to the NJE link driver tasks.

DMTNHD

DMTNHD is called by the networking link drivers to process NJE job headers when RSCS receives files, messages, and commands from a remote node. See [“Receiving NJE Headers”](#) on page 66 for more information.

Entry Point	Attribute	Description
DMTNHDMR	Reentrant	Processes incoming messages and commands.
DMTNHDMT	Reentrant	Creates and sends nodal message records to remote nodes.
DMTNHDHR	Reentrant	Assembles segments of incoming NJE headers.
DMTNHDJH	Reentrant	Processes incoming job headers and places information from the headers into TAG elements.
DMTNHDSH	Reentrant	Processes incoming data set headers.
DMTNHDJT	Reentrant	Processing incoming job trailers.

DMTNHD calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 14	Reentrant	Processes the NJE job header for an incoming file before RSCS updates the TAG element.
Exit 15	Reentrant	Processes the NJE data set header for an incoming file before RSCS updates the TAG element.

Exit Point	Attribute	Description
Exit 16	Reentrant	Processes the NJE job trailer for an incoming file before RSCS updates the TAG element.
Exit 41	Reentrant	Processes the NJE job header for an incoming file after RSCS updates the TAG element.
Exit 42	Reentrant	Processes the NJE data set header for an incoming file after RSCS updates the TAG element.
Exit 43	Reentrant	Processes the NJE job trailer for an incoming file after RSCS updates the TAG element.

DMTNHE

DMTNHE produces NJE job header records for networking links. See [“Building NJE Headers” on page 65](#) for more information.

Entry Point	Attribute	Description
DMTNHEJH	Reentrant	Builds a job header for a file to be transmitted.
DMTNHEDH	Reentrant	Builds a data set header for a file to be transmitted.
DMTNHEJT	Reentrant	Builds a job trailer for a file to be transmitted.

DMTNHE calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 11	Reentrant	Processes NJE job headers created by RSCS.
Exit 12	Reentrant	Processes NJE data set headers created by RSCS.
Exit 13	Reentrant	Processes NJE job trailers created by RSCS.

DMTNOT

DMTNOT is the primary module for the NOTIFY link driver task. It initializes the NOTIFY-type link driver task. DMTNOT also reads and initially parses the message template used to send notes to various users. See [“NOTIFY Link Driver” on page 101](#) for more information.

Entry Point	Attribute	Description
DMTNOTEP	Reentrant	Main entry point to the NOTIFY link driver task.

DMTNOT calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 22	Reentrant	Determines if the NOTIFY-type link driver issues a note to originator of a misdirected file.
Exit 23	Reentrant	Modifies the note sent to the originator of a misdirected file.
Exit 36	Reentrant	Determines if RSCS should purge a file on a NOTIFY-type link.

DMTNPT

DMTNPT is the primary module for the BSC (RJE) workstation link driver task. It performs initialization functions and performs I/O requests for the RJE link driver tasks. See [“RJE Workstation Link Driver” on page 93](#) for more information.

Entry Point	Attribute	Description
DMTNPTPEP	Reentrant	Main entry point of the RJE link driver task.

DMTNRV

DMTNRV is called by the networking link driver tasks (DMTGPI, DMTLIS, DMTNET, and DMTSNE) to empty a TP buffer from a remote node. See [“Receiving Buffers” on page 69](#) for more information.

Entry Point	Attribute	Description
DMTNRVEB	Reentrant	Empties the contents of a TP buffer received on a networking link.

DMTNRV calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 40	Reentrant	Examines records, other than NJE headers, as RSCS receives a file on the link.

DMTNTR

DMTNTR is called by the networking link drivers to fill and send data buffers. See [“Transmitting Buffers” on page 69](#) for more information.

Entry Point	Attribute	Description
DMTNTRSB	Reentrant	Fills a transmission buffer with data.

DMTNTR calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 37	Reentrant	Examines job headers before a store-and-forward file is sent on the link.
Exit 38	Reentrant	Examines dataset headers before a store-and-forward file is sent on the link.
Exit 39	Reentrant	Examines job trailers before a store-and-forward file is sent on the link.

DMTNUS

DMTNUS contains utility routines that are called by the networking link driver tasks. See [“General Purpose Routines” on page 67](#) for more information.

Entry Point	Attribute	Description
DMTNUSCP	Reentrant	Compresses spool file records into TP buffers.
DMTNUSDC	Reentrant	Decompresses records from an incoming TP buffer.

Entry Point	Attribute	Description
DMTNUSCN	Reentrant	Processes any data that can not be accepted by the spool system into segmented NOP records.
DMTNUSDN	Reentrant	Decodes segmented NOPs read from spool and prepares to send the data to a remote node.

DMTPAF

DMTPAF contains routines that parse and validate commands and configuration file statements. See [Chapter 11, “Parsing Commands and Statements,” on page 119](#) for more information.

Entry Point	Attribute	Description
DMTPAFCL	Serially reusable	Validates the command name in text string.
DMTPAFCP	Serially reusable	Parses and validates the syntax of a text string.

DMTPAR

DMTPAR verifies the syntax of parameter strings that include keyword and option information.

Entry Point	Attribute	Description
DMTPAREP	Reentrant	Validates the syntax of an input string.

DMTPCR

DMTPCR contains the RSCS networking printer common routines support for the 3270P, SNA3270P, and TN3270E link drivers.

Entry Point	Attribute	Description
DMTPCRIN	Reentrant	Initialization routine for processing the FEATURE and TRANS parameters for DMTRPT and DMTSPT.
DMTPCRDT	Reentrant	Verification routine for the optional parts of the TAG command for the 3270P and SNA3270P drivers.
DMTPCRTR	Reentrant	Translates the individual records that have been obtained from the spool file block 4K increments (SPLINKS) by DMTRDR into the printer data stream.

DMTPRD

DMTPRD, the port redirector task, handles requests from tasks to listen for connect requests on TCP ports for a specific host. See [“Port Redirector Task” on page 60](#) for more information.

Entry Point	Attribute	Description
DMTPRDEP	Reentrant	Main entry point for the RSCS TCP port redirector task.
DMTPRDDQ	Reentrant	Dequeues a message PRDBLOK for a task.
DMTPRDNQ	Reentrant	Enqueues a message PRDBLOK for a task.

DMTQSA

DMTQSA contains routines that allocate and deallocate storage buffers for reentrant RSCS tasks. The characteristics of the storage are defined by a QSABLOK. See [“Storage Management Routines” on page 113](#) for more information.

Entry Point	Attribute	Description
DMTQSAAB	Reentrant	Allocates a buffer to the calling task.
DMTQSAFA	Reentrant	Frees all buffers associated with a QSABLOK.
DMTQSAFE	Reentrant	Frees all buffers associated with all QSABLOKs.
DMTQSAUB	Reentrant	Deallocates a buffer.

DMTRDR

DMTRDR is the RSCS unit record input routine. It processes files that arrive in the RSCS virtual machine’s virtual reader and presents individual file records to the calling routine. See [“Input Spool Routines” on page 114](#) for more information.

Entry Point	Attribute	Description
DMTRDREP	Reentrant	Reads individual records from input files in RSCS’s virtual reader.
DMTRDROP	Reentrant	Initializes the input file after it has been opened and reads the first SPLINK of the file from spool.

DMTRER

DMTRER performs all functions specified on the REROUTE command and configuration file statement.

Entry Point	Attribute	Description
DMTRERIN	Serially reusable	Initializes hash tables that describe the RSCS network structure (LINKTABL, NODE, REROUTE).
DMTRERAD	Serially reusable	Adds information specified on the REROUTE command and configuration file statement to the REROUTE table.
DMTRERSC	Reentrant	Scans the REROUTE table to determine if a REROUTE request is applicable to the specified node or user ID.
DMTRERDL	Serially reusable	Deletes entries from the REROUTE table.

DMTRER calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 30	Reentrant	Establishes criteria for routing data.

DMTRES

DMTRES manages RSCS’s spool and file resources; each resource is described by a RESBLOCK. See [“DMTRES” on page 42](#) for more information.

Entry Point	Attribute	Description
DMTRESLO	Reentrant	Locks a resource for exclusive use by one task.
DMTRESUN	Reentrant	Unlock a resource that is no longer needed by a task.
DMTRESCL	Reentrant	Process RESBLOCKs for a task that has terminated.

DMTREX

DMTREX, the communications task, attaches other mandatory RSCS system tasks (spool manager, exec processor, auto-start, and event scheduler). It also monitors ECBs and notifies tasks when they must process a command or file. See [“Communications Task” on page 43](#) for more information.

Entry Point	Attribute	Description
DMTREXEP	Reentrant	Initializes and acquires storage for RSCS; it also attaches the mandatory RSCS system tasks.
DMTREXIU	Serially reusable	Tells DMTREX when an IUCV interrupt is generated when the RSCS virtual machine receives a CP SMSG command.

DMTREX calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 0	Serially reusable	Performs additional initialization processing.
Exit 1	Serially reusable	Performs additional termination processing.

DMTRGX

DMTRGX processes command and routing elements received from a remote node.

Entry Point	Attribute	Description
DMTRGXEP	Reentrant	Processes command and message elements received on networking links.

DMTRGX calls the following IBM-defined exit point; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 32	Reentrant	Processes or modifies an incoming command or message element.

DMTRPT

DMTRPT is the primary module for the 3270P link driver task. See [“3270P Printer Link Driver” on page 77](#) for more information.

Entry Point	Attribute	Description
DMTRPTEP	Reentrant	Main entry point to the RSCS 3270P link driver task.

DMTRPT calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 44	Reentrant	Perform special processing (accounting, link clean up) when a printer link terminates.
Exit 45	Reentrant	Perform accounting of output pages from a printer link.
Exit 46	Reentrant	Adjust any output page accounting on a printer link.

DMTSCT

DMTSCT, the primary module in the SNA control task, initializes and maintains the RSCS/VTAM interface. See “SNA Control Task” on page 55 for more information.

Entry Point	Attribute	Description
DMTSCTEP	Serially reusable	Main entry point for the RSCS SNA control task.
DMTSCTCU	Reentrant	Called by DMTMANEX to perform end of task processing when a session driver task is detached.

DMTSEP

DMTSEP contains routines to generate header and trailer separator pages for print files.

Entry Point	Attribute	Description
DMTSEBPL	Reentrant	Formats block letters for separator pages.
DMTSEPHD	Reentrant	Creates the header page.
DMTSEPTR	Reentrant	Creates the trailer page.

DMTSEP calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 17	Reentrant	Determines a separator page style.
Exit 18	Reentrant	Creates an alternate style separator page.

DMTSJE

DMTSJE is the primary module for the SNARJE session driver task. See “SNARJE Workstation Session Driver” on page 96 for more information.

Entry Point	Attribute	Description
DMTSJEEP	Reentrant	Main entry point for the RSCS SNARJE session driver task.
DMTSJEDF	Reentrant	Processes DFASY requests from VTAM.

DMTSLO

DMTSLO builds and maintains the vectors used for processing of the SLOWDOWN command.

Entry Point	Attribute	Description
DMTSLOBL	Serially reusable	Builds initial slowdown vectors.
DMTSLONE	Serially reusable	Adds new entries to the slowdown vector.
DMTSLORE	Serially reusable	Builds a new entry or exit vector, using a new base value.
DMTSLOFI	Reentrant	Finds a values in a slowdown vector.

DMTSML

DMTSML is the primary module for the MRJE workstation link driver task. See [“MRJE Workstation Link Driver” on page 94](#) for more information.

Entry Point	Attribute	Description
DMTSMLEP	Reentrant	Main entry point of the MRJE link driver task.

DMTSNE

DMTSNE is the primary module for the SNA networking session driver. See [“SNA LU_TO NJE Session Driver” on page 70](#) for more information.

Entry Point	Attribute	Description
DMTSNEEP	Reentrant	Main entry point for the SNANJE session driver task.
DMTSNEDF	Reentrant	Processes DFASY requests.
DMTSNERP	Reentrant	Detects positive and negatives responses to SNA requests.

DMTSOK

DMTSOK is the RSCS TCP/IP socket function interface.

Entry Point	Attribute	Description
DMTSOKET	Reentrant	Main entry point to the RSCS TCP/IP socket interface.

DMTSPT

DMTSPT is the primary module for the SNA3270P session driver. See [“SNA 3270 Printer Session Driver” on page 80](#) for more information.

Entry Point	Attribute	Description
DMTSPTEP	Reentrant	Main entry point to the SNA3270P session driver task.
DMTSPTDF	Reentrant	Processes DFASY requests.

DMTSPT calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 44	Reentrant	Perform special processing (accounting, link clean up) when a printer link terminates.
Exit 45	Reentrant	Perform accounting of output pages from a printer link.
Exit 47	Reentrant	Perform any special initialization processing needed for output page accounting on an SNA3270-type printer link.
Exit 48	Reentrant	Perform any special processing needed for if an error occurs while printing output on an SNA3270-type printer link.

DMTTAP

DMTTAP, the TCPASCII link driver task, communicates with ASCII printers and plotters that are attached to a terminal server in a TCP/IP network. See [“TCPASCII Printer and Plotter Link Driver” on page 85](#) for more information.

Entry Point	Attribute	Description
DMTTAPEP	Reentrant	Provides an interface between RSCS and terminal servers attached to TCP/IP.

DMTTAS

DMTTAS contains routines that build and maintain the RSCS task table. It is also called to find entries in the task table.

Entry Point	Attribute	Description
DMTTASKA	Reentrant	Adds entry to the RSCS task table.
DMTTASKD	Reentrant	Deletes entries from the task table.
DMTTASKF	Reentrant	Finds entries in the task table.
DMTTASKG	Reentrant	Locates entries in the task table using a task ID provided by GCS.

DMTTNE

DMTTNE, the TCPNJE link driver task, provides support that enables RSCS to communicate with a remote NJE peer system using TCP/IP as the transport mechanism. See [“TCPNJE Link Driver” on page 72](#) for more information.

Entry Point	Attribute	Description
DMTTNEEP	Reentrant	Main entry point to the TCPNJE link driver task.

DMTTPT

DMTTPT is the primary module for the TN3270E link driver task. See [“TN3270E Printer Link Driver” on page 78](#) for more information.

Entry Point	Attribute	Description
DMTTPTPEP	Reentrant	Main entry point to the RSCS TN3270E link driver task.

DMTTPT calls the following IBM-defined exit points; see [z/VM: RSCS Networking Exit Customization](#) for more information.

Exit Point	Attribute	Description
Exit 44	Reentrant	Perform special processing (accounting, link clean up) when a printer link terminates.
Exit 45	Reentrant	Perform accounting of output pages from a printer link.
Exit 46	Reentrant	Adjust any output page accounting on a printer link.

DMTUFD

DMTUFD is the primary module for the UFTD link driver task, which receives datastreams from a TCP/IP Unsolicited File Transfer (UFT) client in a TCP/IP network for distribution to a destination within the RSCS network. See [“Unsolicited File Transfer Daemon \(UFTD\) Driver” on page 105](#) for more information.

Entry Point	Attribute	Description
DMTUFDEP	Reentrant	Main entry point of the UFTD link driver task.

DMTUFT

DMTUFT is the primary module for the LPR link driver task, which sends datastreams to a TCP/IP Unsolicited File Transfer (UFT) daemon for distribution in a TCP/IP network. See [“Unsolicited File Transfer \(UFT\) Driver” on page 103](#) for more information.

Entry Point	Attribute	Description
DMTUFTEP	Reentrant	Main entry point of the UFT link driver task.

DMTURO

DMTURO builds, sends, and flushes output buffers to unit record devices. See [“Output Spool Routines” on page 115](#) for more information.

Entry Point	Attribute	Description
DMTUROEP	Reentrant	Builds CCWs and data in a buffer that is sent to the unit record device when full.
DMTUROFL	Reentrant	Flushes the buffer built by DMTUROEP.

DMTVXT

Part of the SNA control task, DMTVXT contains VTAM exit routines. VTAM calls these routines when it needs to inform RSCS about specific events or errors that affect the RSCS application or a session driver task. See [“VTAM Exit Routines” on page 57](#) for more information.

Entry Point	Attribute	Description
DMTVXTLG	Reentrant	Scheduled when a VTAM SIMLOGON request is issued because the RSCS operator started an SNA link, or because the VTAM operator issued a VTAM VARY LOGON command for a remote printer. DMTVXTLG examines the information supplied from VTAM and attempts to attach a session driver for this session.

Entry Point	Attribute	Description
DMTVXTLT	Reentrant	Scheduled when an SNA session abends or is disrupted.
DMTVXTNS	Reentrant	Scheduled when RSCS receives certain Network Services Request Units on the SSCP-LU session.
DMTVXTRL	Reentrant	Scheduled when another VTAM application issues a RELREQ request to request a logical unit that is in session with RSCS.
DMTVXTSC	Reentrant	Scheduled when RSCS receives a SCIP session control request (CLEAR, SDT, RQR, STSN, BIND, and UNBIND). If a BIND request is received, DMTVXTSC tries to start a session driver for the session (unless a session driver is already active).
DMTVXTTP	Reentrant	Scheduled when VTAM terminates from a HALT command, abend, or VTAM-detected error.

Exit Points

The following table provides a cross-reference from the IBM-defined exit points to the RSCS modules from which they are called.

Exit Point	Name	Module
0	Initialization	DMTREX
1	Termination	DMTREX
2	Spool File Accept Accounting	DMTAXM
3	Spool File Send Accounting	DMTAXM
4	Spool File Purge Accounting	DMTAXM
5	Spool File Receive Accounting	DMTAXM
6	TAG Priority Change	DMTAXM
7	Auto-answer Sign-on Time Out	DMTDUP
8	Auto-answer Unrecognizable Data	DMTDUP
9	Auto-answer Sign-on Validation	DMTDUP
10	Auto-answer Sign-on Reject	DMTDUP
11	NJE Job Header Creation	DMTNHE
12	NJE Data Set Header Creation	DMTNHE
13	NJE Job Trailer Creation	DMTNHE
14	NJE Job Header Reception	DMTNHD
15	NJE Data Set Header Reception	DMTNHD
16	NJE Job Trailer Reception	DMTNHD
17	Separator Page Selection	DMTSEP
18	Separator Page Generation	DMTSEP
19	Command Screening	DMTCMX
21	Spool File Accept/Reject	DMTAXM

Exit Point	Name	Module
22	NOTIFY Driver Note Selection	DMTNOT
23	NOTIFY Driver Note Editing	DMTNOT
24	Spooling CP Command Screening	DMTAXM
25	Post-CP Command Screening	DMTAXM
26	Link State Change Accounting	DMTCOM
27	Message Request Screening	DMTMGX
28	Message Language Selection	DMTMGI
29	Unknown Command	DMTCMX
30	Reroute Interception	DMTRER
31	Sort Priority Change	DMTAXM
32	NMR Reception	DMTRGX
33	User Parm Processing	DMTCMY
34	Spool Manager Command	DMTAXM
35	Dump Processing	DMTMAN
36	NOTIFY Driver Purge	DMTNOT
37	NJE Job Header Transmission	DMTGPI, DMTNTR
38	NJE Data Set Header Transmission	DMTGPI, DMTNTR
39	NJE Job Trailer Transmission	DMTGPI, DMTNTR
40	NJE Record Reception	DMTGPI, DMTNRV
41	NJE Job Header Post-Processing	DMTNHD
42	NJE Data Set Header Post-Processing	DMTNHD
43	NJE Job Trailer Post-Processing	DMTNHD
44	Link Termination	DMTRPT, DMTSPT
45	Output Page Accounting	DMTRPT, DMTSPT
46	Verification of Page Accounting	DMTRPT, DMTSPT
47	Driver Initialization	DMTSPT
48	Verification of Output Page Error	DMTSPT

Dump Formatting Routines

RSCS provides the following modules to interact with the Dump Viewing Facility to format various data areas from an RSCS dump. These modules run on CMS and are not link-edited into the RSCS load library. See [Chapter 14, “Examining Dumps,” on page 157](#) for more information about the corresponding subcommands.

DMTYCV

DMTYCV locates the CVT in the RSCS-formatted dump and displays its contents as hexadecimal data. Its executable entry point, DMTYCV, is serially reusable.

DMTYDS

The Dump Viewing Facility calls DMTYDS to determine which RSCS formatting routine is needed to display the data area specified on the corresponding subcommand. DMTYDS validates the RSCS load address passed by the calling component and calls the appropriate formatting routine. Its one executable entry point, DMTYDS, is serially reusable.

DMTYEX

DMTYEX extracts data from the RSCS-formatted dump and adds it to the problem report created by the Dump Viewing Facility. Its executable entry point, DMTYEX, is serially reusable.

DMTYIO

DMTYIO locates the specified IOTABLE in the RSCS-formatted dump and displays its contents as hexadecimal data. Its executable entry point, DMTYIO, is serially reusable.

DMTYIT

DMTYIT finds the RSCS internal trace table in the RSCS-formatted dump. It then displays the trace records specified on the ITRACE subcommand. Its executable entry point, DMTYIT, is serially reusable.

DMTYLI

DMTYLI finds the LINKTABL entry for the specified link and displays its contents in hexadecimal format. Its executable entry point, DMTYLI, is serially reusable.

DMTYND

DMTYND finds the networking dynamic work area (NDWA) for the specified networking link and displays its contents in hexadecimal format. Its executable entry point, DMTYND, is serially reusable.

DMTYRI

DMTYRI finds the specified receive information block (RIB) in the RSCS-formatted dump and displays its contents in hexadecimal format. Its executable entry point, DMTYRI, is serially reusable.

DMTYRO

DMTYRO finds the specified ROUTE table and displays its contents in hexadecimal format. Its executable entry point, DMTYRO, is serially reusable.

DMTYTG

DMTYTG finds the specified TAG element in the RSCS-formatted dump and displays its contents in hexadecimal format. Its executable entry point, DMTYTG, is serially reusable.

DMTYTI

DMTYTI finds the specified transmit information block (TIB) in the RSCS-formatted dump and displays its contents in hexadecimal format. Its executable entry point, DMTYTI, is serially reusable.

Chapter 17. Control Blocks

This chapter describes the primary control blocks used by many of the RSCS tasks. These data areas are presented for diagnostic purposes only.

Offsets are shown in hexadecimal notation at the left of each diagram. Following each diagram is a table that presents the hexadecimal offset, name, type, and description of each field.

Primary Data Areas

The primary RSCS data areas, CRV and CVT, are accessed by all RSCS tasks and exit routines. For more information about these data areas, see [“Primary Data Areas” on page 16](#).

CRV

PI

The CRV (common routines vector table) contains pointers to various RSCS routines, some of which can be used by exit routines.



Attention: Some fields in this data area are not supported as programming interfaces.

X'000'	CAXMRQ	DC	V(DMTAXMRQ)	Spool manager request processor
X'004'	CBPLLX	DC	V(DMTBPLLX)	Load an exit (bomb-proof)
X'008'	CCOMDG	DC	V(DMTCOMDG)	EBCDIC (decimal) -> binary
X'00C'	CCOMDQ	DC	V(DMTCOMDQ)	(DEQUEUE) Get entry from queue
X'010'	CCOMFI	DC	V(DMTCOMFI)	Disk file interface routine
X'014'	CCOMGG	DC	V(DMTCOMGG)	(GETGROUP) Get routing group address
X'018'	CCOMGN	DC	V(DMTCOMGN)	(GETNODE) Get route for a node
X'01C'	CCOMHG	DC	V(DMTCOMHG)	EBCDIC (hex) -> binary
X'020'	CCOMLK	DC	V(DMTCOMLK)	(GETLINK) Get a LINKTABL address
X'024'	CCOMNQ	DC	V(DMTCOMNQ)	(ENQUEUE) Put entry on queue
X'028'	CCOMSM	DC	V(DMTCOMSM)	Send a msg/cmd down a link
X'02C'	CCOMTE	DC	V(DMTCOMTE)	S/370 TOD clock -> EBCDIC
X'030'	CCOMTS	DC	V(DMTCOMTS)	EBCDIC -> S/370 TOD clock
X'034'	CDDLEP	DC	V(DMTDDLEP)	Convert CMS file to NETDATA
X'038'	CHASHA	DC	V(DMTHASHA)	Add an element to a hash table
X'03C'	CHASHB	DC	V(DMTHASHB)	Build a hash table
X'040'	CHASHC	DC	V(DMTHASHC)	Unallocate (Freemain) a hash table
X'044'	CHASHD	DC	V(DMTHASHD)	Delete an element from a hash table
X'048'	CHASHF	DC	V(DMTHASHF)	Find an element in a hash table
X'04C'	CHASHG	DC	V(DMTHASHG)	Find an element in a hash table
	*			... with support for generic keys
X'050'	CHASHS	DC	V(DMTHASHS)	Update hash table statistical counts
X'054'	CIOTHD	DC	V(DMTIOTHD)	General I/O halt routine
X'058'	CIOTST	DC	V(DMTIOTST)	General I/O start routine
X'05C'	CLOGCL	DC	V(DMTLOGCL)	Link driver trace close routine
X'060'	CLOGEP	DC	V(DMTLOGEP)	Link driver trace routine
X'064'	CMADE	DC	V(DMTMANDE)	Link driver ESTAE
X'068'	CMGFFM	DC	V(DMTMGFFM)	Format message lines
X'06C'	CMGXEP	DC	V(DMTMGXEP)	Issue a message
X'070'	CMPTBP	DC	V(DMTMPTBP)	Branch on parameter
X'074'	CMPTCK	DC	V(DMTMPTCK)	Check keyword
X'078'	CMPTGD	DC	V(DMTMPTGD)	Get decimal
X'07C'	CMPTGP	DC	V(DMTMPTGP)	Get parameter
X'080'	CPAREP	DC	V(DMTPAREP)	Parameter parsing routine
X'084'	CPRDDQ	DC	V(DMTPRDDQ)	DEQ a message PRDBLOK
X'088'	CPRDNQ	DC	V(DMTPRDNQ)	ENQ a message PRDBLOK
X'08C'	CQSAAB	DC	V(DMTQSAAB)	Quick storage allocate buffer
X'090'	CQSAFA	DC	V(DMTQSAFA)	Quick storage free all buffers
X'094'	CQSAUB	DC	V(DMTQSAUB)	Quick storage unallocate buffer
X'098'	CRDREP	DC	V(DMTRDREP)	Spool record read routine
X'09C'	CRDROP	DC	V(DMTRDROP)	Spool record read open routine
X'0A0'	CRERSC	DC	V(DMTRERSC)	Reroute scanning routine
X'0A4'	CRESLO	DC	V(DMTRESLO)	Claim a lock
X'0A8'	CRESUN	DC	V(DMTRESUN)	Release a lock
X'0AC'	CSEPBL	DC	V(DMTSEPBL)	Block letter routine
X'0B0'	CSOKET	DC	V(DMTSOKET)	START SOCKET function
X'0B4'	CTASKA	DC	V(DMTTASKA)	Add a TASKBLOK entry

X'0B8'	CTASKD	DC	V(DMTTASKD)	Delete a TASKBLOK entry
X'0BC'	CTASKF	DC	V(DMTTASKF)	Find a TASKBLOK entry
X'0C0'	CTASKG	DC	V(DMTTASKG)	Find a TASKBLOK entry - FLS
X'0C4'	CUR0EP	DC	V(DMTUROEP)	Spool record write routine
X'0C8'	CUR0FL	DC	V(DMTUROFL)	Spool record flush routine

NOT Programming Interface Information

X'0CC'	CCOMLS	DC	V(DMTCOMLS)	(LINKSTATE) Exit for link state change
X'0D0'	CRGXEP	DC	V(DMTRGXEP)	Remote message and command handler

End of NOT Programming Interface Information

X'0D4'	CAXMCM	DC	V(DMTAXMCM)	Spool manager command ECB
X'0D8'	CAXMCQ	DC	V(DMTAXMCQ)	Spool manager command anchor
X'0DC'	CBOXPR	DC	V(DMTBOXPR)	Printer box image
X'0E0'	CCOMTN	DC	V(DMTCOMTN)	Local time zone abbreviation
X'0E4'	CCOMTO	DC	V(DMTCOMTO)	Addr of time zone offset (TOD form)
X'0E8'	CEVECM	DC	V(DMTEVECM)	Event manager command ECB
X'0EC'	CEVECQ	DC	V(DMTEVECQ)	Event manager command anchor
X'0F0'	CIRWLK	DC	V(DMTIRWLK)	Prototype LINKTABL image
X'0F4'	CIRWTA	DC	V(DMTIRWTA)	Prototype TAG entry image
X'0F8'	CIRXHL	DC	V(DMTIRXHL)	HASHBLOK for LINKTABLS
X'0FC'	CIRXHN	DC	V(DMTIRXHN)	HASHBLOK for NODES
X'100'	CIRXHR	DC	V(DMTIRXHR)	HASHBLOK for ROUTEGRPs
X'104'	CQSAAU	DC	V(DMTQSAAU)	Anchor for auth block
X'108'	CQSAEC	DC	V(DMTQSAEC)	QSABLOK for conditional 256 bytes
X'10C'	CQSAEU	DC	V(DMTQSAEU)	QSABLOK for unconditional 256 bytes
X'110'	CQSAEV	DC	V(DMTQSAEV)	Anchor for event block
X'114'	CQSAMB	DC	V(DMTQSAMB)	Anchor for message block
X'118'	CQSAML	DC	V(DMTQSAML)	Anchor for message line area
X'11C'	CQSAMW	DC	V(DMTQSAMW)	Anchor for message work area
X'120'	CREXCM	DC	V(DMTREXCM)	Internal command ECB
X'124'	CREXCQ	DC	V(DMTREXCQ)	Internal command anchor
X'128'	CREXME	DC	V(DMTREXME)	Asynchronous message ECB
X'12C'	CREXMQ	DC	V(DMTREXMQ)	Asynchronous message anchor
X'130'	CREXTE	DC	V(DMTREXTE)	REX termination ECB
X'134'	CSCTAC	DC	V(DMTSCTAC)	ACB for VTAM interface

NOT Programming Interface Information

X'138'	CIRWAU	DC	V(DMTIRWAU)	Prototype AUTH entry image
X'13C'	CITRFT	DC	V(DMTITRFT)	Pointer to the ITRACE format table index

End of NOT Programming Interface Information

X'140'	CASTCM	DC	V(DMTASTCM)	Retry/I/O timer task command ECB
X'144'	CASTCQ	DC	V(DMTASTCQ)	Retry/I/O timer task command queue
	CRVLEN	EQU	*-&LABEL	Length of CRV

PI end

CVT

PI

The CVT (communications vector table) contains information about some RSCS data structures and queues.



Attention: Some fields in this data area are not supported as programming interfaces.

X'000'	TLINKS	DC	A(0)	Anchor for LINKTABL entries
X'004'	TROUTEGP	DC	A(0)	Anchor for final ROUTEGP entries
X'008'	TPORTS	DC	A(0)	Address of PORT table
X'00C'	TTAGQ	DC	A(0)	Address of TAG slot queue
X'010'	TVMID	DC	A(0)	Address of RSCS virtual machine ID
X'014'	TAUTH	DC	A(0)	Address of authorization table
X'018'	TREROUTE	DC	A(0)	Address of REROUTE table
X'01C'	TREROUTX	DC	A(0)	End of global REROUTE chain
X'020'	TCHANNEL	DC	A(0)	Address of unit record pool vector
X'024'	TDEST	DC	A(0)	Address of DEST table
X'028'	TEXTITS	DC	A(0)	Address of EXIT table
X'02C'	TLISTPR	DC	A(0)	Address of *LIST LINKTABL

X'030'	TUSER	DC	XL8'00'	For user exit usage
	*	DC	0D'0'	Align for counters
X'038'	TPTHECNT	DC	F'0',F'0'	Curr. active dial-up ports
X'040'	TLCA	DC	F'0',F'0'	Currently active links
X'048'	TLCASNA	DC	F'0',F'0'	Currently active SNA links
X'050'	TMAXHOPS	DC	H'64'	Maximum number of node hops
	*			... a file is allowed to make
X'052'	TQMSGGLIM	DC	H'0'	Q message limit - 1-32700
	*			0 is default - means no limit
X'054'	TGLOBAL1	DC	XL1'00'	Global status switch 1
	*			
	*		Bits defined in TGLOBAL1	
	*			
	TGSSNAUP	EQU	X'80'	RSCS/VTAM interface is active
	TGSSIP	EQU	X'40'	SHUTDOWN in progress
	TGSVSIP	EQU	X'20'	RSCS/VTAM interface stop in progress
	TGSVSUIP	EQU	X'10'	RSCS/VTAM interface startup in progress
	TGSREADY	EQU	X'08'	RSCS ready (initialization complete)
	TGSCACB	EQU	X'04'	RSCS/VTAM interface ACB is being closed
	TGSNOGO	EQU	X'02'	RSCS must be reloaded
	TITRACE	EQU	X'01'	ITRACEing is active
X'055'	TGLOBAL2	DC	XL1'00'	Global status switch 2
	*			
	*		Bits defined in TGLOBAL2	
	*			
	TGSACTIV	EQU	X'80'	RSCS is active
	TGSEE	EQU	X'40'	RSCS exec-in-execution state
	TGDIALAC	EQU	X'10'	Link timeout scan active
	TGSTCUP	EQU	X'08'	Port redirector task is up
	TCPZVM	EQU	X'01'	Identify z/VM 3.1 & above
X'056'	TGLOBAL3	DC	AL1(TGCOA+TLOOPH+TLOOPI)	Global status switch 3
	*			
	*		Bits defined in TGLOBAL3	
	*			
	TGCOF	EQU	X'80'	Forwarding messages
	TGCOA	EQU	X'40'	Final messages
	TGENQM	EQU	X'20'	Enqueued messages
	*			Additional flags:
	TGACCM	EQU	X'10'	Accept messages
	TLOOPI	EQU	X'08'	Immediate loop detection
	TLOOPH	EQU	X'04'	Hop count monitoring
	TGLISTP	EQU	X'02'	Remote sys's have LISTPROC
X'057'	TGLOBAL4	DC	XL1'00'	Global status switch 4
	*			
	*		Bits defined in TGLOBAL4	
	*			
	TJNAMUSR	EQU	X'80'	Use origin user ID as jobname
	*			... on NJE type links
X'058'	TCPFEAT	DC	AL1(TCPVAFP)	VM feature flags
	*		SET DEFAULT TO VAFP YES	
	*		Bits defined in TCPFEAT	
	*			
	TCPSOIDY	EQU	X'20'	SECO=YES was specified on
	*			... the OPTION statement
	TCPSOIDN	EQU	X'10'	SECO=NO was specified on
	*			... the OPTION statement
	TCP5DIG	EQU	X'08'	Use 5-digit device addrs on CP commands
	TCPVAFP	EQU	X'04'	VAFP devices allowed
	TCPSTOP	EQU	X'02'	STOP commands may be passed
	*			to the SCT task that will
	*			use the TERMQ option on the
	*			CLSDST to seek and destroy
	*			queued SIMLOGON requests.
X'059'	TMAXDSH	DC	AL1(10)	Global MAXDSH value
X'05A'	TMSGSKIP	DC	AL1(2)	Global MSGSKIP value
X'05B'	THIDECHR	DC	C'\'	Character used to hide things
X'05C'	TMONITOR	DC	A(0)	Anchor for '*' monitoring
X'060'	TMONIMSG	DC	A(0)	Anchor for msg monitoring
	*			... by message numbers
X'064'	TFORMTAB	DC	A(0)	Address of FORM table
X'068'	TCRVTAB	DC	V(DMTCRVEP)	Address of CRV table
X'06C'	TFILWRKS	DC	A(0)	Anchor for file work areas

SYSIDENT

X'070'	TFILDDEF EQU 10			Default maximum depth of imbeds
	TDDNMVEC DC A(0)			Anchor for ddname usage vector
	TMAXOPEN EQU 1000			Max. number of dynamic ddnames
X'074'	TEVENTS DC A(0)			Anchor for EVEBLOK chain
X'078'	TTANQ DC A(0)			Anchor for TANBLOK chain
X'07C'	TSHIFT DC A(0)			Number last set by SHIFT cmd

NOT Programming Interface Information				
X'080'	TEQUATE DC V(DMTEQUEP)			Anchor for EQUATE chain

End of NOT Programming Interface Information				
X'084'	TRESOURC DC A(0)			Anchor for resource chain
X'088'	TFCBTABA DC A(0)			Anchor for the FCB table
X'08C'	TRECOVER DC A(0)			Pointer to recovery command
X'090'	TITRACEA DC A(0)			Anchor for ITRACE table
X'094'	TTASKTAB DC A(0)			Anchor for TASK table
X'098'	TTRACEN DC CL8' '			Default node ID for trace files
X'0A0'	TTRACEU DC CL8' '			Default user ID for trace files
X'0A8'	TIPCSA DC 0F'0'			Dump Viewing Facility area
X'0A8'	TIREND DC V(DMTEND)			Address of end of the RSCS module
X'0AC'	TIIVERN DC CL8'FLnnn'			Version number
X'0B4'	TIMAINTL DC CL8'-0000'			Maintenance level
X'0BC'	TILASTCM DC CL8' '			Last command
X'0C4'	TICOMPID DC CL9'568409601'			Component ID
X'0CD'	TISLVLR DC CL2' '			z/VM release number
X'0CF'	TISLVLV DC CL2' '			z/VM modification level
X'0D1'	TISLVLV DC CL4' '			z/VM PLC number
X'0D5'	TILASTMS DC CL10' '			Last RSCS message issued
X'0DF'	TIPLTIME DC CL8' '			IPL date and time
X'0E7'	TISLVER DC CL2' '			Version number of product
	TCVTLEN EQU *-&LABEL			Length of CVT area

PI end

SYSIDENT

PI

The SYSIDENT table contains information about each RSCS system task.

X'000'	SYSECB DC F'0'			Task terminated ECB
X'004'	SYSTAID DC F'0'			Task ID
X'008'	SYSNAME DC CL8			Task name
	SYSNAME3 EQU SYSNAME+3,3			Important part of task name
X'010'	SYSFLAG1 DC X'00',X'000000'			First flag (plus reserved)
X'014'	SYSITRAO DC A(0)			Pointer to override byte
*				map for ITRACE settings
	SYSIDLEN EQU *-&LABEL			Length of a SYSIDENT

PI end

Network and Task Structure

The DEST, EQUATE, LINKTABL, PORT, and ROUTE data areas contain information that define the network structure to RSCS. See [Chapter 2, “RSCS Structure,”](#) on page 11 for more information.

DEST

PI

The DEST (destination table) contains a list of PSF destination names. Each entry is chained to the next; the end of the chain indicated by a word of zeros in the DESTNEXT field. The TDEST field in the CVT contains the address of the first destination table entry. See [“DEST”](#) on page 20 for more information.

X'000'	DESTNEXT	DC	A(0)	Address of next destination table entry
X'004'	DESTRESV	DC	A(0)	Reserved
X'008'	DESTNAME	DC	CL8' '	Destination name
	DESTLEN	EQU	*-&LABEL	Destination table entry length

PI end

EQUATE

The Equate Table, found in DMTEQU, contains information about RSCS tasks. The table has an entry for each link driver and system task and a last, empty, entry that contains blanks in the EQUPEP field. The system tasks have blanks in the EQUPEP field. See [“System Task Equates” on page 28](#) for more information.

X'000'	EQUPEP	DC	A(0)	Chain pointer
X'004'	EQUPELOC	DC	A(0)	Entry point address
X'008'	EQUPEP	DC	CL8' '	Symbolic name
X'010'	EQUPEP	DC	CL8' '	Entry point name
X'018'	EQUFLAG	DC	XL1'00'	Entry point type flags
X'019'	EQUSTAT	DC	XL1'00'	Equate status flags
X'01A'	EQUFLAG2	DC	XL1'00'	Driver type flags
	*			
	*		Bits defined in EQUSTAT	
	*			
	EQUPESTOR	EQU	X'80'	Storage for entry was allocated dynamically
	EQUPELIN	EQU	X'40'	Line address not used for this link type
X'01B'	EQUPEVD1	DC	XL5'00'	Reserved
	EQUPELEN	EQU	*-&LABEL	Length of table entry

LINKTABL

PI

The LINKTABL describes all the characteristics of an RSCS link. Each link table entry describes one link in the network.

NOT Programming Interface Information

See [“LINKTABL” on page 17](#) for more information.

End of NOT Programming Interface Information



Attention: Some fields in this data area are not supported as programming interfaces.

X'000'	LINKID	DC	CL8' '	EBCDIC link ID
X'008'	LINKNEXT	DC	A(0)	Address of next link table entry
X'00C'	LINKHASH	DC	A(0)	Link ID hash chain
X'010'	LINKHLUA	DC	A(0)	Active/default LU hash chain
X'014'	LINKHLUD	DC	A(0)	Default LU hash chain if link started with different LName
	*			
X'018'	LINKHCID	DC	A(0)	CID hash chain

NOT Programming Interface Information

X'01C'		DC	A(0)	Reserved
--------	--	----	------	----------

End of NOT Programming Interface Information

LINKINFO	EQU	*		Here begins the real info
	*			

NOT Programming Interface Information

*	Note:	To streamline many parts of RSCS operation that require
*		checking active fields as opposed to default fields when a
*		link is active, it is hereby guaranteed that when a link is
*		inactive, all active fields shall contain the same information

* as the corresponding default fields. The code to do this
 * resides in CMY, IRX and MAN.

End of NOT Programming Interface Information

*
 * Active fields
 *
 LACTFLD EQU * Active fields
 X'020' LACTSYM DC CL8'UNDEFIND' Type

NOT Programming Interface Information

X'028' LACTDRVR DC CL8' ' EName of driver

End of NOT Programming Interface Information

X'030' LACTLUN DC CL8' ' LUname
 X'038' LACTLOG DC CL8' ' Logmode
 X'040' LACTCLS1 DC CL1' ' Class 1
 X'041' LACTCLS2 DC CL1' ' Class 2
 X'042' LACTCLS3 DC CL1' ' Class 3
 X'043' LACTCLS4 DC CL1' ' Class 4
 X'044' LACTPARM DC A(0) Address of active parm
 X'048' LACTDP DC H'5' Dispatching priority
 X'04A' LACTLINE DC AL2(0) Line address
 X'04C' LACTTYP1 DC XL1'00' Link type
 X'04D' LACTTYP2 DC XL1'00' Link type2 flags

NOT Programming Interface Information

X'04E' DC XL2'00' Filler

End of NOT Programming Interface Information

LACTFLEN EQU *-LACTFLD Active fields length
 *
 * Default fields
 *
 LDEFFLD EQU * Default fields
 X'050' LDEFSYM DC CL8'UNDEFIND' Default driver symbolic name

NOT Programming Interface Information

X'058' LDEFDRVR DC CL8' ' Default EName of driver

End of NOT Programming Interface Information

X'060' LDEFLUN DC CL8' ' Default logical unit name
 X'068' LDEFLOG DC CL8' ' Default logmode table name
 X'070' LDEFCLS1 DC CL1'*' Default spool file class 1
 X'071' LDEFCLS2 DC CL1' ' Default spool file class 2
 X'072' LDEFCLS3 DC CL1' ' Default spool file class 3
 X'073' LDEFCLS4 DC CL1' ' Default spool file class 4
 X'074' LDEFPARM DC A(0) Address of default parm
 X'078' LDEFDP DC H'5' Default dispatching priority
 X'07A' LDEFLINE DC AL2(0) Default virtual line address
 X'07C' LDEFTYP1 DC XL1'00' Default link type flag
 *
 * Bits defined in LDEFTYP1/LACTTYP1
 *
 LNET EQU X'80' Networking link identifier
 LSNA EQU X'40' SNA link identifier
 LPRT EQU X'20' 3270 printer link identifier
 LLIS EQU X'10' List Processor Identifier
 LASCII EQU X'08' ASCII link identifier
 LGPI EQU X'04' Gateway link identifier
 LTCP EQU X'02' TCP link identifier
 LNOT EQU X'01' Notify link identifier
 X'07D' LDEFTYP2 DC XL1'00' Default link type2 flag
 *
 * Bits defined in LDEFTYP2/LACTTYP2
 *

	LDEFORM	EQU	X'80'	Forms control for link
	LDEFTRC	EQU	X'40'	Trace specified in LINKDEF

NOT Programming Interface Information				
X'07E'		DC	XL2'00'	Filler
X'080'	*	DC	CL4' '	Reserved

End of NOT Programming Interface Information				
--	--	--	--	--

X'084'	LOLDPARM	DC	A(0)	Override parms
X'088'	LUSRPARM	DC	A(0)	Address of user parms
	*			
	LST		CMORIG DSECT=NO	Start command origin
X'08C'	LSTQUAL	DC	AL1(0)	Origin qualifier
X'08D'	LSTFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'08E'	LSTRSPC	DC	H'0'	Response counter
X'090'	LSTNODE	DC	CL8' '	Origin node
X'098'	LSTUSER	DC	CL8' '	And user ID
X'0A0'	LSTSIG	DC	CL6' '	Response signature
	LCM		CMORIG DSECT=NO	Command origin (active link)
X'0A6'	LCMQUAL	DC	AL1(0)	Origin qualifier
X'0A7'	LCMFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'0A8'	LCMRSPC	DC	H'0'	Response counter
X'0AA'	LCMNODE	DC	CL8' '	Origin node
X'0B2'	LCMUSER	DC	CL8' '	And user ID
X'0BA'	LCMSIG	DC	CL6' '	Response signature
X'0C0'	LCURFORM	DC	CL8'STANDARD'	Current form name
X'0C8'	LCID	DC	F'0'	SNA CID
X'0CC'	LCMDECB	DC	F'0'	Command ECB
X'0D0'	LMSGECB	DC	F'0'	Message ECB
X'0D4'	LFILECB	DC	F'0'	File available ECB
X'0D8'	LETXECB	DC	F'0'	End-of-task ECB
X'0DC'	LTRECECB	DC	F'0'	Driver terminate ECB
X'0E0'	LRELECB	DC	F'0'	Relreq exit ECB
X'0E4'	LRECECB	DC	F'0'	Receive any ECB
	LPRDECB	EQU	LRECECB,4	Port redirector ECB
X'0E8'	LSCPECB	DC	F'0'	SCIP exit ECB
	LFLAGF	DS	0XL4	Fullword flags for easy msgs
X'0EC'	LFLAG	DC	XL1'00'	Link table status flag byte

NOT Programming Interface Information				
*	*** IMPORTANT ***			
*				
*	NOTE: The LFLAG macro acts as an interface to the LFLAG bits.			
*	Thus, all changes/additions/deletions to to LFLAGS MUST be			
*	reflected to the LFLAG macro.			
*				
*	*** IMPORTANT ***			

End of NOT Programming Interface Information				
--	--	--	--	--

*				
*	Bits defined in LFLAG			
*				
LACTIVE	EQU	X'80'	Link active	
LRPLWAIT	EQU	X'40'	Indicate that a SNA link is	
*			waiting for a SIMLOGON RPL	
*			to become available for the	
*			start command being	
*			processed by the SCT task.	
LHOLD	EQU	X'20'	Link HOLD set	
LDRAIN	EQU	X'10'	Link DRAIN in progress	
LCONNECT	EQU	X'08'	Link CONNECTed	
LABEND	EQU	X'04'	Link has abended	
LSTART	EQU	X'02'	START command issued for inactive link	
LHALT	EQU	X'01'	Link was forced	
*				
L4STA	EQU	LSTART*16777216	Word-aligned LSTART	
L4ACT	EQU	X'80000000'	Word-aligned LACTIVE	
L4CON	EQU	LCONNECT*16777216	Word-aligned LCONNECT	
L4HOLD	EQU	LHOLD*16777216	Word-aligned LHOLD	
L4RPLWT	EQU	LRPLWAIT*16777216	Word-aligned LRPLWAIT	

X'0ED'	LFLAG1	DC	XL1'00'	Link table status flag byte
	*			
	*		Bits defined in LFLAG1	
	*			
	LLOGWAIT	EQU	X'80'	SIMLOGON complete
	LTRLOG	EQU	X'40'	Link transaction tracing (log)
	LTRALL	EQU	X'20'	Link transaction tracing (all)
	LTRREC	EQU	X'10'	Link transaction tracing (record)
	LDIALED	EQU	X'08'	Link dialed to dial-up task
	LINTREQ	EQU	X'04'	Intervention required on printer
	LRELSL	EQU	X'02'	SNA session has been released
	LDIALOUT	EQU	X'01'	Link has dialed out
	L4INT	EQU	LINTREQ*65536	Word-aligned LINTREQ
	L4REL	EQU	LRELSL*65536	Word-aligned LRELSL
	L4LOGWT	EQU	LLOGWAIT*65536	Word-aligned LLOGWAIT
X'0EE'	LFLAG2	DC	XL1'00'	Link table status flag byte
	*			
	*		Bits defined in LFLAG2	
	*			
	LPROPTD	EQU	X'80'	Mount has been issued
	LAUTO	EQU	X'40'	Auto specified on start
	LSETUP	EQU	X'20'	Setup specified on start
	LSEC	EQU	X'10'	SNA NJE secondary link identifier
	LWAITM	EQU	X'08'	Waiting for form mount
	LDOSET	EQU	X'04'	Performing setup operation
	LFORMP	EQU	X'02'	Form control allowed
	LPUNOK	EQU	X'01'	Punch files accepted
	L4AUTO	EQU	LAUTO*256	Word-aligned LAUTO
	L4SETUP	EQU	LSETUP*256	Word-aligned LSETUP
X'0EF'	LFLAG3	DC	XL1'00'	Link table status flag byte
	*			
	*		Bits defined in LFLAG3	
	*			
	LASISTR	EQU	X'80'	Queued for AUTOSTART
	LASIDLM	EQU	X'40'	Attached by dialing manager
	LASIELIG	EQU	X'20'	Link eligible for auto start
	LOPARM	EQU	X'10'	Override parm specified
	LRTRY	EQU	X'08'	Link eligible for start retry
	LRTRING	EQU	X'04'	Link is retrying start
	LHOLDOUT	EQU	X'02'	Other side has input slowdown
	LHOLDINP	EQU	X'01'	Input streams are held
	L4DQU	EQU	LASISTR	Word-aligned LASISTR
	L4RET	EQU	LRTRING	Word-aligned LRTRING
	L4HINP	EQU	LHOLDINP	Word-aligned LHOLDINP
	L4HOUT	EQU	LHOLDOUT	Word-aligned LHOLDOUT
X'0F0'	LFLAG4	DC	XL1'00'	Link table status flag byte
	*			
	*		Bits defined in LFLAG4	
	*			
	LFCBDYNA	EQU	X'80'	FCB=<fcbyname> DYNAMIC
	LFCBNAME	EQU	X'40'	FCB=fcbyname
	LSTOP	EQU	X'20'	Stop CMD issued for SNANJE
	LSLOUNIQ	EQU	X'10'	Unique slowdown entry/exit
	LORDERED	EQU	X'08'	Shadow was ordered on this link
	*			(used for CHANGE cmd processing in AXM)
	LDORTRY	EQU	X'04'	Indicate RETRY upon link
	*			deactivation
	LALERT	EQU	X'02'	Link to be alerted when file
	*			arrives
	LSHUTIP	EQU	X'01'	Link shutdown in progress
	LFLAGLEN	EQU	*-LFLAG	Length (bytes) of flags
	LFLAGS	EQU	LFLAGF,5	All the LFLAGS
X'0F1'	LITODEF	DC	AL1(0)	Default dial-out max time
X'0F2'	LITOCUR	DC	AL1(0)	Current dial-out time left
	*			
X'0F3'	LQUEFLAG	DC	X'00'	Flag used for link queueing info
	*			
	*		Bits defined in LQUEFLAG	
	*			
	LFIFO	EQU	X'80'	Queueing on FIFO basis
	LSIZE	EQU	X'40'	Queueing on SIZE basis
	LQCHANGE	EQU	X'20'	Queueing has just been changed
	LQWCHANG	EQU	X'10'	Queueing changes when link deactivates
	LHLDINPQ	EQU	X'08'	A HOLD INPUT CMD was queued

NOT Programming Interface Information

*
 * Note: The following 10 pointers must remain together; they are
 * treated by AXM as a single vector and are used to speed
 * up the process of placing a shadow element on a queue.
 *

End of NOT Programming Interface Information

X'0F4'	LINPUTQ	DC	A(0)	Input file shadow element queue
X'0F8'	LINPUTQI	DC	8A(0)	Intermediate ptrs for fast access
	LINPUTQL	EQU	LINPUTQI+7*4,4	Address of last of the pointers
	LINPQLEN	EQU	*-LINPUTQI	Number of bytes pointers reside in
	LINQFORK	EQU	LINPQLEN/4	Number of 'fork tines' we use
X'118'	LINPUTQE	DC	A(0)	End of shadow element queue
X'11C'	LINPUTPO	DC	8H'0'	Position counters for the pointers
	LINPUTPL	EQU	LINPUTPO+7*2,2	Address of last position counter
	LINPPLEN	EQU	*-LINPUTPO	Number of bytes counters reside in
X'12C'	LRECRPLA	DC	A(0)	Address of receive (any) RPL copy
X'130'	LCMDQ	DC	A(0)	Command queue anchor
X'134'	MSGQ	DC	A(0)	Message queue anchor
X'138'	LUWORD	DC	A(0)	User word (defined by link driver)
X'13C'	LSTATPTR	DC	A(0)	Pointer to LINKSTAT element
X'140'	LUSER	DC	XL8'00'	For user exit usage
				Multi-streaming control fields
X'148'	LMSSMAX	DC	AL1(0)	Maximum no. of active streams
X'149'	LMSTAFLG	DC	X'00'	TA related flag byte
				Flags defined in LMSTAFLG
	LMSTAIN	EQU	X'80'	Using internal TA
	LMSTAEXT	EQU	X'40'	Using external TA

NOT Programming Interface Information

X'14A'	LFNUMBER	DC	PL2'0'	RFC1179 Job number
--------	----------	----	--------	--------------------

End of NOT Programming Interface Information

X'14C'	LMSSACT	DC	XL4'00000000'	Active stream mask
X'150'	LMSSAVL	DC	XL4'00000000'	Available file mask
X'154'	LMSTAEP	DC	A(0)	Address of transmission algorithm
X'158'	LMSTAP	DC	A(0)	Address of TA parm text
				The shadow counters follow
X'15C'	LSHADCNT	DC	H'0'	Count of primary shadows +
				... eligible alternate shadows
X'15E'	LALTSCNT	DC	H'0'	Count of alternate shadows
				... not eligible for transmission
X'160'	LLOOPCNT	DC	H'0'	Count of looping shadow elements
X'162'	LACTICNT	DC	H'0'	Count of active input files
X'164'	LACTOCNT	DC	H'0'	Count of active output files
				(does not include TRACE file)
X'166'	LORDECNT	DC	H'0'	Number of ORDERed files
X'168'	LHOLDCNT	DC	H'0'	Count of held files
X'16A'	LTRNSCNT	DC	H'0'	Link transaction count
X'16C'	LERRCNT	DC	H'0'	Error count
X'16E'	LTOCNT	DC	H'0'	Timeout count
X'170'	LTASKID	DC	H'0'	Task ID
X'172'	LSEQNO	DC	H'0'	File sequence number
X'174'	LRETN	DC	AL1(0)	Number retries done
X'175'	LRETLEFT	DC	AL1(0)	Number mins to next retry
X'176'	LINVECTG	DC	AL2(0)	Init vector length
X'178'	LINVECTA	DC	AL4(0)	and address
X'17C'	LMONITOR	DC	A(0)	Anchor for link monitoring entries
X'180'	LNODEID	DC	CL8' '	Nodeid on other side of link
X'188'	LFANOUT	DC	CL8' '	Fanout linkid
X'190'	LSLOWDIF	DC	H'0'	Difference from base slowdown point

PORT

X'192'	LSLOWEN	DC	H'0'	Unique slowdown entry
X'194'	LSLOWEX	DC	H'0'	Unique slowdown exit

NOT Programming Interface Information

X'196'	DC	H'0'	Reserved
--------	----	------	----------

End of NOT Programming Interface Information

X'198'	LINKDWA	DC	A(0)	Address of DWA for link
X'19C'	LENDWA	DC	F'0'	Length of DWA
X'1A0'	LINKDWA2	DC	A(0)	Address of secondary DWA
X'1A4'	LENDWA2	DC	F'0'	Length of secondary DWA
X'1A8'	LFCBADDR	DC	A(0)	Anchor for FCB table list
X'1AC'	LITRACE0	DC	A(0)	Anchor for ITRACE overrides
X'1B0'	LBUFSIZE	DC	X'80000000'	Size of buffer used by link
*	LINKLEN	EQU	*-&LABEL	Length of link table entry
	LINIKLEN	EQU	*-LINKINFO	Length of LINKTABL - chains

PI end

PORT

The PORT (Port table) area describes the line ports available to RSCS. The TPORTS field of the CVT contains the address of the first entry of the port table queue. Each entry is chained to the next; the end of the queue indicated by a word of zeros in the PORTNEXT field.

X'000'	PORTNEXT	DC	A(0)	Address of next port table entry
X'004'	PORTLINK	DC	V(DMTIRWLK)	Address port's link
X'008'	PORTCUU	DC	XL2'00'	Port address
X'00A'	PORTFLAG	DC	XL1'00'	Flags
X'00B'	PORTFLG1	DC	XL1'00'	More flags
*	Bits defined in PORTFLAG			
*				
	PORTUSED	EQU	X'80'	Port in use
	PORTENAB	EQU	X'40'	Port enabled for use
	PORTDISA	EQU	X'20'	Port being disabled
	PORTREEN	EQU	X'10'	Port being reenabled
	PORTDIAL	EQU	X'08'	Port has autodial device for autostart usage
	PORTABND	EQU	X'04'	Port has abended
*	Bits defined in PORTFLG1			
*				
	PTRALL	EQU	X'80'	Trace all (like LTRALL)
	PTRLLOG	EQU	X'40'	Trace log (like LTRLLOG)
	PTRREC	EQU	X'20'	Trace records (like LTRREC)
	PITRACE	EQU	X'10'	ITRACE is on for this port
X'00C'	PORTECB	DC	A(0)	Terminate ECB for dial-up
X'010'	PORTETX	DC	A(0)	Task terminate ECB
X'014'	PORTFSAT	DC	AL2(0)	Number of failed signons
X'016'	PORTTSK	DC	H'0'	Taskid using this port
	PORTEORG	DS	0CL26	ENABLE port cmdnd originator
X'018'	POENQUAL	DC	AL1(0)	Origin qualifier
X'019'	POENFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'01A'	POENRSPC	DC	H'0'	Response counter
X'01C'	POENNODE	DC	CL8' '	Origin node
X'024'	POENUSER	DC	CL8' '	and userid
X'02C'	POENSIG	DC	CL6' '	Response signature
X'032'	PORTINVL	DC	AL1(0)	Leng of auto-ans init vector
X'034'	PORTINVA	DC	AL4(0)	Addr of auto-ans init vector
	PORTTTO	DS	0CL16	ENABLE port trace to user
X'038'	PORTTLOC	DC	CL8' '	ENABLE trace to locid
X'040'	PORTTVM	DC	CL8' '	ENABLE trace to vmid
X'048'	PITRACE0	DC	A(0)	Anchor for ITRACE overrides
	PORTLEN	EQU	*-&LABEL	Port table entry length

REROUTE

A REROUTE entry describes a reroute definition in the network. Each entry is chained to the next; the last entry in the chain is identified by zeros in the RERNEXT field. See [“REROUTE” on page 20](#) for more information.

```

X'000' RERNEXT DC A(0) Next in global chain
X'004' RERPREV DC A(0) Previous in global chain
X'008' RERHASH DC A(0) Next in RERFNODE hash chain
X'00C' RERSAME DC A(0) Next in same fornode/foruser chain
*
X'010' RERTYPE DC AL1(0) Type of reroute
X'011' RERFLAG DC AL1(0) Other flags
X'012' DC XL6'00' Reserved
*
X'018' RERFNODE DC CL8' ' For node
X'020' RERFUSER DC CL8' ' For user
X'028' RERTNODE DC CL8' ' To node
X'030' RERTUSER DC CL8' ' To user

```

```

*-----*
*      Bits defined in RERTYPE      *
*-----*

```

```

RERNTRCV EQU X'01' Not-received messages
RERFILES EQU X'02' Files
RERMSGSGS EQU X'04' Messages
RERCMDSGS EQU X'08' Commands
RERALL EQU RERMSGSGS+RERFILES Messages and files
RERTYPES EQU RERCMDSGS+RERMSGSGS+RERFILES+RERNTRCV All types

```

```

*-----*
*      Bits defined in RERFLAG      *
*-----*

```

```

RERQUIET EQU X'80' No message please
RERFLAGSGS EQU RERQUIET All flags
*
* Note: bits in RERFLAG and RERTYPE must not be the same,
* that is, RERTYPES .AND. RERFLAGSGS must equal B'00000000'
*

```

```

RERLEN EQU *-&LABEL Length of RERROUTE element

```

ROUTEGRP

A route group (ROUTEGRP) entry describes a group of nodes or a collection of groups in the RSCS network. See [“ROUTEGRP” on page 18](#) for more information.

```

X'000' ROUTNEXT DC A(0) Address of next ROUTEGRP entry
X'004' ROUTHASH DC A(0) Address of colliding ROUTEGRP
X'008' ROUTNAME DC CL8' ' Name of this routing group
*
X'010' ROUTPREV DC A(0) Address of previous ROUTEGRP entry
X'014' ROUTFLAG DC X'00' Routing flag byte
*
*      Bits defined in ROUTFLAG
*
ROUTHONR EQU X'80' This is an 'honorary' group
ROUTCHLD EQU X'40' This is just a child group
*
X'015' DC AL1(0) Spare byte
X'016' ROUTLNUM DC H'0' Number of links in ROUTLNKS vector
X'018' ROUTLNKS DC A(0) Pointer to LINKTABLs vector
X'01C' ROUTALNK DC A(0) Pointer to alternate LINKTABL
ROUTGDAD EQU ROUTLNKS Pointer to father group
X'020' ROUTNODA DC A(0) Pointer to first NODE in this group entry
X'024' ROUTGRPA DC A(0) Pointer to first ROUTEGRP child of this ROUTEGRP
X'028' ROUTGNXT DC A(0) Pointer to next ROUTEGRP sibling
X'02C' ROUTGPRV DC A(0) Pointer to previous ROUTEGRP sibling
X'030' ROUTSEND DC H'0' Number of files being sent
X'032' ROUTRECV DC H'0' Number of files being received
X'034' ROUTQUEU DC H'0' Number of inactive files for this routing group
X'036' ROUTHOLD DC H'0' Number of file in hold state
X'038' ROUTLOOP DC H'0' Number of files in hop-count loop
ROUTCNTL EQU *-ROUTSEND Length of counter section
DS 6X Pad to double-word
ROUTGLEN EQU *-&LABEL ROUTEGRP table entry length

```

TASKBLOK

A task block (TASKBLOK) describes a type of RSCS task (system, link driver, and auto answer); a TASKBLOK only represents an active task.

```

X'000' TASKCOLL DC    A(0)          Collision chain pointer for hashing
*                               algorithm
X'004'          *                               Reserved
X'008' TASKTOD  DC    D'0'          Creation time stamp.
X'010' TASKID   DC    H'0'          Task ID of the task being described
X'012' TASKNAME DC    CL8' '        Module name + EP (ie. DMTAXMEP,
*                               DMTNETP, DMTDUPEP)
X'01A' TASKDESC DC    CL20' '        Short description of task (ie. SYSTEM
*                               SPOOL, LINK 'linkid', PORT 'ccuu')
X'02E'          *                               Reserved
X'030' TASKDATA DC    A'0'          Pointer to data area inherent to this
*                               task (LINKTABL, PORT, SYSIDENT)
X'034' TASKFLAG DC    AL1(0)        Miscellaneous flag bits

*           Bits defined in TASKFLAG

TASKSYS EQU    X'80'          This is a system task
TASKLINK EQU   X'40'          This is a link driver task
TASKPORT EQU   X'20'          This is a PORT task
*                               X'10'          spare bit
*                               X'08'          spare bit
*                               X'04'          spare bit
*                               X'02'          spare bit
*                               X'01'          spare bit

X'038' TASKITPL DS    0F
X'038'          DC    XL(4+8*10)'00' Generate parm list
*                               TASKITLN EQU *TASKITPL Length of the plist

X'08C' TASKITR  DC    CL256'00'     Byte map of ITRACE settings for this task
*                               TASKBLN EQU  *-TASKBLOK Length of the sucker

```

Accounting Structures

RSCS uses the following structures to create accounting records and identify the users who are authorized to issue various commands.

ACNTBUFF

PI

The ACNTBUFF macro maps the format of the standard RSCS accounting record.

NOT Programming Interface Information

The AXM task creates an accounting record when RSCS receives or sends a file.

End of NOT Programming Interface Information

```

*           Local network userid fixed by CP
X'000' ACNTUSER DC    CL8' '        Originating location user ID
X'008' ACNTDATE DC    CL12' '        Date + time record cut (mmddyyhhmmss)
X'014' ACNTOID  DC    XL2'00'        Origin spool file ID
X'016' ACNTID   DC    XL2'00'        Local spool file ID
X'018' ACNTILOC DC    CL8' '        Originating location ID
X'020' ACNTDEST DC    CL8' '        Destination location ID
X'028' ACNTCLAS DC    CL1' '        Class
X'029' ACNTINDV DC    XL1'00'        Origin device type ('8N'=PUN/'4N'=PRT)
X'02A'          DC    CL2' '        Filler
X'02C' ACNTRECS DC    XL4'00'        Number of records in file
X'030' ACNTTOVM DC    CL8' '        Destination location user ID
X'038'          DC    CL8' '        Filler
X'040' ACNTSYS  DC    CL5' '        System ID (serial + model)
X'045' ACNTCODE DC    XL1'01'        Transmission code ('01'=SEND/'02'=RECV)
*                               Record identifier ('C0') fixed by CP
*           ACNTLEN EQU    *-ACNTBUFF Account work area length
END

```

PI end

AUTHBLOK

The AUTHBLOK (authorization table) lists users who are authorized to act as an alternate RSCS operator or as a link operator. DMTIRX builds this table when RSCS is initialized. Each AUTHBLOK is chained to the next; the end of the chain indicated by a word of zeros in the AUTHNEXT field. See [“AUTHBLOK” on page 32](#) for more information.

```

X'000' AUTHNEXT DC A(0)      Address of next authorization table entry
X'004' AUTHFLG1 DC XL1'00'   Authorization flags

*           Bits defined in AUTHFLG1:

AUTHCP     EQU X'80'         User authorizes for CP command

X'005' AUTHORGQ DC XL1'00'   Authorized node qualifier
X'006' AUTHRESV DC XL2'00'   Reserved
X'008' AUTHNODE DC CL8' '    Authorized node ID
X'010' AUTHUSER DC CL8' '    Authorized user ID
X'018' AUTHLINK DC CL8' '    Authorized target link
AUTHLEN     EQU *-AUTHBLOK

```

Printer-Related Structures

This section describes the data areas RSCS uses when processing print files.

FORM

The FORM table describes the characteristics of a print form. DMTIRX builds the table when RSCS initializes. Each entry is chained to the next; the end of the chain is indicated by a fullword of zeros in the FORMNEXT field. The TFORMTAB field in the CVT contains the address of the first form table entry.

```

X'000' FORMNEXT DC A          Address of next form table entry
X'004' FORMWOTH DC F'0'       Width of form in spaces
X'008' FORMLNTH DC F'0'       Length of form in lines
X'00C' FORMLPI  DC F'0'       Lines per inch of form
X'010' FORMFLAG DC XL1'00'    Flag byte

*
*           Bits defined in FORMFLAG
*
FORMVM      EQU X'80'         Generate VM-style separator
FORMSHRT    EQU X'40'         Generate short-style separator
FORMNOSP    EQU X'20'         Generate no separator

*
X'011' FORMRESV DC XL3'00'    Reserved
X'014' FORMNAME DC CL8' '     User name of form
FORMLEN     EQU *-&LABEL      Form table entry length

```

RFCBTAB

The RFCBTAB contains printer form information specified on FCB statements. See [“FCB Table” on page 32](#) for more information.

```

X'000' RFCBNEXT DC A(0)      Address of next FCB table entry
X'004' RFCBNAME DC CL4' '    Name of the FCB image
X'008' RFCBLICH DC CL256' '  Array of line/channel pairs
RFCBEND     DS 0H            Mark the end of the structure
RFCBLEN     EQU *-&LABEL      FCB table entry length

```

SEPBLOK

SEPBLOK contains input parameters and work areas used by DMTSEP when it produces a separator page for a print file.

```

X'000' SEPLINK    DC A(0)      Link table pointer
X'004' SEPRDEV    DC A(0)      RDEVBLK pointer
X'008' SEPTAG     DC A(0)      TAG pointer
X'00C' SEPWORK    DC A(0)      Pointer to workarea
X'010' SEPUSER    DC D'0'      Pointer to user workarea
X'018' SEPFLAG    DC XL1'00'   Separator flag

*           Bits defined in SEPFLAG
SEPBEOF    EQU X'80'          Page eject before header
SEPAFTER   EQU X'40'          Page eject after header
SEPPUNCH   EQU X'20'          Treat punch headers differently
SEPLEN     EQU *-SEPBLOK      Length of request block

```

File Queueing Structures

The following sections describe the data areas RSCS uses to process files. See [“Processing Files” on page 21](#) for more information.

SAFTAG

The SAFTAG describes the tag element for store-and-forward files.

X'000'	SAFFLAG	DS	4C	Store and forward indicator
X'004'	SAFTOLOC	DS	CL8	Destination location ID
X'00C'		DS	CL1	Space
X'00D'	SAFTOVM	DS	CL8	Destination VM userid
X'015'		DS	CL1	Space
X'016'	SAFPRIOR	DS	CL2	Transmission priority
X'018'		DS	CL1	Space
X'019'	SAFINLOC	DS	CL8	Originating location ID
X'021'		DS	CL1	Space
X'022'	SAFINVM	DS	CL8	Originating VM userid
X'02A'		DS	CL1	Space
X'02B'	SAFINTOD	DS	CL16	Originating TOD
X'03B'		DS	CL1	Space
X'03C'	SAFORGID	DS	CL4	Originating spoolid
X'040'		DS	CL1	Space
X'041'	SAFCNTRL	DS	CL4	Tag control record format
X'045'		DS	CL1	Space
X'046'	SAFFORMN	DS	CL8	Tag form name
X'04E'		DS	CL1	Space
X'04F'	SAFKEY	DS	CL6	NJE key (hex)
X'055'		DS	CL1	Space
X'056'	SAFFLAG2	DS	CL2	Flag byte (hex)
X'058'		DS	CL1	Space
X'059'	SAFRECNM	DS	CL8	Logical record count (hex)
X'061'		DS	CL1	Space
X'062'	SAFFLAG3	DS	CL2	Flag byte (hex)
X'064'	SAFJULN1	DS	CL1	Nibble 1 of day in epoch
X'065'	SAFDSHNO	DS	CL4	Number of dshs processed
X'069'	SAFJULN2	DS	CL1	Nibble 2 of day in epoch
X'06A'	SAFRECD5	DS	CL8	Record count for dataset
X'072'	SAFFLAGU	DS	CL2	Flag for user exits kept in TAGFLAGU
X'074'	SAFINNOD	DS	CL8	Previous node for file
X'07C'	SAFORLOC	DS	CL8	Override 'to' location
X'084'	SAFFLAG4	DS	CL2	Flag byte (hex)
X'086'	SAFJULN3	DS	CL2	Last byte of day in epoch
* * Alternate definition of SAFTAG for local retagged files: *				
	ORG	SAFFLAG		Go back to the start
X'000'	SAFRR	DS	CL1	C'r' marker
X'001'	SAFRHEX	DS	CL2	Local flag byte
X'003'		DS	CL1	Space
X'004'	SAFRJULN	DS	CL4	Epoch day
X'008'		DS	CL1	Space
X'009'	SAFRTLLOC	DS	CL8	Destination location id
X'011'		DS	CL1	Space
X'012'	SAFRTOVM	DS	CL8	Destination VM userid
X'01A'		DS	CL1	Space
X'01B'	SAFRPRIO	DS	CL2	Transmission priority
X'01D'		DS	CL1	Space
X'01E'	SAFRUSER	DS	C	Start of user tag text

TAG

A TAG element describes each file enqueued for processing by RSCS. See [“TAG Element” on page 23](#) for more information.

X'000'	TAGNEXT	DC	A(0)	Addr of next TAG slot entry
X'004'	TAGPREV	DC	A(0)	Addr of previous TAG slot entry
* The following offset and bit map MUST always remain in this order				
* in contiguous bytes.				
* * TAFOFFAL DC H'0' Offset into allocation map				
X'008'	TAFOFFAL	DC	H'0'	Offset into allocation map
X'00A'	TAGBITMP	DC	X'00'	Bitmap into the allocation map
* * TAGFLAGU DC X'00' Flag byte for user exits kept in SAFFLAGU				
X'00B'	TAGFLAGU	DC	X'00'	Flag byte for user exits kept in SAFFLAGU
X'00C'	TAGUSER	DC	XL8'00'	Doubleword for user usage
* * The information after this point is contained in TAGDATA and will be				
* overwritten with a prototype when the TAG slot is reassigned to a new file				
* * TAGBLOCK DC A(0) Address of associated I/O area				
X'014'	TAGBLOCK	DC	A(0)	Address of associated I/O area
	TAGINADR	DS	0CL16	Originating network address
X'018'	TAGINLOC	DC	CL8' '	Originating location
X'020'	TAGINVM	DC	CL8' '	Originating userid (SFBORIG)
X'028'	TAGINNOD	DC	CL8' '	Previous node file was on
X'030'	TAGLINK	DC	CL8' '	File active on this link
X'038'	TAGINTOD	DC	CL8' '	Time of file origin
X'040'	TAGRECNM	DC	0F'0',X'80000000'	Number of records in file (SFBRECN0)

X'044'	TAGRECLN	DC	H'0'	Max possible rec length (SFBRECSZ)
X'046'	TAGINDEV	DC	X'00'	Device code of orig dev (SFBTYPE)
X'047'	TAGCLASS	DC	C'	File output class (SFBCLAS)
X'048'	TAGID	DC	H'0'	Current spool file ID (SFBFILID)
X'04A'	TAGCOPY	DC	H'1'	Number of copies requested (SFBCOPY)
* Fullword status flag to allow for easy message and filter processing				
* Fullword flags				
X'04C'	TAGFFLG	DS	0F	Fullword flags
X'04D'	TAGFLAG	DC	X'00'	Same bits as SFBFLAG
X'04D'	TAGSFLAG	DC	X'00'	Same bits as TASFLAG in TASHADOW
X'04E'	TAGFLAG9	DC	X'00'	Another flag byte (SAFFLAG4)
* Bits defined in TAGFLAG9				
* Special hold (used with NOTIFY)				
	TAGSPHLD	EQU	X'80'	Special hold (used with NOTIFY)
	TAGNBURN	EQU	X'40'	File to be purged (used with NOTIFY)
	TAGNNEWJ	EQU	X'20'	Give file a new day number (used with NOTIFY)
	TAGHEX	EQU	X'10'	SAFORGID field is in hex
	TAGASYNC	EQU	X'08'	File is being read via *SPL
X'04F'	TAGFLAG8	DC	X'00'	TAG flag byte number 8 ...
* Bits defined in TAGFLAG8				
* Maximum hop count reached				
	TAGLOOPH	EQU	X'80'	Maximum hop count reached
	TAGHSALT	EQU	X'40'	File has alternate routing shadows
	TAGRXING	EQU	X'20'	File is being received
	TAGTXING	EQU	X'10'	File is being sent
	TAGATRAC	EQU	X'08'	This is an active trace file
	TAGTLOPI	EQU	X'04'	Temporary immed loop for fan-out
	TAGDS2ND	EQU	X'02'	2nd dataset found in file
* Bits defined in above four flag bytes, fullword-aligned				
* User hold				
	TAG4UHLD	EQU	SFBUHOLD*16777216	User hold
* System hold				
	TAG4SHLD	EQU	SFBSHOLD*16777216	System hold
* Special held				
	TAG4SPHL	EQU	TAGSPHLD*256	Special held
* Receiving				
	TAG4RX	EQU	TAGRXING	Receiving
* Sending				
	TAG4TX	EQU	TAGTXING	Sending
* Tracing				
	TAG4TRAC	EQU	TAGATRAC	Tracing
X'050'	TAGFLAG2	DC	X'00'	VM SFBLOK flag (SFBFLAG2)
	TAGREQUE	EQU	X'20'	Indicates file has been requeued
X'051'	TAGFLAG3	DC	X'00'	3800 SPLINK flag (SFBFLAG1)
X'052'	TAGFLAG4	DC	X'00'	VM SFBLOK flag (SFBFLAG3)
X'053'	TAGFLAG6	DC	X'00'	VM SFBLOK flag (SFBFLAG4)
X'054'	TAGFLAG5	DC	X'00'	TAG flag byte (SAFFLAG2)
* Bits defined in TAGFLAG5				
* Transfer or change done				
	TAGXFERD	EQU	X'80'	Transfer or change done
* Cannot transfer or change				
	TAGFREEZ	EQU	X'40'	Cannot transfer or change
* No 3800 section				
	TAGN3800	EQU	X'20'	No 3800 section
* File contains CPDS records				
	TAGCPDS	EQU	X'10'	File contains CPDS records
* File is store-and-forward				
	TAGSAF	EQU	X'08'	File is store-and-forward
* File has been rerouted				
	TAGREROD	EQU	X'04'	File has been rerouted
* File has been split/spun				
	TAGSPLIT	EQU	X'02'	File has been split/spun
* File is MAXURO overflow				
	TAGOVFLW	EQU	X'01'	File is MAXURO overflow
X'055'	TAGFLAG7	DC	X'00'	TAG flag byte (SAFFLAG3)
* Bits defined in TAGFLAG7				
* SENT message to origin				
	TAGCOF	EQU	X'80'	SENT message to origin
* FINAL message to origin				
	TAGCOA	EQU	X'40'	FINAL message to origin
* Enqueued message to origin				
	TAGENQM	EQU	X'20'	Enqueued message to origin
* Accept message to origin				
	TAGACCM	EQU	X'10'	Accept message to origin
* Locally created list file				
	TAGLISTL	EQU	X'08'	Locally created list file
* List processor created file				
	TAGLISTP	EQU	X'04'	List processor created file
* File hop count maximum exceeded				
	TAGDLOPH	EQU	X'02'	File hop count maximum exceeded
* Looping message issued to origin user				
	TAGLOOPM	EQU	X'01'	Looping message issued to origin user
X'056'	TAGNTJUL	DC	AL2(0)	Notify day number in epoch (unsigned - set for files on NOTIFY)
X'058'	TAGLCTOD	DC	F'0'	Top 32 bits of local time origin for sorting FIFO queues

TAGAREA

X'05C'	TAGORGID	DC	H'0'	VM spoolid at origin location
X'05E'	TAGPRIOR	DC	H'50'	Transmission priority
	TAGDSN	DS	0CL24	File name/file type, dataset name
X'060'	TAGNAME	DC	CL12'	File name (SFBFNAME)
	TAGNAME8	EQU	TAGNAME,8	Define equate for filters
X'06C'	TAGTYPE	DC	CL12'	File type (SFBFTYPE)
	TAGTYPE8	EQU	TAGTYPE,8	Define equate for filters
X'078'	TAGDIST	DC	CL8'	File distribution code (SFBDIST)
X'080'	TAGORLOC	DC	CL8'	Override 'T0' location
X'088'	TAGTOADR	DS	0CL16	Destination network address
X'088'	TAGTOLOC	DC	CL8'	Destination location ID
X'090'	TAGTOVM	DC	CL8'	Destination virtual machine ID
X'098'	TAGDEST	DC	CL8'	PSF destination (SFBDEST)
X'0A0'	TAGDEV	DC	AL2(0)	Active file's virtual dev address
X'0A2'	TAGWORK1	DC	C'	Work Area
X'0A3'	TAGFMQUL	DC	X'00'	From node qualifier
X'0A4'	TAGCNTRL	DC	CL4'	Network control record format
X'0A8'	TAGRECDN	DC	F'0'	Number of records done (xmit & rcv)
X'0AC'	TAGWORK	DS	F	Incoming NJE file record count accumulator (NHD)

*
 * The following five fields shall be kept together to ease
 * the process of initializing them.
 *

X'0B0'	TAGFORMN	DC	CL8'	User form name (SFBUFORM)
X'0B8'	TAGFLSHN	DC	CL4'	3800 flash name (SFBFLASH)
X'0BC'	TAGMODN	DC	CL4'	3800 copy mod name (SPCMOD)
X'0C0'	TAGCHARN	DC	CL4'	3800 CHARs name 0 (SPCHAR)
X'0C4'	TAGFCBN	DC	CL4'	3800 FCB name (SPFCB)
	TAG38NML	EQU	*-TAGFORMN	Length of this section
X'0C8'	TAGFLC	DC	X'00'	3800 FLASH count (SPFLSHC)
X'0C9'	TAGKEY	DC	XL3'00'	Pseudo-random number for NJE
	TAGCHARX	DS	0CL12	3800 CHARs names 1-3
X'0CC'	TAGCHAR1	DC	CL4'	3800 CHARs name 1 (SPCHAR1)
X'0D0'	TAGCHAR2	DC	CL4'	3800 CHARs name 2 (SPCHAR2)
X'0D4'	TAGCHAR3	DC	CL4'	3800 CHARs name 3 (SPCHAR3)
X'0D8'	TAGFORMO	DC	CL8'	Operator form name (SFBFORM)
X'0E0'	TAGMODTR	DC	C'	3800 MODIFY TRC (SPCMCHR)
X'0E1'	TAGPGLEN	DC	X'00'	Virtual 3800 page length (SPPGLEN)
X'0E2'	TAGFCBNL	DC	H'0'	Maximum FCB length (SFBFCBNL)
X'0E4'	TAGFCBXL	DC	H'0'	Maximum extend FCB length (SFBFCBXL)
X'0E6'	TAGXABL	DC	H'0'	XAB length (SFBXABL)
X'0E8'	TAGRCMAX	DC	H'0'	Longest trunc rec (SPRECMAX)
X'0EA'	TAGDSHNO	DC	H'0'	Number of DSHs sent
X'0EC'	TAGSPLNM	DC	0F'0',X'80000000'	Number of SPLINKS (SPSPLNKC)
X'0F0'	TAGOPTOD	DC	D'0'	Open time-of-day for file
X'0F8'	TAGSPLDN	DC	F'0'	Number of SPLINKS done
X'0FC'	TAGRECDS	DC	F'0'	Rec number for *LIST dsh
X'100'	TAGSHPTR	DC	A(0)	Anchor for shadow elements for this TAG slot

X'104'	TAGSTRID	DC	X'00'	Stream ID for this TAG slot
X'105'		DS	3X	Round out to a double word
X'108'	TAGPRLNK	DC	CL8'	Preferred print link (multi-copy)
X'110'	TAGJOBID	DC	H'0'	Origin job number
X'112'		DC	H'0',F'0'	Reserved/dbl wrd bdry
X'118'	TAGXWRT	DC	CL8'	External writer name
	TAGTLEN	EQU	*-TAGBLOCK	Length of the TAG w/o pointer area
	TAGLEN	EQU	*-TAG	Length of the TAG slot
	TAGDATA	EQU	TAGBLOCK,TAGTLEN	TAG slot data area alias with length

TAGAREA

The Tag Queue Area (TAGAREA) contains data about the active TAG queue, pointers, and other tag control information. For more information, see [“TAGAREA” on page 21](#).

X'000'	TAGACIN	DC	A(0)	Active input queue
X'004'	TAGACOUT	DC	A(0)	Active output queue
X'008'	TAGATSTO	DC	A(0)	Pointer to TASTORAG for TAG slots
X'00C'	TAGASSTO	DC	A(0)	Pointer to TASTORAG for TASHADOWs
X'010'	TAGASVEC	DC	A(0)	Address of 10000 word spool ID vector
X'014'	TAGASHCN	DC	A(0)	Address of 10000 byte vector that ... maintains shadow use by spids
X'018'	TAGASHND	DC	A(0)	Address of 10000 byte vector that ... maintains shadow needs by spids
X'01C'	TAGASLOT	DC	A(0)	Anchor for the TAG slots (global)
X'020'	TAGASMAX	DC	AL1(255)	Maximum number of shadows
X'021'		DC	X'00'	Spare byte
X'022'	TAGASCNT	DC	H'0'	Skip count for degraded message
X'024'	TAGASKIP	EQU	100	Issue degraded msg every 100 times
X'024'	TAGAFRSH	DC	F'0'	Count of free shadows (kept in ... degraded mode only)
*	TAGAFRMX	EQU	1000	Here's where we get generous
	TAGAFRMN	EQU	500	Here's where we get stingy
X'028'	TAGASLVE	DC	A(0)	Anchor for the slowdown entry vector
X'02C'	TAGASLVX	DC	A(0)	Anchor for the slowdown exit vector
X'030'	TAGANUMU	DC	A(0)	Number of TAG slots in use
X'034'	TAGASLEB	DC	H'0'	Base entry to slowdown
X'036'	TAGASLEX	DC	H'0'	Base exit to slowdown
	TAGALEN	EQU	*-LABEL	TAG area length

TASHADOW

A TAG shadow element (TASHADOW) represents an inactive file on each link that can send the file. For more information, see [“TASHADOW” on page 23](#).

X'000'	TASNEXT	DC	A(0)	Address of next shadow element
X'004'	TASPREV	DC	A(0)	Address of previous shadow element
	* The following two pointers chain all TASHADOW elements for a			
	* given spoolid together (the queue is anchored in the TAG slot).			
X'008'	TASPLNXT	DC	A(0)	Address of next shadow element
X'00C'	TASPLPRV	DC	A(0)	Address of previous shadow element
X'010'	TASLKPTR	DC	A(0)	Pointer to owning LINKTABL
X'014'	TASSPID	DC	H'0'	Spoolid represented by this element
X'016'	TASFLAG	DC	X'00'	Flag byte for the element
	* Bits defined in TASFLAG (also used in TAGSFLAG in TAG)			
	* TASL00PH EQU X'80' File in hop count loop			
	* TASL00PI EQU X'40' Shadow element in immediate loop			
	* TASORDER EQU X'20' This shadow element has been ORDERed			
	* TASNOSTR EQU X'10' File is not eligible for any stream			
	* TASEXHL D EQU X'08' File held by user exit			
	* TASMCOPY EQU X'04' Non-first copy of multicopy file			
	* Note: As TAGSFLAG is the second byte in TAGFFFLG, 4 byte			
	* equates are defined by multiplying by 2**8			
	* TAS4L0PH EQU TASL00PH*65536 Set up for TAGFFFLG definition			
	* TAS4L0PI EQU TASL00PI*65536 Set up for TAGFFFLG definition			
	* TAS4ORD EQU TASORDER*65536 Set up for TAGFFFLG definition			
	* TAS4NSTR EQU TASNOSTR*65536 Set up for TAGFFFLG definition			
	* TAS4EHL D EQU TASEXHL D*65536 Set up for TAGFFFLG definition			
X'017'	TASFLAG2	DC	X'00'	Flag byte #2 for the element
	* Bits defined in TASFLAG2			
	* TASHOLD EQU X'80' File is in HOLD state			
	* TASALTER EQU X'40' This is an alternate routing shadow			
	* TASALTNS EQU X'20' Alternate routing shadow not			
	* TASPULL EQU X'10' Shadow will be pulled (CHANGE/XFER)			
	* TASSPHLD EQU X'08' eligible for transmission			
	* TASSPHLD EQU X'08' File is in special HOLD.			
	* TASSPHLD EQU X'08' Used by notify link driver.			
X'018'	TASSORT	DC	XL8'00'	Queue order based on this field
	TASSORTV	EQU	TASSORT,4	Primary value used for queueing
	TASSORTB	EQU	TASSORT+4,4	Queueing tie-breaker
	* The following offset and bitmap MUST remain in this order in			
	* contiguous bytes and MUST be after TASSORTB.			
	* TASOFFAL DC H'0' Offset to byte into bit map			
X'020'	TASOFFAL	DC	H'0'	Offset to byte into bit map
X'022'	TASBTMAP	DC	X'00'	Bit mask in byte in the bit map
X'023'	TASCLASS	DC	C' '	Class for the file being sent
X'024'	TASSTRID	DC	XL4'00000000'	Stream file is eligible for (NJE)
	TASLEN	EQU	*-&LABEL	Length of TASHADOW element

TASTORAG

The TASTORAG area is used to manage storage needed for TAG and TASHADOW elements. See [“TASTORAG” on page 22](#) for more information.

PRDBLOK

X'000'	TASTIPPG	DC	H'0'	Number of items per page
X'002'	TASTPPCH	DC	H'0'	Number of pages per chunk
X'004'	TASTIPCH	DC	H'0'	Number of items per chunk
X'006'	TASTILEN	DS	H'0'	Length of each item
X'008'	TASTBPTR	DC	A(0)	Address of 'origin' byte in ... the allocation map
*				Length of allocation map after ... the 'origin' byte
X'00C'	TASTBLEN	DC	F'0'	Eyecatcher to use at top of page
*				Eyecatcher to use with free items
X'010'	TASTEYEC	DC	CL8' '	Pointer to bitmap for these items
X'018'	TASTFEYE	DC	CL8' '	Total length of bit map (bytes)
X'020'	TASTBITM	DC	A(0)	Pointer to vector of chunk addresses
X'024'	TASTTLEN	DC	F'0'	Offset from beginning of item to hword ... bitmap offset and bit pattern
X'028'	TASTCVEC	DC	A(0)	Offset from beginning of item to ... eyecatcher
X'02C'	TASTO0FF	DC	H'0'	Number of items allocated
*				Number of chunks allocated
X'02E'	TASTE0FF	DC	H'0'	Offset from beginning of item to ... eyecatcher
*				Number of items allocated
X'030'	TASTIALL	DC	F'0'	Number of chunks allocated
X'034'	TASTCALL	DC	F'0'	Number of chunks allocated
*	TASTLEN	EQU	*-&LABEL	TAG area length

TCP/IP-Related Structures

This section describes the control blocks used by the TCPNJE-type link drivers. It also describes the DSECTs created by the SOCKET macro.

PRDBLOK

A PRDBLOK (TCP/IP port redirector block) is built and sent to the port redirector task, DMTPRD, for each host/port pair that a TCPNJE link driver task wants to listen for. It is also used to cancel these listen requests. DMTPRD sends a PRDBLOK to the TCPNJE link driver task when an incoming connect request matches or when an error prevents DMTPRD from listening for a request. See [“Port Redirector Task” on page 60](#) for more information.

X'000'	PRDLENG	DC	XL1'00'	Length of total element - 1
X'001'	PRDTYPE	DC	XL1'00'	Function code
X'002'	PRDFLAG	DC	XL1'00'	Flags
X'003'		DC	XL1'00'	Reserved
*				
	* Values defined in PRDTYPE			
	PRDADD	EQU	1	Add request
	PRDDEL	EQU	2	Delete request
X'000'	PRDERR	EQU	3	Error reply
	PRDGIVE	EQU	4	Giving socket
*				
	* Bits defined in PRDFLAG			
	PRDSOKA	EQU	X'80'	This block owns a socket
	PRDESENT	EQU	X'40'	We've sent an error msg
X'004'	PRDNEXT	DC	A(0)	Address of next PRDBLOK
X'008'	PRDSOCKN	DC	F'0'	Socket number
X'00C'	PRDERRNO	DC	F'0'	Error number
X'010'	PRDTCPID	DC	CL8' '	TCP ID
X'018'	PRDTASK	DC	CL8' '	RSCS task ID
X'020'	PRDSOCKA	DC	XL16'00'	SOCKADDR structure
	PRD_FAM	EQU	PRDSOCKA,2	Addressing family
	PRD_PORT	EQU	PRDSOCKA+2,2	TCP Port number
	PRD_ADDR	EQU	PRDSOCKA+4,4	IP address
X'030'	PRDCLIEN	DC	XL40'00'	CLIENTID structure
	PRD_DOM	EQU	PRDCLIEN,4	Domain
	PRD_NAME	EQU	PRDCLIEN+4,8	User name
	PRD_TASK	EQU	PRDCLIEN+12,8	Subtask name
X'058'	PRDSMASK	DS	8X	Select mask
		DS	00	
	PRDLEN	EQU	*-&LABEL	Length of data area

SOCKBLOK

The SOCKBLOK (socket set descriptor block) maps a control that is used by DMTSOK. There is one SOCKBLOK for each socket.

```

*-----
* This area MUST remain at the head of the macro ... it is necessary
* to place it here 'cause GCS STIMER macro has no UWORD support.
*-----
X'000' SOCKTIME EQU *
NI          *      SOCKFLAG-SOCKTIME(R15),X'FF'-SOCKTIMR
            *      Say timer no longer set
            *      L      R1,SOCKTECB-SOCKTIME(0,R15)
            *      Get address of ECB to post
            *      L      R0,SOCKCODE-SOCKTIME(0,R15)
            *      Get ECB completion code
            *      POST (R1),(0)
            *      BR      R14      Post the timer ecb
            *      And return to GCS
*-----
* Here beginneth the actual data
*-----
X'014' SOCKTECB DC A(0)      Address of ECB to post from
            *      ... timer routine
X'018' SOCKCODE DC F'0'      Code to post timer ECB with

X'01C' SOCKMAXD DC H'0'      Maximum socket descriptors
X'01E' SOCKPATH DC H'0'      IUCV pathid used by link
X'020' SOCKVMID DC CL8'      VMID I want to talk to
X'028' SOCKTASK DC CL8'      Task ID

X'030' SOCKALST DC A(0)      List of active call blocks

X'034' SOCKFLAG DC X'00'      Flag for use with IUCV stuff
            *
            *      Bits defined in SOCKFLAG
            *
            *      SOCKDECL EQU X'80'      IUCVINI has been issued
            *      SOCKOINI EQU X'40'      Outstanding initialize request
            *      SOCKCON  EQU X'20'      Path is connect
            *      SOCKTIMR EQU X'10'      Timer is set

X'035' SOCKKEY  DC X'E0'      Mainline storage access key
X'036' SOCKMAXC DS H          Maximum active socket calls
X'038' SOCKMAP  DS XL256      Bitmap of sockets in use
X'138' SOCKCLST RCALL ,,,,,,MF=L Space for RCALL plists
            DS      0D
            SOCKLEN EQU *-SOCKBLOK      Length of data area

```

SOCKCBLK

SOCKCBLK (active socket call block) is used by DMTSOK to handle active socket calls. There is one SOCKCBLK for each concurrent socket call. The *maxcall* parameter of the SOCKET INITIALIZE function determines the maximum number of calls. See [z/VM: RSCS Networking Exit Customization](#) for more information about the SOCKET macro parameters.

```

***          SOCKCBLK - Active socket call block

X'000' SOCKCFWD DC A(0)      Pointer to next SOCKCBLK

X'004' SOCKICOD DC A(0)      Code to call at interrupt

X'008' SOCKIPMS DS XL(IPSIZE*8) Space to build an IPARML
X'030' SOCKIPMI DS XL(IPSIZE*8) Space for interrupt IPARML

X'058' SOCKBUF1 DS 9D      Buffer/list for IPPBADR1
            ORG      SOCKBUF1
            SOCKB1A1 DS A      Address 1
            SOCKB1L1 DS F      Length 1
            SOCKB1A2 DS A      Address 2
            SOCKB1L2 DS F      Length 2
            SOCKB1A3 DS A      Address 3
            SOCKB1L3 DS F      Length 3
            SOCKB1A4 DS A      Address 4
            SOCKB1L4 DS F      Length 4
            SOCKB1A5 DS A      Address 5
            SOCKB1L5 DS F      Length 5
            SOCKB1A6 DS A      Address 6
            SOCKB1L6 DS F      Length 6

```

SOCKET

	SOCKB1A7 DS	A	Address 7
	SOCKB1L7 DS	F	Length 7
	SOCKB1A8 DS	A	Address 8
	SOCKB1L8 DS	F	Length 8
	SOCKB1A9 DS	A	Address 9
	SOCKB1L9 DS	F	Length 9
	ORG	/	
X'0A0'	SOCKBUF2 DS	9D	Buffer/list for IPBFADR2
	ORG	SOCKBUF2	
	SOCKB2A1 DS	A	Address 1
	SOCKB2L1 DS	F	Length 1
	SOCKB2A2 DS	A	Address 2
	SOCKB2L2 DS	F	Length 2
	SOCKB2A3 DS	A	Address 3
	SOCKB2L3 DS	F	Length 3
	SOCKB2A4 DS	A	Address 4
	SOCKB2L4 DS	F	Length 4
	SOCKB2A5 DS	A	Address 5
	SOCKB2L5 DS	F	Length 5
	SOCKB2A6 DS	A	Address 6
	SOCKB2L6 DS	F	Length 6
	SOCKB2A7 DS	A	Address 7
	SOCKB2L7 DS	F	Length 7
	SOCKB2A8 DS	A	Address 8
	SOCKB2L8 DS	F	Length 8
	SOCKB2A9 DS	A	Address 9
	SOCKB2L9 DS	F	Length 9
	ORG	/	
X'0E8'	SOCKZRC DS	F	Place for RC when null
X'0EC'	SOCKZERN DS	F	Place for ERRNO when null
	ORG	/	
X'0F0'	SOCKPRMS DS	0A	Copy of parameter list
	SOCKFCDE DS	F	Function code
	SOCKBLKA DS	A	Copy of address of SOCKBLOK
	SOCKECB DS	A	Address of ECB to post
	SOCKRC DS	A	Address of RC
	SOCKERRN DS	A	Address of errno
	SOCKCID DS	A	Address of Call ID
	SOCKPM1 DS	A	Address of parm 1
	SOCKPM2 DS	A	Address of parm 2
	SOCKPM3 DS	A	Address of parm 3
	SOCKPM4 DS	A	Address of parm 4
	SOCKPM5 DS	A	Address of parm 5
	SOCKPM6 DS	A	Address of parm 6
	ORG	/	
X'120'	SOCKIUCL IUCVCOM ,MF=L		Space for IUCVCOM plist
	DS	0D	
	SOCKCLEN EQU	*-SOCKCBK	Length of data area

SOCKET

The following equates are generated when the DSECT keyword is specified on the SOCKET macro.

DMTSOCKET function call number equates			
	Socket Macro Function #		IUCV Socket Function #
SOKINIT	EQU 0	Initialize	-
SOKTERM	EQU 1	Terminate	-
SOKSOCK	EQU 2	Socket	25
SOKCONN	EQU 3	Connect	4
SOKIOCTL	EQU 4	IOCTL	12
SOKSSOKO	EQU 5	SetSockOpt	23
SOKSELEC	EQU 6	Select	19
SOKRECV	EQU 7	Recv	16
SOKRECVF	EQU 8	Recvfrom	16
SOKSEND	EQU 9	Send	20
SOKSHUT	EQU 10	Shutdown	24
SOKCLOSE	EQU 11	Close	3
SOKACCEP	EQU 12	Accept	1
SOKBIND	EQU 13	Bind	2
SOKFCNTL	EQU 14	Fcntl	5
SOKGETCL	EQU 15	GetClientID	30
SOKGETHB	EQU 28	GetHostByName	-
SOKGETHI	EQU 16	GetHostID	7
SOKGETHN	EQU 17	GetHostName	8
SOKGETPN	EQU 18	GetPeerName	9
SOKGETSN	EQU 19	GetSockName	10
SOKGETSO	EQU 20	GetSockOpt	11
SOKGIVES	EQU 21	GiveSocket	31
SOKLISTE	EQU 22	Listen	13
SOKREAD	EQU 23	Read	14
SOKSENDT	EQU 24	SendTo	22
SOKTAKES	EQU 25	TakeSocket	32
SOKWRITE	EQU 26	Write	26
SOKCANCE	EQU 27	Cancel	42

Socket types

SOCK_STR EQU	1	stream socket
SOCK_DGM EQU	2	datagram socket
SOCK_RAW EQU	3	raw-protocol interface
SOCK_RDM EQU	4	reliably-delivered message
SOCK_SQP EQU	5	sequenced packet stream

Option flags per-socket

SO_DEBUG EQU	X'0001'	turn on debugging info recording
SO_ACCEP EQU	X'0002'	socket has had listen
SO_REUSE EQU	X'0004'	allow local address reuse
SO_KEEPA EQU	X'0008'	keep connections alive
SO_DONTR EQU	X'0010'	just use interface addresses
SO_BROAD EQU	X'0020'	permit sending of broadcast msgs
SO_USELO EQU	X'0040'	bypass hardware when possible
SO_LING EQU	X'0080'	linger on close if data present
SO_OOBIN EQU	X'0100'	leave received OOB data in line

Additional options, not kept in so_options

SO_SNDBF EQU	X'1001'	send buffer size
SO_RCVBF EQU	X'1002'	receive buffer size
SO_SNDLO EQU	X'1003'	send low-water mark
SO_RCVLO EQU	X'1004'	receive low-water mark
SO_SNDTI EQU	X'1005'	send timeout
SO_RCVTI EQU	X'1006'	receive timeout
SO_ERROR EQU	X'1007'	get error status and clear
SO_TYPE EQU	X'1008'	get socket type

Level number for Get/Set SockOpt

SOL_SOCKET EQU X'FFFF'

Address families

AF_UNSPC EQU	0	unspecified
AF_UNIX EQU	1	local to host (pipes, portals)
AF_INET EQU	2	internetwork: UDP, TCP, etc.
AF_IMPLI EQU	3	arpanet imp addresses
AF_PUP EQU	4	pup protocols: e.g. BSP
AF_CHAOS EQU	5	mit CHAOS protocols
AF_NS EQU	6	XEROX NS protocols
AF_NBS EQU	7	nbs protocols
AF_ECMA EQU	8	european computer manufacturers
AF_DATAK EQU	9	datakit protocols
AF_CCITT EQU	10	CCITT protocols, etc
AF_SNA EQU	11	IBM SNA
AF_DECNE EQU	12	DECnet
AF_DLI EQU	13	Direct data link interface
AF_LAT EQU	14	LAT
AF_HYLIN EQU	15	NSC Hyperchannel
AF_APPLE EQU	16	Apple Talk
AF_IUCV EQU	17	IBM IUCV

Protocol families, same as address families for now.

PF_UNSPC EQU	AF_UNSPC	unspecified
PF_UNIX EQU	AF_UNIX	local to host (pipes, portals)
PF_INET EQU	AF_INET	internetwork: UDP, TCP, etc.
PF_IMPLI EQU	AF_IMPLI	arpanet imp addresses
PF_PUP EQU	AF_PUP	pup protocols: e.g. BSP
PF_CHAOS EQU	AF_CHAOS	mit CHAOS protocols
PF_NS EQU	AF_NS	XEROX NS protocols
PF_NBS EQU	AF_NBS	nbs protocols
PF_ECMA EQU	AF_ECMA	european computer manufacturers
PF_DATAK EQU	AF_DATAK	datakit protocols
PF_CCITT EQU	AF_CCITT	CCITT protocols, etc
PF_SNA EQU	AF_SNA	IBM SNA
PF_DECNE EQU	AF_DECNE	DECnet
PF_DLI EQU	AF_DLI	Direct data link interface
PF_LAT EQU	AF_LAT	LAT
PF_HYLIN EQU	AF_HYLIN	NSC Hyperchannel
PF_APPLE EQU	AF_APPLE	Apple Talk
PF_IUCV EQU	AF_IUCV	IBM IUCV

Flags for Send and Recv

MSG_OOB EQU	1	process out-of-band data
MSG_PEEK EQU	2	peek at incoming message

MSG_DONT EQU 4 send without using routing tables

Request equates for IOCTL

FIONBIO	EQU	X'8004A77E'	Set/clear non-blocking I/O
FIONREAD	EQU	X'4004A77F'	Get # of bytes to read
SIOCADDR	EQU	X'8030A70A'	Add route
SIOCATMA	EQU	X'4004A707'	At 00B mark?
SIOCDELRL	EQU	X'8030A70B'	Delete route
SIOCGIFA	EQU	X'C020A70D'	Get ifnet address
SIOCGIFB	EQU	X'C020A712'	Get broadcast address
SIOCGIFC	EQU	X'C008A714'	Get ifnet list
SIOCGIFD	EQU	X'C020A70F'	Get p-p address
SIOCGIFF	EQU	X'C020A711'	Get ifnet flags
SIOCGIFM	EQU	X'C020A717'	Get IF metric
SIOCGIFN	EQU	X'C020A715'	Get net address mask
SIOCSIFD	EQU	X'8020A70E'	Set p-p address
SIOCSIFF	EQU	X'8020A710'	Set ifnet flags
SIOCSIFM	EQU	X'8020A718'	Set IF metric

Request equates for FCNTL

F_GETFL	EQU	3	Get file flags
F_SETFL	EQU	4	Set file flags

Flags for F_GETFL and F_SETFL

FNDELAY EQU X'00000004' Non-blocking reads

SOCKADDR structure

SOCKADDR DSECT			
SIN_FAM	DS	H	Addressing family
SIN_PORT	DS	H	Port number
SIN_ADDR	DS	F	Address
SIN_ZERO	DS	XL8	reserved zeros

TIMEVAL structure

TIMEVAL DSECT			
TV_SEC	DS	F	Seconds
TV_USEC	DS	F	Microseconds

Linger structure

LINGER DSECT			
L_ONOFF	DS	F	Option on/off
L_LINGER	DS	F	Linger time

Client identification structure

CLIENTID DSECT			
C_DOMAIN	DS	F	Domain
C_NAME	DS	CL8	User name
C_TASK	DS	CL8	Subtask name
C_RESV1	DS	XL20	reserved zeroes

Error codes

EPERM	EQU	1	Not owner
ENOENT	EQU	2	No such file or directory
ESRCH	EQU	3	No such process
EINTR	EQU	4	Interrupted system call
EIO	EQU	5	I/O error
ENXIO	EQU	6	No such device or address
E2BIG	EQU	7	Arg list too long
ENOEXEC	EQU	8	Exec format error
EBADF	EQU	9	Bad file number
ECHILD	EQU	10	No children
EAGAIN	EQU	11	No more processes
ENOMEM	EQU	12	Not enough core
EACCES	EQU	13	Permission denied
EFAULT	EQU	14	Bad address
ENOTBLK	EQU	15	Block device required
EBUSY	EQU	16	Mount device busy
EEXIST	EQU	17	File exists
EXDEV	EQU	18	Cross-device link
ENODEV	EQU	19	No such device
ENOTDIR	EQU	20	Not a directory
EISDIR	EQU	21	Is a directory
EINVAL	EQU	22	Invalid argument

ENFILE	EQU	23	File table overflow
EMFILE	EQU	24	Too many open files
ENOTTY	EQU	25	Not a typewriter
ETXTBSY	EQU	26	Text file busy
EFBIG	EQU	27	File too large
ENOSPC	EQU	28	No space left on device
ESPIPE	EQU	29	Illegal seek
EROFS	EQU	30	Read-only file system
EMLINK	EQU	31	Too many links
EPIPE	EQU	32	Broken pipe
EDOM	EQU	33	Argument too large
ERANGE	EQU	34	Result too large

Non-blocking and interrupt I/O

EWOULDBL	EQU	35	Operation would block
EINPROGR	EQU	36	Operation now in progress
EALREADY	EQU	37	Operation already in progress

IPC/network software

Argument errors

ENOTSOCK	EQU	38	Socket operation on non-socket
EDESTADD	EQU	39	Destination address required
EMSGSIZE	EQU	40	Message too long
EPROTOTY	EQU	41	Protocol wrong type for socket
ENOPROTO	EQU	42	Protocol not available
EPROTONO	EQU	43	Protocol not supported
ESOCKTNO	EQU	44	Socket type not supported
EOPNOTSU	EQU	45	Operation not supported on socket
EPFNOSUP	EQU	46	Protocol family not supported
EAFNOSUP	EQU	47	Address family not supported by protocol family
*EADDRINU	EQU	48	Address already in use
EADDRNOT	EQU	49	Can't assign requested address

Operational errors

ENETDOWN	EQU	50	Network is down
ENETUNRE	EQU	51	Network is unreachable
ENETRESE	EQU	52	Network dropped connection on reset
ECONNABO	EQU	53	Software caused connection abort
ECONNRES	EQU	54	Connection reset by peer
ENOBUFS	EQU	55	No buffer space available
EISCONN	EQU	56	Socket is already connected
ENOTCONN	EQU	57	Socket is not connected
ESHUTDOWN	EQU	58	Can't send after socket shutdown
ETOOMANY	EQU	59	Too many references: can't splice
ETIMEDOU	EQU	60	Connection timed out
ECONNREF	EQU	61	Connection refused
ELOOP	EQU	62	Too many levels of symbolic links
ENAMETOO	EQU	63	File name too long
EHOSTDOW	EQU	64	Host is down
EHOSTUNR	EQU	65	No route to host
ENOTEMPT	EQU	66	Directory not empty

Quotas & mush

EPROCLIM	EQU	67	Too many processes
EUSERS	EQU	68	Too many users
EDQUOT	EQU	69	Disc quota exceeded

Network File System

ESTALE	EQU	70	Stale NFS file handle
EREMOTE	EQU	71	Too many levels of remote in path

Streams

ENOSTR	EQU	72	Device is not a stream
ETIME	EQU	73	Timer expired
ENOSR	EQU	74	Out of streams resources
ENOMSG	EQU	75	No message of desired type
EBADMSG	EQU	76	Trying to read unreadable message

SystemV IPC

EIDRM	EQU	77	Identifier removed
-------	-----	----	--------------------

SystemV Record Locking

EDEADLK	EQU	78	Deadlock condition.
ENOLCK	EQU	79	No record locks available.

RFS

ENONET	EQU	80	Machine is not on the network
ERREMOTE	EQU	81	Object is remote
ENOLINK	EQU	82	the link has been severed
EADV	EQU	83	advertise error
ESRMNT	EQU	84	srmount error
ECOMM	EQU	85	Communication error on send
EPROTO	EQU	86	Protocol error
EMULTIHO	EQU	87	multihop attempted
EDOTDOT	EQU	88	Cross mount point (not an error)
EREMCHG	EQU	89	Remote address changed

Error codes unique to VM socket implementation

EIBMBADC	EQU	1000	A bad socket-call constant was found in the IUCV header
*			
EIBMBADP	EQU	1001	Other IUCV header error, bad length, etc.
EIBMSCKO	EQU	1002	Socket number assigned by client interface code (for socket() and accept()) is out of range
*			
EIBMSCKI	EQU	1003	Socket number assigned by client interface code is already in use
*			
EIBMIUCV	EQU	1004	Request failed due to IUCV error. This error is generated by the client stub code.
*			
*			
EOFFLERR	EQU	1005	Offload box error
EOFFLRST	EQU	1006	Offload box restart
EOFFLDWN	EQU	1007	Offload box down
EIBMCONF	EQU	1008	There's already a conflicting call outstanding on the socket
*			
EIBMCANC	EQU	1009	Call cancelled via SOCKET CANCEL
EIBMTFAI	EQU	1010	Actually for offload only. Returned by offload box if _beginthread fails
*			

Tracing Structures

RSCS uses the following structures in the internal trace record table. This data is produced when you use the ITRACE command or macro to trace data and specific events.

ITRACFRM

ITRACFRM describes the 16-byte prefix for each sub-entry in each ITRACE record.

X'000'	ITRACFRM	DSECT		Field ID
X'000'	FIELDID	DC	H'0'	Length of this field
X'002'	FIELDLEN	DC	H'0'	Offset of this field within the trace table entry
X'004'	FIELDOFF	DC	H'0'	Eyecatcher - brief description this field
*				
X'006'	FIELDEYE	DC	CL10' '	Length of format table entry
*				
	ITRACFLN	EQU	*-ITRACFRM	

ITRACHDR

ITRACHDR defines the structure of the entire ITRACE table, which is pointed to by the CVT.

```

X'000' ITRACHDR DSECT
X'000' ITRAHTOD DC D'0' Time stamp - last created/updated
X'008' ITRACTOP DC A(0) Pointer to the top of the table
X'00C' ITRACEND DC A(0) Pointer to the end of the table
X'010' DS 0D Double-align the next fields
* as they are subject of a CDS
X'010' ITRACCUR DC A(0) Pointer to the current entry
X'014' ITRACNXT DC A(0) Pointer to the next entry
X'018' ITRACSYS DC A(0) Pointer to the default system task
* ITRACE settings
X'01C' ITRACLNK DC A(0) Pointer to the default line/session
* driver ITRACE settings
X'020' ITRACPOR DC A(0) Pointer to the default auto-answer
* port ITRACE settings
X'024' ITRACSIZ DC H'0' Size of the internal trace table
X'026' ITRACFLG DC X'00' Miscellaneous Flag Byte
X'027' DC XL1'00' Filler for alignment
*
* Bits defined in ITRACFLG
*
ITRACGTR EQU X'80' Copy to GTRACE as well
ITRACDMP EQU X'40' Dump when table wraps
* EQU X'20' spare bit
* EQU X'10' spare bit
* EQU X'08' spare bit
* EQU X'04' spare bit
* EQU X'02' spare bit
* EQU X'01' spare bit
ITR0007 EQU (((*-ITRACHDR)+31)/32)*32
X'028' ORG ITRACHDR+ITR0007 Round to 32_Byte boundary
ITRACHDL EQU *-ITRACHDR ITRACE table header length

```

ITRACREC

The ITRACREC area defines the prefix for each ITRACE record in the internal trace table.

```

X'000' ITRREC DSECT
X'000' ITRRCID DC H'0' Entry type identifier
X'002' ITRRCTID DC H'0' Taskid that generated the entry
X'004' ITRRCNAM DC CL8' ' RSCS task name
X'00C' ITRRCDSC DC CL20' ' Short description of task (ie.
* linkid, ccuu, spool, ast)
X'020' ORG ITRRCDSC+5 Position to LINKID
X'011' ITRRCLNK DC CL8' ' LINKID of this task
X'019' ORG Restore location counter
X'020' ITRRCTOD DC D'0' Time stamp - last created/updated
X'028' ITRRTBLK DC F'0' Pointer to the RSCS TASKBLOK
* (not the GCS TASKBLOK)
X'02C' ITRRCLEN DC F'0' Length of this entire entry -
* including TOD and this length field
X'030' ITRRCFWD DC F'0' Pointer to the next entry
X'034' ITRRCBCK DC F'0' Pointer to the previous entry
X'038' ITRRPLST DS 0F
X'038' DC XL(4+8*10)'00' Generate parm list
ITRRPLEN EQU *-ITRRPLST Length of the plist
ITRRRECX EQU *-ITRREC Length of fixed portion of
a trace table record. X
ITRRRECL EQU (((ITRRRECX+15)/16)*16) Round to left side of
a line of the dump. X
X'08C' DS (ITRRRECL-ITRRRECX)XL1'00' Pad to entry data
X'090' ITRRCDAT DS 0X Start of this entry's data

```

Miscellaneous Structures

This section describes the data areas RSCS uses at various times during its processing.

EVEBLOK

An EVEBLOK represents each request to schedule an event to the EVE task (see “EVEBLOK” on page 31). The different forms of the EVEBLOK are described in the following sections.

General Format

X'000'	EVELENG	DC	XL1'00'	Length of total element - 1
	EVEFLAGM	DS	0XL3	Three byte flag for messages
X'001'	EVETYPE	DC	XL1'00'	Function code
X'002'	EVEQUAL	DC	XL1'00'	Function qualifiers
X'003'	EVMOD	DC	XL1'00'	Function modifiers
* Bits defined in EVETYPE				
* Bits defined in EVEQUAL				
	EVECMDEX	EQU	X'80'	Command execution event
	EVERCALL	EQU	X'40'	Internal module call
	EVEENQUE	EQU	X'20'	Internal call to DMTCOMNQ
	EVEPOST	EQU	X'10'	Internal POST of an ECB
	EVMIDKT	EQU	X'08'	This is the event Midnight
	EVEALTER	EQU	X'04'	Element to do internal alter
* Bits defined in EVEMOD for command elements				
	EVEALTSU	EQU	X'40'	Suspend event
	EVEALTD	EQU	X'80'	Delete event
	EVEALTDL	EQU	X'10'	Perform DISKLOAD
	EVEALTID	EQU	X'08'	Alter is for a particular task
* Bits defined in EVEFLAGM for messages				
	E4RCALL	EQU	EVERCALL*65536	3 byte version of EVERCALL
	E4MIDN	EQU	EVMIDKT*65536	3 byte version of EVMIDKT
	E4FILE	EQU	EVEFILE*256	3 byte version of EVEFILE
	E4SCHED	EQU	EVESCHED*256	3 byte version of EVESCHED
	E4SUSPN	EQU	EVESUSPN	3 byte version of EVESUSPN
X'004'	EVENEXT	DC	A(0)	Pointer to next EVEBLOK
X'008'	EVEDATE	DC	CL8'	Date associated with event
	EVEYEAR	EQU	EVEDATE+0,2	Year sub-field
	EVEMONTH	EQU	EVEDATE+3,2	Month sub-field
	EVEDAY	EQU	EVEDATE+6,2	Day sub-field
X'010'	EVETIME	DC	CL8'	Time associated with event
	EVEHOUR	EQU	EVETIME+0,2	Hour sub-field
	EVEMIN	EQU	EVETIME+3,2	Minute sub-field
X'018'	EVEDOFW	DC	CL8'	Days of week associated with event
	EVERANGE	DS	0CL16	Label for following two fields
X'020'	EVERNGLO	DC	CL8'	Start of time range
X'028'	EVERNGHI	DC	CL8'	End of time range
X'030'	EVETASK	DC	CL8'	Event task name
X'038'	EVETASKN	DC	F'0'	Event task ID
X'03C'		DS	4X	Reserved
X'040'	EVELRTU	DC	F'0'	Event low range in STIMER units
X'044'	EVEHRTU	DC	F'0'	Event high range in STIMER units
X'048'	EVEDELTU	DC	F'0'	Cycle time for repetitive events
				... in STIMER units
X'04C'	EVEGOTU	DC	F'0'	Next execution time for event
				... today in STIMER units
X'050'	EVEGOTOD	DC	D'0'	Next execution time for event
				... today in TOD clock units
X'058'	EVEINLID	DC	CL8'	Link ID
X'060'	EVEINLNK	DC	A(0)	Pointer to incoming LINKTABL,
				... -1 if local MSG, 0 if console
	EVEC	CMORIG	DSECT=NO	Command origin info
X'064'	EVEQUAL	DC	AL1(0)	Origin qualifier
X'065'	EVECFLEG	DC	X'00'	Flags (bits as MSGBFLAG)
X'066'	EVECRSPC	DC	H'0'	Response counter
X'068'	EVECNODE	DC	CL8'	Origin node
X'070'	EVECUSER	DC	CL8'	And user ID
X'078'	EVECSIG	DC	CL6'	Response signature
X'07E'		DS	XL2	Reserved slack bytes
	EVEHDLR	EQU	*-&LABEL	Length of fixed part of EVEBLOK
X'080'	EVEPROTO	DS	0D	Beginning of variable format area

Command Execution Event

X'080'	EVECMD	ORG DS	EVEPROTO CL128	Event command text
--------	--------	--------	----------------	--------------------

Internal Module Call Event

		ORG	EVEPROTO	
X'080'	EVEENTRY	DC	A(0)	Module entry point address
X'084'	EVEREG0	DC	F'0'	REG0 passed to entry point
X'088'	EVEPLIST	DS	0A	Beginning of parm list

Internal Call to Enqueue an Element or POST an ECB

		ORG	EVEPROTO	
X'080'	EVEADELM	DC	A(0)	Address of element to COMNQ
X'084'	EVEADANC	DC	A(0)	Address of the COMNQ anchor
X'088'	EVEADECB	DC	A(0)	Address of the ECB to be POSTed
X'08C'	EVEPOSTC	DC	F'0'	The POST code to be used
		ORG	,	Back to the future
	EVEBLOKL	EQU	256	Maximum length of an EVEBLOK
		MEND		

HASHBLOK

A HASHBLOK defines the characteristics of a hash table. See [“HASHBLOK” on page 112](#) for more information.

X'000'	HATCATCH	DC	CL8'&CATCH'	Hash table eye catcher
X'008'	HATADTOD	DC	D'0'	Time of the last add or delete
	*			... (time in TOD clock format)
X'010'	HATSTCTOD	DC	D'0'	Time of the last statistical update
	*			... (time in TOD clock format)
X'018'	HATSLOTS	DC	A(&ANCHORS)	Number of hash table anchors
X'01C'	HATCHPOF	DC	A(&CHAIN)	Chain pointer offset
X'020'	HATCLPOF	DC	A(&COLLIDE)	Collision pointer offset
X'024'	HATKEYOF	DC	A(&KEY)	Key field offset
X'028'	HATKEYL	DC	A(&KEYLEN)	Key field length (in bytes)
X'02C'	HATSIZE	DC	F'0'	Size of the hash table (in bytes)
X'030'	HATBASE	DC	A(0)	Address of hash table storage
X'034'	HATGANCH	DC	A(0)	Address of the generic key length
	*			... counter array
X'038'	HATFIOPS	DC	F'0'	Number of find requests
X'03C'	HATFIPRO	DC	F'0'	Number of find probes
X'040'	HATNUMCH	DC	F'0'	Number of anchors in use
X'044'	HATMAXCL	DC	F'0'	Length of the longest chain(s)
X'048'	HATCNTCL	DC	XL(4*6)'00000000'	Number of chains of length 1,...,6
X'060'	HATOFFLAG	DC	AL1(&0FLAGS)	Hash option flags
X'061'		DC	XL3'000000'	Filler
X'064'	HATGKEAT	DC	A(0)	Addr of generic key-eating area
X'068'		DC	((*&LABEL+7)/8*8-(*-&LABEL))X'00'	Pad with zeros to Dword boundary
	HATBLEN	EQU	*-&LABEL	Length of hashing descriptor block
	*			
	*		Bits defined in HATOFFLAG	
	*			
	HATLOCK	EQU	X'80'	Hash functions are to run disabled
	HATUANCH	EQU	X'40'	For Build call, unconditionally use
				... the anchor value in the HDB
	HATPSTOR	EQU	X'20'	Get hash table storage from SP 243
	HATGSUPP	EQU	X'10'	Generic key support is active

IOTABLE

RSCS tasks use the IOTABLE to define a request to write output, either to a line or to the spool. When writing a file to spool, the IOTABLE is defined at the start of the output file I/O area (FIOA).

X'000'	IOSYNCH	DC	F'0'	I/O complete ECB
X'004'	IOSYNCHA	DC	F'0'	Asynchronous I/O event ECB
X'008'	IODEVCUU	DC	XL2'00'	Device address (CUU)
X'00A'	IODRSVD1	DC	XL1'00'	Reserved
X'00B'	IODEVCOD	DC	XL1'00'	VM device type
X'00C'	IODEVCAW	DC	A(0)	Address of channel program
X'010'	IODSIOCC	DC	0XL1'00'	SIO condition code
X'010'	IODVSCSW	DC	XL12'00'	Ending subchannel status word
X'01C'	IODVSCSA	DC	XL12'00'	Asynchronous subchannel status word
X'028'	IODSENSE	DC	XL1'00'	Sense byte 0
X'029'	IODRSVD2	DC	XL1'00'	Reserved
X'02A'	IODFLAG1	DC	XL1'00'	Device flag 1
X'02B'	IODFLAG2	DC	XL1'00'	Device flag 2 (NJE only)
X'02C'		DS	F	Filler
X'030'	IOTIME1	DS	1D	STCK value at start of I/O
X'038'	IOTIME2	DS	1D	STCK value at end of I/O
	*		BITS DEFINED IN IODFLAG1	
	IODGOPEN	EQU	X'80'	Device is open for general I/O
	IODSTART	EQU	X'40'	I/O has been started for this device
	IODURIO	EQU	X'20'	I/O is for unit record device
	IODASYIO	EQU	X'10'	I/O is asynchronous
	IODSPERR	EQU	X'08'	Spooling error occurred

MONITENT

* BITS DEFINED IN IODFLAG2			
IODACTIV	EQU	X'80'	Device is active
IODNOP	EQU	X'20'	Store all records as NOPs
IODHOLD	EQU	X'10'	File on this device is to be held
IODSTUB	EQU	X'08'	Stub for re-enqueue
IODSUSP	EQU	X'04'	Suspended device
IODFNREN	EQU	X'02'	Renamed to fanout *MULTI*
IODKEEPH	EQU	X'01'	Hdrs being retained on device
* GENIO MF=L GENIO parameter list			
X'040'	IODGLIST	DS 0F	
X'040'		DC CL8'GENIO'	Macro name
X'048'		DC X'00'	Function requested
X'049'		DC X'00'	Reserved flag
X'04A'		DC X'0000'	Device address
X'04C'		DC F'0'	CCW address
X'050'		DC F'0'	Data address
X'054'		DC F'0'	Exit address
X'058'		DC F'0'	UWORD address
	IODGLLEN	EQU *-IODGLIST	GENIO parameter list length
X'05C'	IODNEXT	DC A(0)	Address of next device table (NJE)
X'060'	IODRLINK	DC A(0)	Address of LINKTAB1 for ROUTE (NJE)
X'064'	IONFS	DC A(0)	Address of next free slot in FIOA
X'068'	IOEBA	DC A(0)	Address of current end of buffer
X'06C'	IODEVXAB	DC A(0)	Address of XAB
X'070'	IODEVXL	DC H'0'	Length of XAB
X'072'		DC XL2'00'	Reserved
X'074'	IODEVTAG	DC A(0)	Active device TAG slot (NJE only)
X'078'	IOCHANPG	DC 0D'0'	Start of unit record channel pgm.
	IOTABLEN	EQU *-IOTABLE	Length of table

MONITENT

MONITENT entries represent message subscriptions. For more information, see [“Message Subscriptions” on page 30](#).

X'000'	MONNEXT	DS	A	Pointer to next chained entry
X'004'	MONNQVAL	DS	CL1	Node qualifier for MONNODE
X'005'	MONCQUAL	DS	CL1	Node qualifier for MONCNODE
X'006'	MONAPIFL	DS	X	API flag byte ... for bit settings see MSGBFLAG
X'007'	MONFLAG	DS	X	Flag byte for MONITENT
	*			
	*			Bits defined in MONFLAG
	*			
	MONLINK	EQU	X'80'	This is a link monitoring entry
	MONDISAB	EQU	X'40'	Entry disabled (MONLINK only)
X'008'	MONNODE	DS	CL8	Nodeid for send messages
X'010'	MONUSER	DS	CL8	Userid for send messages
X'018'	MONCNODE	DS	CL8	Command originator's node
X'020'	MONCUSER	DS	CL8	Command originator's userid
X'028'	MONSIGN	DS	CL6	Signature for CRI response
	MONLNKLN	EQU	*-MONITENT	Length of link monitoring entry
X'02D'	MONSMCNT	DS	H	Counter for error message
	MONSMMAX	EQU	100	Message every 100th time
X'030'	MONBITS	DS	XL125	Bitmap for message based monitoring entries
	MONBITSL	EQU	*-MONBITS	Length of the bitmap in bytes
	MONMSGLEN	EQU	*-MONITENT	Length of msg monitoring entry

SAVEAREA

The SAVEAREA macro generates the mapping DSECT for the RSCS register save area and extension.

X'000'	SAVEPOOL	DS	X	Subpool of storage in stack
X'001'	SAVESTOL	DS	2X	Length of storage in stack (used in first SAVEAREA only)
X'003'	SAVEDPTH	DS	X	Number of saveareas (including this one) remaining in stack
X'004'	SAVEPREV	DS	A	Address of previous save area
X'008'	SAVENEXT	DS	A	Address of next save area
X'00C'	SAVER14	DS	F	Register 14
X'010'	SAVER15	DS	F	Register 15
X'014'	SAVER0	DS	F	Register 0
X'018'	SAVER1	DS	F	Register 1
X'01C'	SAVER2	DS	F	Register 2
X'020'	SAVER3	DS	F	Register 3
X'024'	SAVER4	DS	F	Register 4
X'028'	SAVER5	DS	F	Register 5
X'02C'	SAVER6	DS	F	Register 6
X'030'	SAVER7	DS	F	Register 7
X'034'	SAVER8	DS	F	Register 8
X'038'	SAVER9	DS	F	Register 9
X'03C'	SAVER10	DS	F	Register 10
X'040'	SAVER11	DS	F	Register 11
X'044'	SAVER12	DS	F	Register 12
X'048'	SAVEWRK1	DS	F	Work area word 1
X'04C'	SAVEWRK2	DS	F	Work area word 2
X'050'	SAVEWRK3	DS	F	Work area word 3
X'054'	SAVEWRK4	DS	F	Work area word 4
X'058'	SAVEWRK5	DS	F	Work area word 5
X'05C'	SAVEWRK6	DS	F	Work area word 6
X'060'	SAVEWRK7	DS	F	Work area word 7
X'064'	SAVEWRK8	DS	F	Work area word 8
	SAVEALEN	EQU	*-SAVEAREA	Length of the save area

Chapter 18. Command and Request Elements

This chapter describes the format of the command and request elements that RSCS uses to notify tasks to perform work.

CMNDAREA

PI

The CMNDAREA macro maps the internal command formats, which are used when sending and queueing commands and messages to modules. The specific format of the structure depends on the command type:

Type A

Spool manager task.

Type C

Auto-start task.

Type E

EXEC processor task.

Type L

Link driver tasks.

Type V

SNA control task.

Basic Structure

Each type of command element format contains a common format. The origin of each command is described by the CMORIG DSECT.

X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
NOT Programming Interface Information				
*	Required by DMTCOMNQ/DQ			
End of NOT Programming Interface Information				
X'001'	CMNDTYPE	DC	XL1'00'	Type of command
NOT Programming Interface Information				
*	Command types are defined in RSSEQU and are used by requester and server to distinguish between different types of a command.			
*				
*				
End of NOT Programming Interface Information				
X'002'	CMNDMOD	DC	XL1'00'	Command modifier
*	This byte is used to define extra refinements on the basic command type, eg, the reason for a reorder			
*				
*				
X'003'		DS	XL1	Filler
*	Origin of command Includes origin node, user ID, qual and various CRI settings			
CMND				
*				
*				
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the ... format depends on CMNDTYPE
*				

Type A0 (REORDER)

NOT Programming Interface Information

The A0 element is used to pass REORDER commands from the command processing modules of the REX task to the spool manager task.

End of NOT Programming Interface Information

X'000'	ORG	CMNDTEXT		
	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
X'001'	*			Required by DMTCOMNQ/DQ
	CMNDTYPE	DC	XL1'00'	Type of command
	*			Command types are defined in RSSEU and are
	*			used by requester and server to distinguish
	*			between different types of a command.
X'002'	CMNDMOD	DC	XL1'00'	Command modifier
	*			This byte is used to define extra
	*			refinements on the basic command
	*			type, eg, the reason for a reorder
X'003'		DS	XL1	Filler
	*			
	*			Command type/modifiers
	*			
	REORDCMD	EQU	X'01'	REORDER command
	A0REORIN	EQU	X'00'	- full internal reorder
	A0REORRL	EQU	X'80'	- full real reorder
	A0REORDF	EQU	X'40'	- DEFINE command, new link
	A0REORST	EQU	X'20'	- link started
	A0REORNT	EQU	X'10'	- NETWORK START completes
	A0REORRD	EQU	X'08'	- DEFINE command, old link
	A0REORLP	EQU	X'04'	- LOOPING command issued
	A0REORRO	EQU	X'02'	- ROUTE command
	A0REORDA	EQU	X'01'	- link deactivated
	*			
	CMND		CMORIG DSECT=NO	Origin of command
	*			Includes origin node, user ID, qual
	*			and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
	*			... format depends on CMNDTYPE
X'020'	A0LINKAD	DC	A(0)	Address of effected LINKTABL
X'024'	A0REFLAG	DC	X'00'	Reorder command flag byte
X'025'		DC	XL3'00'	Reserved
	A0CMDLEN	EQU	*-&LABEL-1	Length of whole minus one
	*			
	*			Bits defined in A0REFLAG byte
	*			
	A0SUPQRM	EQU	X'80'	Signal spool manager to suppress
				... the 'queue reordered' message

Type A1 (CLOSE, ORDER, PURGE)

NOT Programming Interface Information

The REX task, link drivers, and DMTMANEX use the Type A1 element to pass ORDER, PURGE and CLOSE commands to the spool manager task.

End of NOT Programming Interface Information

X'000'	ORG	CMNDTEXT		
*	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
X'001'	*			Required by DMTCOMN/DQ
*	CMNDTYPE	DC	XL1'00'	Type of command
*	*			Command types are defined in RSSEU and are
*	*			used by requester and server to distinguish
X'002'	*			between different types of a command.
*	CMNDMOD	DC	XL1'00'	Command modifier
*	*			This byte is used to define extra
*	*			refinements on the basic command
X'003'	*			type, eg, the reason for a reorder
*		DS	XL1	Filler
*				
*				Command types/modifiers
*				
	ORDERCMD	EQU	X'10'	ORDER command
	PURGECMD	EQU	X'11'	PURGE command
	PURGEFLT	EQU	X'80'	- filter program included
	PURGNFIL	EQU	X'40'	- no file around, but clean up data areas
	CLOSECMD	EQU	X'12'	CLOSE command
	CLSALL	EQU	X'40'	- all files
*				
*	CMND		CMORIG DSECT=NO	Origin of command
*	*			Includes origin node, user ID, qual
*	*			and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
*	*			... format depends on CMNDTYPE
X'020'	A10BJLNK	DC	CL8' '	Link concerned
X'028'	A1COUNT	DC	H'0'	Number of spool IDs involved
	A1SPIDL	DS	0CL134	Max length of spool ID string
	A1SPIDS	DS	0H	Spool IDs
	A1FILTER	DS	0XL(SHTFLTIN)	Filter program

Type A1 (TRANSFER)

NOT Programming Interface Information

The REX task passes the spool manager task a modified version of the A1 element for the TRANSFER command.

End of NOT Programming Interface Information

X'000'	ORG	CMNDTEXT		
*	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
X'001'	*			Required by DMTCOMN/DQ
*	CMNDTYPE	DC	XL1'00'	Type of command
*	*			Command types are defined in RSSEU and are
*	*			used by requester and server to distinguish
X'002'	*			between different types of a command.
*	CMNDMOD	DC	XL1'00'	Command modifier
*	*			This byte is used to define extra
*	*			refinements on the basic command
X'003'	*			type, eg, the reason for a reorder
*		DS	XL1	Filler
*				
*				Command type/modifier
*				
	TRANSCMD	EQU	X'13'	TRANSFER command
	TRANSFLT	EQU	X'80'	- filter specified
*				
*	CMND		CMORIG DSECT=NO	Origin of command
*	*			Includes origin node, user ID, qual
*	*			and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
*	*			... format depends on CMNDTYPE
X'020'	A1TOBLNK	DC	CL8' '	Link concerned
X'028'	A1NEWLOC	DC	CL8' '	New destination node
X'030'	A1NEWVM	DC	CL8' '	And user ID

For filter version

X'038'	A1TFILTR	DC	XL(SHTFLTLN)'00'	Filter program
	A1TSCMDL	EQU	*-&LABEL-1	"Short" length minus one

For non-filter version

X'038'	A1TCOUNT	ORG	A1TFILTR	For non-filter versions,
	A1TSPIDL	DC	H'0'	Count of spool IDs
	A1TSPIDS	DS	0CL134	Max length of spool ID string
	A1TCMDLN	DS	0H	Spool IDs
		EQU	*-&LABEL-1	Length of element minus one

Type A2 (CHANGE)

The REX task (command processor) passes the Type A2 element to the spool manager task to represent the CHANGE command.

X'000'	ORG	CMNDTEXT		
	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
	*			Required by DMTCOMNQ/DQ
X'001'	CMNDTYPE	DC	XL1'00'	Type of command
	*			Command types are defined in RSSEQU and are
	*			used by requester and server to distinguish
	*			between different types of a command.
X'002'	CMNDMOD	DC	XL1'00'	Command modifier
	*			This byte is used to define extra
	*			refinements on the basic command
	*			type, eg, the reason for a reorder
X'003'		DS	XL1	Filler
	*			
	*			Command type/modifier
	*			
	CHANGCMD	EQU	X'20'	CHANGE command
	*			
	CMND	CMORIG	DSECT=NO	Origin of command
	*			Includes origin node, user ID, qual
	*			and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter

X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
	*			... format depends on CMNDTYPE
X'020'	A2LINK	DC	CL8' '	Link concerned
	A2TEXT	EQU	*	Stuff we move in from prototype
X'028'	A2NOLLOOP	DC	XL1'00'	NOLLOOP indicator
X'029'	A2NOPREF	DC	XL1'00'	NOPREFLINK indicator
X'02A'	A2PRIOR	DC	XL2'0000'	New priority
	*			
	* Note: from H0 to DEST must correspond with VCHCNTRL in DMTAXM			
	*			
X'02C'	A2HO	DC	XL1'00'	New HOLD/NOHOLD state
X'02D'	A2CL	DC	XL1'00'	New class
X'02E'	A2COPY	DC	XL2'0000'	New copy count
X'030'	A2DIST	DC	CL8' '	New distribution code
X'038'	A2DSN	DS	0CL24	New dsname
X'038'	A2FN	DC	CL12' '	Filename
X'044'	A2FT	DC	CL12' '	Filetype
X'050'	A2FORMN	DC	CL8' '	New form name
X'058'	A2FLSHN	DC	CL4' '	New flash name
X'05C'	A2MODN	DC	CL4' '	New copy mod name
X'060'	A2CHARS	DS	0CL16	New chars names - 0-3
X'060'	A2CHARN	DC	CL4' '	0
X'064'	A2CHARX	DS	0CL12	1-3
X'064'	A2CHAR1	DC	CL4' '	1
X'068'	A2CHAR2	DC	CL4' '	2
X'06C'	A2CHAR3	DC	CL4' '	3
X'070'	A2FCBN	DC	CL4' '	New FCB name
X'074'	A2FLC	DC	XL1'00'	New flash count
X'075'	A2FLAG1	DC	XL1'00'	New copy group flag
X'076'	A2MODTR	DC	CL1' '	New modify TRC
X'077'	A2RESV	DC	XL1'00'	Reserved
X'078'	A2DEST	DC	CL8' '	New DEST name
	*			
	* Note: from H0 to DEST must correspond with VCHCNTRL in DMTAXM			
	*			
X'080'	A2JULN	DC	XL2'0000'	NOTIFY processing day number
	A2VLEN	EQU	*-A2TEXT	Length of command text section
X'082'	A2FILTER	DC	XL(SHTFLTLN)'00'	Filter to select files to be
				... changed
	A2LEN	EQU	*-&LABEL-1	Length of whole, minus one

Type C0 (FORCE)

The REX task (command processor DMTCMYFO) passes a Type C0 element to the auto-start task to request it to FORCE a link that it attached.

X'000'	ORG	CMNDTEXT			
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1	
X'001'	*			Required by DMTCMNQ/DQ	
X'001'	CMNDTYPE	DC	XL1'00'	Type of command	
X'002'	*			Command types are defined in RSSEU and are used by requester and server to distinguish between different types of a command.	
X'002'	CMNDMOD	DC	XL1'00'	Command modifier	
X'003'	*			This byte is used to define extra refinements on the basic command type, eg, the reason for a reorder	
X'003'		DS	XL1	Filler	
X'004'	*			Command type/modifier	
X'005'	FORCECMD	EQU	X'C0'	FORCE command	
X'006'	*				
X'007'	CMND	CMORIG DSECT=NO		Origin of command	
X'008'	*			Includes origin node, user ID, qual and various CRI settings	
X'009'	CMNDQUAL	DC	AL1(0)	Origin qualifier	
X'010'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)	
X'011'	CMNDRSPC	DC	H'0'	Response counter	
X'012'	CMNDNODE	DC	CL8' '	Origin node	
X'013'	CMNDUSER	DC	CL8' '	and user ID	
X'014'	CMNDSIG	DC	CL6' '	Response signature	
X'015'	*	DS	XL2	Filler	
X'016'	CMNDTEXT	DS	0D	Text of command: from here, the ... format depends on CMNDTYPE	
X'017'	*				
X'018'	C0LINKID	DC	CL8' '	Link to be forced	
X'019'	C0ALTLEN	EQU	*-&LABEL-1	Length of whole, minus one	

Type C1 (ITO)

After they initialize, link drivers pass the Type C1 element for the ITO command to the auto-start task. This indicates that the links are enrolling in the ITO process.

X'000'	ORG	CMNDTEXT			
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1	
X'001'	*			Required by DMTCMNQ/DQ	
X'001'	CMNDTYPE	DC	XL1'00'	Type of command	
X'002'	*			Command types are defined in RSSEU and are used by requester and server to distinguish between different types of a command.	
X'002'	CMNDMOD	DC	XL1'00'	Command modifier	
X'003'	*			This byte is used to define extra refinements on the basic command type, eg, the reason for a reorder	
X'003'		DS	XL1	Filler	
X'004'	*			Command type/modifier	
X'005'	ITOCMD	EQU	X'C1'	Enroll-in-ITO command	
X'006'	*				
X'007'	CMND	CMORIG DSECT=NO		Origin of command	
X'008'	*			Includes origin node, user ID, qual and various CRI settings	
X'009'	CMNDQUAL	DC	AL1(0)	Origin qualifier	
X'010'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)	
X'011'	CMNDRSPC	DC	H'0'	Response counter	
X'012'	CMNDNODE	DC	CL8' '	Origin node	
X'013'	CMNDUSER	DC	CL8' '	and user ID	
X'014'	CMNDSIG	DC	CL6' '	Response signature	
X'015'	*	DS	XL2	Filler	
X'016'	CMNDTEXT	DS	0D	Text of command: from here, the ... format depends on CMNDTYPE	
X'017'	*				
X'018'	C1LINKID	DC	CL8' '	Link which wants to enroll	
X'019'	C1ALTLEN	EQU	*-&LABEL-1	Length of whole, minus one	

Type C2 (RETRY)

The Type C2 element is passed from the REX or AST task (DMTMANEX) to the auto-start task to start retry processing for a link that has deactivated.

X'000'	ORG	CMNDTEXT			
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1	
X'001'	CMNDTYPE	DC	XL1'00'	Required by DMTCOMNQ/DQ	
*				Type of command	
*				Command types are defined in RSSEQU and are	
*				used by requester and server to distinguish	
X'002'	CMNDMOD	DC	XL1'00'	between different types of a command.	
*				Command modifier	
*				This byte is used to define extra	
*				refinements on the basic command	
X'003'		DS	XL1	type, eg, the reason for a reorder	
*				Filler	
*					
*				Command type/modifier	
*					
	RETRYCMD	EQU	X'C2'	Start-retrying command	
*					
*	CMND	CMORIG	DSECT=NO	Origin of command	
*				Includes origin node, user ID, qual	
*				and various CRI settings	
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier	
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)	
X'006'	CMNDRSPC	DC	H'0'	Response counter	
X'008'	CMNDNODE	DC	CL8' '	Origin node	
X'010'	CMNDUSER	DC	CL8' '	and user ID	
X'018'	CMNDSIG	DC	CL6' '	Response signature	
X'01E'		DS	XL2	Filler	
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the	
*				... format depends on CMNDTYPE	
X'020'	C2LINKID	DC	CL8' '	Link to retry	
C2ALTLEN	EQU		*-&LABEL-1	Length of whole, minus one	

Type E0 (Execs)

Type E0 elements pass exec requests to the EXE task. E0 requests are issued by the EXEC command processor, DMTCMZEX, running under REX task. They can also be issued for non-SNA links by DMTMANEX, running under the REX or AST tasks.

X'000'	ORG	CMNDTEXT			
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1	
X'001'	CMNDTYPE	DC	XL1'00'	Required by DMTCOMNQ/DQ	
*				Type of command	
*				Command types are defined in RSSEQU and are	
*				used by requester and server to distinguish	
X'002'	CMNDMOD	DC	XL1'00'	between different types of a command.	
*				Command modifier	
*				This byte is used to define extra	
*				refinements on the basic command	
X'003'		DS	XL1	type, eg, the reason for a reorder	
*				Filler	
*					
*	CMND	CMORIG	DSECT=NO	Origin of command	
*				Includes origin node, user ID, qual	
*				and various CRI settings	
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier	
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)	
X'006'	CMNDRSPC	DC	H'0'	Response counter	
X'008'	CMNDNODE	DC	CL8' '	Origin node	
X'010'	CMNDUSER	DC	CL8' '	and user ID	
X'018'	CMNDSIG	DC	CL6' '	Response signature	
X'01E'		DS	XL2	Filler	
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the	
*				... format depends on CMNDTYPE	
X'020'	E0INLINK	DC	AL4(0)	Incoming link ID for EXEC	
X'024'	E0TYPE	DC	XL1'00'	X'FF' if restart from MAN	
X'025'		DS	3X	Pad to fullword	
	E0HDRLEN	EQU	*-&LABEL	Length excluding name/parms	
X'028'	E0EXPARM	DC	CL132' '	EXEC name and parameters	
E0ALTLEN	EQU		*-&LABEL-1	Length-1 of whole	

Type L0 (DRAIN, FREE, HOLD, READY, START, and TRACE)

The L0 element is used to pass START, DRAIN, FREE, HOLD and TRACE commands from the REX task (DMTCMY) to active link driver tasks. For the TRACE command, this element contains the destination node and user ID to receive the trace file.

ORG	CMNDTEXT			
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
X'001'	CMNDTYPE	DC	XL1'00'	Required by DMTCOMN/DQ
*				Type of command
*				Command types are defined in RSSEQU and are
*				used by requester and server to distinguish
X'002'	CMNDMOD	DC	XL1'00'	between different types of a command.
*				Command modifier
*				This byte is used to define extra
*				refinements on the basic command
X'003'		DS	XL1	type, eg, the reason for a reorder
*				Filler
*				Command types/modifiers
*				
STRTCMD	EQU	X'80'		START command
STACCLASS	EQU	X'80'		- reset class
DRCMD	EQU	X'81'		DRAIN command
FREECMD	EQU	X'82'		FREE command
HOLDCMD	EQU	X'83'		HOLD command
HOLDIMM	EQU	X'80'		- immediate
HOLDINP	EQU	X'40'		- input files
TRACECMD	EQU	X'84'		TRACE command
TRACNLOG	EQU	X'10'		- finish tracing (OFF/NOLOG etc)
TRACTO	EQU	X'04'		- TO node/user ID specified
				- other modifiers from LTRxxx
				fields, see LINKTABL MACRO
READYCMD	EQU	X'85'		READY command
*				
CMND	CMORIG	DSECT=NO		Origin of command
*				Includes origin node, user ID, qual
*				and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
*				... format depends on CMNDTYPE
X'020'	L0TTOLOC	DC	CL8' '	Node to send trace to
X'028'	L0TTOVM	DC	CL8' '	User ID to send trace to
	L0ALTLEN	EQU	*-&LABEL-1	Length of this format

Type L1 (BACKSPACE, FWDSPACE)

The REX task (DMTCMY) uses the Type L1 element to pass BACKSPACE and FWDSPACE commands to active printer links.

ORG	CMNDTEXT			
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
X'001'	CMNDTYPE	DC	XL1'00'	Required by DMTCOMN/DQ
*				Type of command
*				Command types are defined in RSSEQU and are
*				used by requester and server to distinguish
X'002'	CMNDMOD	DC	XL1'00'	between different types of a command.
*				Command modifier
*				This byte is used to define extra
*				refinements on the basic command
X'003'		DS	XL1	type, eg, the reason for a reorder
*				Filler
*				Command types/modifiers
*				
BACKCMD	EQU	X'90'		BACKSPACE command
BACKFILE	EQU	X'00'		- by whole file
BACKCNT	EQU	X'80'		- by # records in L1COUNT
FWD CMD	EQU	X'91'		FWDSPACE command
				(always by # recs in L1COUNT)
*				
CMND	CMORIG	DSECT=NO		Origin of command
*				Includes origin node, user ID, qual
*				and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
*				... format depends on CMNDTYPE
X'020'	L1COUNT	DC	F'0'	BACKSPAC/FWDSPACE count
	L1ALTLEN	EQU	*-&LABEL-1	Length of this format

Type L2 (FLUSH)

The REX task (DMTCMY) uses the Type L3 element to pass a FLUSH command to active links that also have active files.

X'000'	ORG	CMNDTEXT			
	CMNDLEN	DC	XL1'00'	Length of total element, minus 1	
X'001'	*			Required by DMTCMNQ/DQ	
	CMNDTYPE	DC	XL1'00'	Type of command	
	*			Command types are defined in RSSEU and are	
	*			used by requester and server to distinguish	
	*			between different types of a command.	
X'002'	CMNDMOD	DC	XL1'00'	Command modifier	
	*			This byte is used to define extra	
	*			refinements on the basic command	
	*			type, eg, the reason for a reorder	
X'003'		DS	XL1	Filler	
	*				
	*			Command type/modifiers	
	*				
	FLUSHCMD	EQU	X'A0'	FLUSH command	
	FLUSHCPY	EQU	X'00'	- just single copy	
	FLUSHALL	EQU	X'80'	- all copies	
	FLUSHOLD	EQU	X'40'	- hold flushed file	
	*				
	CMND		CMORIG DSECT=NO	Origin of command	
	*			Includes origin node, user ID, qual	
	*			and various CRI settings	
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier	
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)	
X'006'	CMNDRSPC	DC	H'0'	Response counter	
X'008'	CMNDNODE	DC	CL8' '	Origin node	
X'010'	CMNDUSER	DC	CL8' '	and user ID	
X'018'	CMNDSIG	DC	CL6' '	Response signature	
X'01E'		DS	XL2	Filler	
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the	
	*			... format depends on CMNDTYPE	
X'020'	L2COUNT	DC	H'0'	Number of spool IDs to flush	
	L2SPIDS	DS	0H	Each spool ID mentioned	

Type L3 (Commands, Messages)

The L3 type element is used for passing commands and messages from task to task.

X'000'	ORG	CMNDTEXT			
	CMNDLEN	DC	XL1'00'	Length of total element, minus 1	
X'001'	*			Required by DMTCMNQ/DQ	
	CMNDTYPE	DC	XL1'00'	Type of command	
	*			Command types are defined in RSSEU and are	
	*			used by requester and server to distinguish	
	*			between different types of a command.	
X'002'	CMNDMOD	DC	XL1'00'	Command modifier	
	*			This byte is used to define extra	
	*			refinements on the basic command	
	*			type, eg, the reason for a reorder	
X'003'		DS	XL1	Filler	
	*				
	*			Command type/modifiers	
	*				
	CMDCMD	EQU	X'B0'	Command flavor	
	MSGCMD	EQU	X'B1'	Message flavor	
	*				
	CMND		CMORIG DSECT=NO	Origin of command	
	*			Includes origin node, user ID, qual	
	*			and various CRI settings	
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier	
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)	
X'006'	CMNDRSPC	DC	H'0'	Response counter	
X'008'	CMNDNODE	DC	CL8' '	Origin node	
X'010'	CMNDUSER	DC	CL8' '	and user ID	
X'018'	CMNDSIG	DC	CL6' '	Response signature	
X'01E'		DS	XL2	Filler	
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the	
	*			... format depends on CMNDTYPE	
X'020'	L3TOLOC	DC	CL8' '	Destination node	
X'028'	L3TOVM	DC	CL8' '	And user ID	
	*				
X'030'	L3NOUT	DC	XL8'00'	NMR local output information	
X'038'	L3NFLAG	DC	XL1'00'	NMR flag byte	

X'039'	L3NLEVEL	DC	XL1'77'	NMR importance level
X'03A'	L3NTYPE	DC	XL1'00'	NMR type byte
X'03B'	L3NTOQUL	DC	XL1'00'	NMR TO node qualifier
X'03C'	L3NFMQUL	DC	XL1'00'	NMR FROM node qualifier
X'03D'	L3NDCTR	DC	XL1'00'	NMR indicator field
	L3NMRSF	EQU	X'80'	Indicates NMR is S&F
	L3NMRLN	EQU	*-L3NOUT	Define NMR data length
X'03E'	*	DS	XL2	Fill out to end of L3NOUT
X'040'	L3INLINK	DC	A(0)	Pointer to input link
	*			... or 0 for SMSG-type command
	*			... or -1 for console command
	L3HDRLEN	EQU	*-&LABEL	Length excluding command
X'044'	L3TEXT	DC	CL140'	Command text
	L3TOTLEN	EQU	*-&LABEL-1	Length of whole minus 1

Type V1 (START)

The REX and AST tasks use the Type V1 element to pass a START command for a SNA-type link to the SCT task.

X'000'	ORG CMNDTEXT			
	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
	*			Required by DMTCOMNQ/DQ
X'001'	CMNDTYPE	DC	XL1'00'	Type of command
	*			Command types are defined in RSSEU and are
	*			used by requester and server to distinguish
	*			between different types of a command.
X'002'	CMNDMOD	DC	XL1'00'	Command modifier
	*			This byte is used to define extra
	*			refinements on the basic command
	*			type, eg, the reason for a reorder
X'003'		DS	XL1	Filler
	*			Command type/modifiers
	V1STCMND	EQU	X'40'	START command for SNA link
	*			
	CMND	CMORIG DSECT=NO		Origin of command
	*			Includes origin node, user ID, qual
	*			and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
	*			... format depends on CMNDTYPE
X'020'	V1LINKID	DC	CL8' '	Link to start
X'028'	V1ACTLUN	DC	CL8' '	LUname
X'030'	V1ACTLOG	DC	CL8' '	Logmode
	V1ALTLEN	EQU	*-&LABEL-1	Length of whole, minus one

Type V2 (STOP)

The REX and AST tasks use the Type V2 element to pass a STOP command for a SNA-type link to the SCT task.

MSGBLOK

	ORG	CMNDTEXT		
X'000'	CMNDLEN	DC	XL1'00'	Length of total element, minus 1
*				Required by DMTCOMN/DQ
X'001'	CMNDTYPE	DC	XL1'00'	Type of command
*				Command types are defined in RSSEQU and are
*				used by requester and server to distinguish
*				between different types of a command.
X'002'	CMNDMOD	DC	XL1'00'	Command modifier
*				This byte is used to define extra
*				refinements on the basic command
*				type, eg, the reason for a reorder
X'003'		DS	XL1	Filler
*				
*				Command type/modifiers
*				
	V2STPCMD	EQU	X'40'	STOP command for SNA link
*				
*	CMND	CMORIG	DSECT=NO	Origin of command
*				Includes origin node, user ID, qual
*				and various CRI settings
X'004'	CMNDQUAL	DC	AL1(0)	Origin qualifier
X'005'	CMNDFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'006'	CMNDRSPC	DC	H'0'	Response counter
X'008'	CMNDNODE	DC	CL8' '	Origin node
X'010'	CMNDUSER	DC	CL8' '	and user ID
X'018'	CMNDSIG	DC	CL6' '	Response signature
X'01E'		DS	XL2	Filler
X'020'	CMNDTEXT	DS	0D	Text of command: from here, the
*				... format depends on CMNDTYPE
X'020'	V2LINKID	DC	CL8' '	LINKID
X'028'	V2ACTLUN	DC	CL8' '	LUname
	V2ALTLEN	EQU	*-&LABEL-1	Length of whole, minus one

MSGBLOK

PI

A message request parameter list (MSGBLOK) contains information about a message request. It is built by any module that needs to issue a message.

X'000'	MSGBACT	DC	C' '	Action code (length for RF=REX)
NOT Programming Interface Information				
X'001'	MSGBMOD	DC	CL3'&MODID'	Calling module identifier
End of NOT Programming Interface Information				
X'004'	MSGBNUM	DC	AL2(0)	Message number
X'006'	MSGBRCD	DC	XL1'00'	Routing code override
X'007'	MSGBSCOD	DC	XL1'00'	Severity code override
X'008'	MSGBQCN	DC	H(0)	Q message limit counter
NOT Programming Interface Information				
X'00A'		DC	AL2(0)	Filler
*				
*				The following fields should be kept in order,
*				to correspond with CMORIG
End of NOT Programming Interface Information				
X'00C'	MSGBQAL	DC	AL1(0)	Destination qualifier
X'00D'	MSGBFLAG	DC	X'00'	Flag byte for MSGBLOK
*				
*				Bits defined in MSGBFLAG
NOT Programming Interface Information				
*				(CRI-related from PAFAPIFL).
*				These bit settings are also used in MONAPIFL in MONITENT
*				and xxxxFLAG in CMORIG
End of NOT Programming Interface Information				
	MSGBAPI	EQU	X'80'	Using the CRI
	MSGBMSG	EQU	X'40'	Use MSG, not SMSG (CRI)
	MSGBTEXT	EQU	X'20'	Deliver with text (CRI)
	MSGBCMDP	EQU	X'10'	Message from command proc
	MSGBNTXT	EQU	X'08'	Network language text (CRI)
	MSGBLTXT	EQU	X'04'	Local language text (CRI)
	MSGBRDYN	EQU	X'02'	RMSG-dynamic call to message builder
	MSGBQRY	EQU	X'01'	Query response
	*			
X'00E'	MSGBRSPC	DC	H'0'	CRI response counter
X'010'	MSGBLOC	DC	CL8' '	Destination node
X'018'	MSGBVM	DC	CL8' '	Destination userid
X'020'	MSGBSIGN	DC	CL6' '	CRI signature
	*			
	MSGBORIG	EQU	&MSGBQAL,*-&MSGBQAL Origin fields altogether	
NOT Programming Interface Information				
X'026'	*			
		End of CMORIG bits		
	DC	AL2(0)		Filler
End of NOT Programming Interface Information				

RDEVBLK

X'028'	MSGBWA	DC	A(0)	Pointer to MSGWA
X'02C'	MSGBFUN	DC	XL1'02'	Function code
X'02D'	MSGBFLG2	DC	XL1'00'	Second flag byte
*				
Bits defined in MSGBFLG2				
*				
MSGBSVDY	EQU	X'80'		Dynamic saveareas allocated
MSGBCHOP	EQU	X'40'		Columns have been chopped
MSGBNORR	EQU	X'20'		Don't try to reroute again
MSGCAPI	EQU	X'10'		API responses for console, not orig routing
MSGBNORC	EQU	X'08'		Indicates not Recv rer done - don't try again
*				
MSGBQLIM	EQU	X'04'		Query message limit reached

NOT Programming Interface Information

X'02E'	DC	AL2(0)	Filler
--------	----	--------	--------

End of NOT Programming Interface Information

X'030'	MSGBALNK	DC	CL8' '	Associated link
X'038'	MSGBSHOW	DC	XL8'00'	SHOW mask
X'040'	MSGBUSER	DC	D'0'	User exit work area
X'048'	MSGBCONV	DC	A(0)	User conversion repository
X'04C'	MSGBTRAN	DC	A(0)	User translation repository
*				
MSGBHLEN	EQU	*-MSGBLOK		Header length
X'050'	MSGBVARS	DS	0CL8	Start of variables
	DC	&VARS.CL8' '		Number of variables
	MEND			

PI end

RDEVBLK

RSCS tasks pass a file request element (RDEVBLK) to the AXM task to open or close a spool file. Networking link driver tasks also use the RDEVBLK to initialize transmission algorithms.

X'000'	RDEVRLN	DC	AL1(0)	Request length
X'001'	RDEVFUN	DC	XL1'00'	Request function
	*			
	*		Bits Defined in RDEVFUN	
	*			
	RDEV0IN	EQU	X'01'	Open input
	RDEV0OUT	EQU	X'11'	Open output
	RDEVOTA	EQU	X'21'	Open transmission algorithm
	RDEVCIN	EQU	X'02'	Close input
	RDEVCOUT	EQU	X'12'	Close output
X'002'	RDEVFLG1	DC	XL1'00'	Flags
X'003'	RDEVSOPT	DC	XL1'00'	Sub-option
	*			
	*		Bits Defined in RDEVSOPT	
	*			
	RDEVSPHL	EQU	X'80'	Open input (no special hold)
	RDEVRXAB	EQU	X'40'	Open input (read XAB)
	*			
	RDEVHOLD	EQU	X'80'	Close input/output (hold)
	RDEVALL	EQU	X'40'	Close input (all copies)
	RDEVRENQ	EQU	X'20'	Close input (Reenqueue)
	RDEVRTAG	EQU	X'10'	Close input (reenqueue and rewrite SAFTAG)
	RDEVSPHD	EQU	X'08'	Close input (special hold)
	RDEVKEEP	EQU	X'04'	Close input (keep)
	RDEVGOP	EQU	X'80'	Open output (multiple files)
	RDEVOSAV	EQU	X'20'	Open output (save partial files)
	RDEVLOG	EQU	X'10'	Open output (trace file)
	RDEVPUR	EQU	X'40'	Close output (purge)
	RDEVINTA	EQU	X'80'	Openin ta (internal TA)
	RDEVEXTA	EQU	X'40'	Openin ta (external TA)
X'004'	RDEVTAG	DC	A(0)	Address of TAG
X'008'	RDEVFIOA	DC	A(0)	Address of file I/O area
X'00C'	RDEVLINK	DC	A(0)	Address of link table
X'010'	RDEVXAB	DC	A(0)	Address of XAB area
X'014'	RDEVXABL	DC	H(0)	Length of XAB area
X'016'	RDEVSPFL	DC	X'00'	Separator page flags
X'017'	RDEVSPFD	DC	X'00'	Default separator page flags
	*			
	*		Bits Defined in RDEVSPFL and RDEVSPFD	
	*			
	RDEVSPVM	EQU	X'80'	Write VM separator pages
	RDEVSPSH	EQU	X'40'	Write short separator pages
	RDEVSPNO	EQU	X'20'	Write no separator pages
	RDEVSPUS	EQU	X'10'	Write user defined pages
	*			
X'018'	RDEVSPLN	DC	H'0'	Form length (number lines)
X'01A'	RDEVSPWI	DC	H'0'	Form width (number of chars)
X'01C'	RDEVSPPI	DC	H'0'	Lines per inch
X'01E'	RDEVSPLD	DC	H'0'	Default form length
X'020'	RDEVSPWD	DC	H'0'	Default form width
X'022'	RDEVSPPD	DC	H'0'	Default lines per inch
X'024'	RDEVFCBA	DC	A(0)	Address of FCB image to use
	RDEVRDR	EQU	RDEVTAG,8	Address of TAG and FIOA
	*			
	*		Alternate Definition for OPEN TA Request	
	*			
	RDEVSTR	EQU	RDEVSPFL,1	Number of streams
	RDEVATAN	EQU	RDEVTAG,4	Address of transmission algorithm
	RDEVTAP	EQU	RDEVFIOA,4	Transmission algorithm parm addr
	RDEVLEN	EQU	*-RDEVBLOK	Length of request block

Chapter 19. Networking Data Areas and Record Formats

This section describes some of the data areas and equates used by the RSCS networking link drivers. It also describes the format of various NJE records. For more information about NJE records and protocols, see *z/OS: Network Job Entry (NJE) Formats and Protocols* (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf).

Data Areas and Equates

The following sections describe the format of data areas and equates used by the networking link drivers.

BUFFER

The BUFFER macro maps the telecommunication buffer for the networking link drivers. The format of the BUFFER area differs for non-SNA link drivers (DMTNET, DMTLIS) and SNA session drivers (DMTSNE).

Non-SNA Telecommunications Buffer

X'000'	BUFCHAIN	DS	A	Buffer chain field
X'004'	BUFCOUNT	DS	1H	Number of bytes in buffer (data field)
X'006'	BUFSTAT	DS	XL1	Buffer status byte
* Bit defined in BUFSTAT				
* BUFSMALL EQU X'01' Small size buffer				
X'007'	BUFRSVD	DS	XL1	Reserved field
X'008'	BUFBCLN	EQU	*-BUF	Length of buffer control
X'008'	BUFTCTL	DS	XL2	Transmission control bytes (non-SNA)
X'00A'	BUFBCLB	DS	1C	Block control byte (non-SNA)
X'00B'	BUFFCS	DS	XL2	Function control sequence information (non-SNA)
X'00B'	BUFBXLEN	EQU	*-BUFTCTL	Length of non-SNA buffer special data fields
X'00D'	BUFDATA	DS	0C	Data portion of buffer (non-SNA)
X'00D'	BUFBXTRA	EQU	BUFBXLEN+1+2	Length of extra buffer fields, (non-SNA) including end of buffer
* 2 more for "XETBSEQ"(1026).				

SNA Telecommunications Buffer

X'000'	BUFCHAIN	DS	A	Buffer chain field
X'004'	BUFCOUNT	DS	1H	Number of bytes in buffer data field
X'006'	BUFSTAT	DS	XL1	Buffer status byte
* Bits defined in BUFSTAT				
* BUFSMALL EQU X'01' Small size buffer				
X'007'	BUFRSVD	DS	XL1	Reserved field
X'00D'	BUFSDC	ORG	BUFTCTL	
X'008'	BUFSDC	DS	1H	* SNA decompress count (RECEIVE)
X'008'	BUFSCLN	EQU	*-BUF	* Length of SNA control fields
X'00A'	BUFSDATA	DS	0C	* Start of data area for SNA information

HDRTRL

The HDRTRL macro maps a data area that holds an NJE header or trailer while it is stored by RSCS. The length field contains the total length of the header after all transmission segments are combined.

First Segment

+-----+

NCC

X'00'	HDRLEN	HDRRES	HDRGEN
+	-----	-----	-----

Second Segments

```

-----*
* General Mapping of Each Incoming NJE Header/Trailer Transmission Segment. *
-----*

X'000' HDR DSECT
X'002' HDRLEN DS AL2 Total size of header
X'002' HDRRES DS AL2 Reserved
X'004' HDRGEN DS 0F Start of general section
                        (followed by other sections)
*
* SEG DSECT
X'000' SEGLen DS AL2 Length of segment
X'002' SEGFLAGS DS XL1 Flags
X'003' SEGSEQ DS XL1 Sequence indicator
*
* Bit(S) defined in SEGSEQ
*
NOTLAST EQU X'80' "Not last segment" indicator
*
X'004' SEGDATA DS 0F Data part of segment
*

```

NCC

Networking link drivers use the network connection control area (NCC) to build a SIGNON record to send to a remote system.

```

X'000' NCCRID DS 0XL3 SNA record identifier
X'000' NCCRCB DC X'F0' General record control byte
X'001' NCCSRCB DC C'I' Sub-record control byte
*
* Initial signon control record (may be discarded by RTAM)
* Response signon control record
*
* Possibilities for NCCSRCB
*
NCCI EQU C'I' Initial signon character
NCCJ EQU C'J' Response signon character
*
X'002' NCCIDL DC AL1(NCCIL) Length of logical record
X'003' NCCINODE DC CL8' ' Node identification
X'00B' NCCIQAL DC X'01' Qualifier if shared spool
X'00C' NCCIEVNT DC FL4'0' Event sequence number
X'010' NCCIREST DC H'0' Part node to node resistance
X'012' NCCIBFSZ DC H'400' Maximum transmission block size
X'014' NCCILPAS DC CL8' ' Line password
X'01C' NCCINPAS DC CL8' ' Node password
X'024' NCCIFLG DC X'00' Feature flags
*
* Bits set in NCCIFLG
*
NCCIFLGM EQU X'80' Multiple trunk (response)
NCCIOLDL EQU *-NCCRCB
X'025' NCCIFEAT DC AL1 (NCCIPREP+NCCIPACK+NCCIRIF+NCCIMRCB)
                        Signon concurrence features mask
X'026' NCCFEAT1 DC XL3'00' Reserved
                        First byte in signon concurrence mask
NCCFEAT1 EQU NCCIFEAT,1
NCCIL EQU *-NCCRCB Length of element
X'029' NCCIEND DC X'0' End RCB
*
* Bits defined in NCCFEAT1
*
NCCIPREP EQU X'80' Prepare protocol for
                        ... quiescence to be used
NCCIPACK EQU X'10' Supporting packed buffers
NCCIRIF EQU X'08' Supporting RIF omission
NCCIMRCB EQU X'04' Supporting mixed RCBs

```

NJEEQU

The NJEEQU macro establishes the networking equates used by all networking link drivers.

```

*
* SRCBS used in data records
*
NOCC EQU X'80' No carriage control in rec.
PMACH EQU X'90' Machine carriage control
PASA EQU X'A0' ASA carriage control in rec.
SOMECC EQU X'30' Test for machine or ASA or CC
CPDSM EQU X'B0' CPDS (stream mode) record
*
* Spanned record bits in SRCB

```



```

*
FSPAN    EQU    X'08'           First segment of spanned rec
MSPAN    EQU    X'04'           Middle segment of span rec.
LSPAN    EQU    X'0C'           Last segment of spanned rec.

*-----*
*          Abnormal termination by sender          *
*-----*

ABORT     EQU    X'40'           Abort transmission

*-----*
*          Record format bit settings              *
*-----*

RFVAR     EQU    X'40'           Variable format
RFFIX     EQU    X'80'           Fixed format
RFUNDEF   EQU    X'C0'           Undefined format
RFASA     EQU    X'04'           ASA carriage control
RFMC      EQU    X'02'           Machine carriage control

*-----*
*          Masks for stream RCBS and FCS mask checks  *
*-----*

TSYSOUT   EQU    X'01'           SYSOUT bit identifier (for RCB)
MSYSIN    EQU    X'80'           SYSIN mask (for FCS)
MSYSOUT   EQU    X'01'           SYSOUT mask (for FCS)

*-----*
*          Coded NOP codes                          *
*-----*

NJEHDR    EQU    X'01'           Network protocol header code
DATAREC   EQU    X'02'           Data record (non-NJE header)

*-----*
*          SCB equates                              *
*-----*

RIDSCB    EQU    X'03'           SCB used for RID (SNA only)
SNAASIS   EQU    X'00'           SNA noncompressible char. string
SCBASIS   EQU    X'C0'           BSC noncompressible char. string

*-----*
*          CCW operation codes                      *
*-----*

NOP        EQU    X'03'           No operation code
PREP       EQU    X'06'           Prepare code
PUNCHOP    EQU    X'41'           Punch operation code

*-----*
*          I/O completion information               *
*-----*

TO         EQU    X'01'           Time out

*-----*
*          RSCS control block lengths              *
*-----*

LTAGR     EQU    136             Length of user tag record

*-----*
*          Other equates used frequently            *
*-----*

HEX00     EQU    X'00'
HEX01     EQU    X'01'
HEX04     EQU    X'04'
HEX07     EQU    X'07'
HEX10     EQU    X'10'
HEX80     EQU    X'80'
HEX7F     EQU    X'7F'
HEXF0     EQU    X'F0'
HEXFF     EQU    X'FF'
EMPTY     EQU    X'BF'
BLNK      EQU    C' '

```

```

*-----*
* Below are storage allocation lengths used in *
* obtaining storage for various data areas and *
* control blocks used in networking modules.   *
*-----*

CTCBFSZE EQU 4096      Default buffer size for CTCA
SNEBFSZE EQU 1024      SNE default buffer size
TCPBFSZE EQU 4096      TCP default buffer size
BSCBFSZE EQU 800       BSC default buffer size
SMSBFLEN EQU 18        Length of small buffer for SNA
SMBBFLEN EQU 23        Length of small buffer for BSC
SMTBFLEN EQU 23        Length of small buffer for TCP
IJHSIZE  EQU 260       Size of RIB job header BUILD area
*                   ... for incoming job headers
IJTSIZE  EQU 260       Size of RIB job trailer BUILD area
*                   ... for incoming job trailers
IDHSIZE  EQU 516       Size of RIB data set hdr BUILD area
*                   ... for incoming dataset headers

IAXTSIZE EQU 516       Initial size of TIBAXTNK (used
*                   ... for transmitting files)
BMPHSIZE EQU 512       Increment for RIB header BUILD area
JHDHSZ   EQU IJHSIZE*2+IDHSIZE*2 Total size of storage to get in
*                   Initialization for headers in RIBs

```

NMR

Networking link drivers use a nodal message record (NMR) to transmit messages and commands to remote nodes.

```

X'000' NMRFLAG DC X'0'      Flag byte
X'001' NMRLEVEL DC 0X'0'    Importance level (high 4 bits)
X'001' NMRPRIO DC X'0'      Output priority (low 4 bits)
X'002' NMRTYPE DC X'0'      Type byte
X'003' NMRML DC X'0'        Length of message
X'004' NMRT0 DC 0CL9' '    To node
X'004' NMRTONOD DC CL8' '   To node name
X'00C' NMRT0QUL DC X'0'     To node qualifier
X'00D' NMROUT DC XL8'0'     Local output information
X'015' NMRFM DC 0CL9' '    From node
X'015' NMRFMNOD DC CL8' '   From node name
X'01D' NMRFMQUL DC X'0'     From node qualifier
X'01E' NMRMSG DC CL140' '   Message
      NMRL EQU *-NMR
      NMRA EQU NMR,NMRL     Alias for NMR DSECT with length
*
* Formatted command definitions
*
      ORG NMRMSG
X'01E' NMRFNORM DC 0XL20'0' Formatted area for normal command
X'01E' NMRFRTE DC 0XL36'0' Formatted area for route command
X'01E' NMRFOP DC X'0'       Opcode
X'01F' NMRFFLG DC X'0'      Flags or opcode modifier
X'020' NMRFJID DC XL2'0'    Initial job number
X'022' NMRFORGN DC CL8' '   Origin node name
X'02A' NMRFJNAM DC CL8' '   Job name

X'032' NMRFD DC CL8' '      Destination for route command
X'03A' NMRFR DC CL8' '      Remote if not implied by NMRFD
*
* NMROUT format for UCMID messages
*
      ORG NMROUT
X'00D' NMRUCM DC X'0'       MCS console ID
X'00E' NMRUCMA DC X'0'      MCS console area
X'00F' NMRLINET DC XL2'0'   Line type for MLWTO
X'011' DC XL4'0'           Spacer
*
* NMROUT format for logical routed msgs
*
      ORG NMROUT
X'00D' NMRDESC DC XL2'0'    MCS descriptor codes
X'00F' NMRROUT DC XL2'0'    MCS console routings
X'011' NMRDOMID DC XL4'0'   MCS DOM ID
*
* NMROUT format for remote messages
*

```

```

X'00D' NMRRMT   ORG   NMROUT
          DC   CL8' '           Remote name 'RNNN'
*
*   NMROUT format for user messages (NMRFLAGT on and NMRFLAGC off)
*
X'00D' NMRUSER   ORG   NMROUT
          DC   CL8' '           Receiving user ID
*
*   For commands (NMRFLAGC on) and NMRFLAGT on
*   NMRUSER contains the sending user ID
*
*   NMRMSG format if NMRTYPE4 bit is on and NMRFLAGC is off
*
X'01E' NMRECSID  ORG   NMRMSG
          DC   CL8' '           Sending user ID
*
*   NMFLAG definitions
*
NMRFLAGC EQU  B'10000000'      NMRMSG contains a command
NMRFLAGW EQU  B'01000000'      NMROUT has JES2 RMT number
NMRFLAGT EQU  B'00100000'      NMROUT has user ID
NMRFLAGU EQU  B'00010000'      NMROUT has UCMID information
NMRFLAGR EQU  B'00001000'      Console is only remote authorized
NMRFLAGJ EQU  B'00000100'      Console not job authorized
NMRFLAGD EQU  B'00000010'      Console not device authorized
NMRFLAGS EQU  B'00000001'      Console not system authorized
*
*   NMRTYPE definitions
*
NMRTYPEX EQU  B'11110000'      Reserved bits
NMRTYPEP EQU  B'00000001'      DOM (not supported)
NMRTYPEF EQU  B'00000010'      Formatted command in NMRMSG
NMRTYPEE EQU  B'00000010'      Msg text only in NMRMSG
NMRTYPE4 EQU  B'00001000'      Msg text contains control info
*
*   NMRFOP definitions
*
NMRFOPD EQU  1                Display job command
NMRFOPC EQU  2                Cancel job command
NMRFOPA EQU  3                Release job command
NMRFOPH EQU  4                Hold job command
NMRFOPR EQU  5                Route job command
*
*   NMRFFLG definitions
*
NMRFFLGO EQU  X'80'           Cancel or route output
NMRFFLGD EQU  X'40'           Cancel execution with dump

```

RIB

Networking link drivers use receiver information blocks (RIB) when they receive information from a remote node. Each RIB is chained to another by an address in the RIBNEXT field, which contains zeros in the last RIB in the chain. A RIB can vary in length, depending on if it receives a message or a file.

For Messages

X'000'	RIBNEXT	DS	1F	Chain pointer to next RIB
X'004'	RIBBUFF	DS	1F	Buffer assigned to RIB
X'008'	RIBSBUFA	DS	1F	Address of the next record ... in buffer to decompress Pointer to NJEFILE
X'00C'	RIBGPIPT	DS	1F	
	RIBCNTL	DS	0XL2	
X'010'	RIBRCB	DS	XL1	RCB for RIB
X'011'	RIBSRCB	DS	XL1	SRCB for RIB
X'012'	RIBSEQ	DS	XL1	Segseq for RIB
X'013'	RIBTYPE	DS	XL1	RIB type
	*			
	*		Bits defined in RIBTYPE	
	*			
	RIBCONS	EQU	X'80'	Console type processor
	RIBCTLR	EQU	X'40'	Control record processor
	RIBPERM	EQU	X'20'	Stream is permanently open
	*			
X'014'	RIBSTAT	DS	XL1	Status flags
	*			
	*		Bits defined in RIBSTAT	
	*			
	RIBOPEN	EQU	X'80'	Stream open
	RIBRJECT	EQU	X'40'	Receiver cancel(file rejected)
	RIBCKBUF	EQU	X'20'	Check buffer for more data
	RIBCKMOR	EQU	X'10'	Check more buffers for EOF
X'015'	RIBFLAG1	DS	XL1	Flag
	*			
	*		Bits defined in RIBFLAG1	
	*			
	RIBPUNCH	EQU	X'80'	Punch SYSOUT file (if off print)
	RIBSYSIN	EQU	X'40'	SYSIN (job) file
	RIBNOP	EQU	X'20'	Special - store as NOP
	RIBNJSAF	EQU	X'10'	NJE store and forward
	RIBSPIN	EQU	X'08'	Split file
	RIBTAGRB	EQU	X'04'	Tag record indicator
	RIBOPCDJ	EQU	X'02'	OPTCD=J specified
	RIBM971	EQU	X'01'	Msg 971 has been issued
	RIBMLEN	EQU	*-RIB	RIB length for message receiver

For Files

X'000'	RIBNEXT	DS	1F	Chain pointer to next RIB
X'004'	RIBBUFF	DS	1F	Buffer assigned to RIB
X'008'	RIBSBUFA	DS	1F	Address of the next record ... in buffer to decompress Pointer to NJEFILE
X'00C'	RIBGPIPT	DS	1F	
	RIBCNTL	DS	0XL2	
X'010'	RIBRCB	DS	XL1	RCB for RIB
X'011'	RIBSRCB	DS	XL1	SRCB for RIB
X'012'	RIBSEQ	DS	XL1	Segseq for RIB
X'013'	RIBTYPE	DS	XL1	RIB type
	*			
	*		Bits defined in RIBTYPE	
	*			
	RIBCONS	EQU	X'80'	Console type processor
	RIBCTLR	EQU	X'40'	Control record processor
	RIBPERM	EQU	X'20'	Stream is permanently open
	*			
X'014'	RIBSTAT	DS	XL1	Status flags
	*			
	*		Bits defined in RIBSTAT	
	*			
	RIBOPEN	EQU	X'80'	Stream open
	RIBRJECT	EQU	X'40'	Receiver cancel(file rejected)
	RIBCKBUF	EQU	X'20'	Check buffer for more data
	RIBCKMOR	EQU	X'10'	Check more buffers for EOF
	RIBEOF	EQU	X'08'	EOF has been received
X'015'	RIBFLAG1	DS	XL1	Flag
	*			
	*		Bits defined in RIBFLAG1	
	*			
	RIBPUNCH	EQU	X'80'	Punch SYSOUT file (if off print)
	RIBSYSIN	EQU	X'40'	SYSIN (job) file
	RIBNOP	EQU	X'20'	Special - store as NOP
	RIBNJSAF	EQU	X'10'	NJE store and forward
	RIBSPIN	EQU	X'08'	Split file
	RIBTAGRB	EQU	X'04'	Tag record indicator
	RIBOPCDJ	EQU	X'02'	OPTCD=J specified
	RIBM971	EQU	X'01'	Msg 971 has been issued

	RIBMLEN	EQU	*-RIB	RIB length for message receiver
X'016'	RIBFLAG2	DS	XL1	Another flag byte
	*			
	*		Bits defined in RIBFLAG2	
	*			
	RIBVM	EQU	X'80'	RSCS section in data set header
	RIBJOBH	EQU	X'20'	Job header processed
	RIBDSH	EQU	X'10'	Dataset header processed
	RIBJOBTR	EQU	X'08'	Job trailer processed
	RIBFAN	EQU	X'04'	Possible fan out
	RIBHOLD	EQU	X'02'	Indicates file is held
	RIBSKCH1	EQU	X'01'	Ind. skip for skip CH 1
X'017'	RIBFLAG3	DS	XL1	And another one yet
	*			
	*		Bits defined in RIBFLAG3	
	*			
	RIBDEVOP	EQU	X'80'	Device opened had rc=0
	RIBSDSH	EQU	X'40'	Spanned flag data set header
	RIBSJOBH	EQU	X'20'	Spanned flag job header
	RIBSJTRL	EQU	X'10'	Spanned job trailer flag
	RIBMAXUR	EQU	X'08'	We are at max UR level
	RIBDSHCO	EQU	X'04'	Current DSH is a companion
				... dataset header
	RIBDSDMY	EQU	X'02'	Current DSH is a not to be forwarded
	RIBFANOU	EQU	X'01'	DSH is taking fanout link
X'018'	RIBCRTRC	DS	XL1	Current TRC for 3800
X'019'	RIBFCSM	DS	XL1	FCS stream mask
X'01A'	RIBTRC	DS	XL1	Number of TRCs for 3800 file
X'01B'	RIBFLAG4	DS	XL1	More flags
	*			
	*		Bits defined in RIBFLAG4	
	*			
	RIBSELFL	EQU	X'80'	Write out SELECT CCWs for
				... multiple destinations
	RIBKEEPH	EQU	X'40'	Keep headers in file
	RIBDVCH1	EQU	X'20'	RIBDVCHN first device
X'01C'	RIBORGID	DS	XL2	Origin spool ID for messages
X'01E'		DS	XL2	Reserved
X'020'	RIBSPADR	DS	1F	Address of spanned record area
X'024'	RIBJHDR	DS	1F	Address of job header area
X'028'	RIBDSHD	DS	1F	Address of dataset header area
X'02C'	RIBJTRL	DS	1F	Address of job trailer area
X'030'	RIBHWORK	DS	1F	Work area for HDRBUILD
X'03C'		DS	CL4	Reserved
X'038'	RIBUSER	DS	XL8	User field for exit routines
X'040'	RIBFULL	DS	1F	Full word work area
X'044'	RIBXAB	DS	1F	Address of XAB
X'048'	RIBXABL	DS	1H	Length of XAB
X'04A'	RIBSPANS	DS	1H	Size of spanned record area
X'04C'	RIBSPZE1	DS	1H	Size of spanned record
X'04E'	RIBJHSZ	DS	1H	Size of job header area
X'050'	RIBDHSZ	DS	1H	Size of dataset header area
X'052'	RIBJTSZ	DS	1H	Size of job trailer area
X'054'	RIBHALF	DS	1H	Half word work area
X'056'	RIBDVNUM	DS	1H	Number of devices in chain
X'058'	RIBDVCHN	DS	1F	Anchor of device chain
X'05C'	RIBDVSTB	DS	1F	Pointer to stub device
X'060'	RIBPDTAG	DS	(TAGLEN)X	Prototype device tag for RIB
	RIBLEN	EQU	*-RIB	Length of RIB for file processors

TANK

Networking link drivers use TANKS as an intermediate buffer to hold a deblocked output record. The TANK macro maps the data areas for the different forms of the tanks. These forms are described in the following sections.

Unit Record Tank

X'000'	TANKCNT	DS	1H	Count of data bytes in tank
X'002'	TANKRCB	DS	XL1	Tank record control byte
X'003'	TANKSRCB	DS	XL1	Tank sub-record control byte
	TANKCLEN	EQU	*-TANK	Length of tank control information
X'004'	TANKDATA	DS	CL256	Data area in the tank
X'104'	TANKEND	DS	1F	Compression work area
	TANKLEN	EQU	*-TANK	Length of tank

Network SYSOUT Record Tank

X'000'	TANKCNT	DS	1H	Count of data bytes in tank
X'002'	TANKRCB	DS	XL1	Tank record control byte
X'003'	TANKSRCB	DS	XL1	Tank sub-record control byte
	TANKCLEN	EQU	*-TANK	Length of tank control information
X'004'	TANKDATA	DS	CL256	Data area in the tank
		ORG	TANKDATA	
X'004'	TANKLRCL	DS	XL1	Original record length
X'005'	TANKCCTL	DS	XL1	Carriage control
X'006'	TANKTRC	DS	XL1	3800 TRC byte

Network SYSOUT Spanned Record Tanks

First Record Segment

X'000'	TANKCNT	DS	1H	Count of data bytes in tank
X'002'	TANKRCB	DS	XL1	Tank record control byte
X'003'	TANKSRCB	DS	XL1	Tank sub-record control byte
	TANKCLEN	EQU	*-TANK	Length of tank control information
X'004'	TANKDATA	DS	CL256	Data area in the tank
		ORG	TANKDATA	
X'004'	TANKSGL1	DS	CL1	Segment length
X'005'	TANKSRL1	DS	CL2	Total record length
X'007'	TANKSGCC	DS	CL1	Carriage control
X'008'	TANKSGD1	DS	CL252	Segment data

Middle and End Record Segments

X'000'	TANKCNT	DS	1H	Count of data bytes in tank
X'002'	TANKRCB	DS	XL1	Tank record control byte
X'003'	TANKSRCB	DS	XL1	Tank sub-record control byte
	TANKCLEN	EQU	*-TANK	Length of tank control information
X'004'	TANKDATA	DS	CL256	Data area in the tank
		ORG	TANKDATA	
X'004'	TANKSGL2	DS	CL1	Segment length
X'005'	TANKSGD2	DS	CL252	Segment data

TIB

Transmitter information blocks (TIBs) are used for streams sent to remote systems. Each TIB is chained to another by an address in the TIBNEXT field, which contains zeros in the last TIB in the chain. A TIB can vary in length, depending on if receives a message or a file.

For Messages

```

X'000' TIBNEXT DS 1F Chain pointer
X'004' TIBBUFF DS 1F Buffer assigned to TIB
X'008' TIBSBUFA DS 1F Addr. to start next record in buffer
X'00C' TIBUSER DS XL8 User field for exit routines
X'014' TIBXAB DS 1F Address of XAB
X'018' TIBXABL DS 1H Length of XAB
X'01A' TIBREAS DS 1H Receiver cancel reason code
X'01C' TIBGPIPT DS 1F NJEFILE pointer
X'020' TIBTYPE DS XL1 Type flags

```

```

*-----*
*           Bits defined in TIBTYPE           *
*-----*
*
*   NOTE: TIBTYPE is the flag which is not changed for each new file
*
TIBCONS EQU X'80' Console type processor
TIBTHROT EQU X'40' Using message throttle (console TIB only)
TIBPRIN EQU X'20' SYSIN stream permanently opened
TIBPROUT EQU X'10' SYSOUT stream permanently opened

```

```

X'021' TIBRCB DS XL1 RCB for active file
X'022' TIBSTAT DS XL1 Status flags

```

```

*-----*
*           Bits defined in TIBSTAT           *
*-----*
*
TIBINACT EQU X'80' TIB not active
TIBRJECT EQU X'40' Reject perm or receiver cancel
TIBABORT EQU X'20' Abort sent for file
TIBWAITS EQU X'10' Wait-a-bit stream on
TIBWAITR EQU X'08' Waiting for reply
TIBWOSYS EQU X'04' TIB waiting for other systems RECV
TIBUNUSE EQU X'02' TIB is unusable (other side
* ... has issued a permanent reject
TIBTCOMP EQU X'01' Transmission complete

```

```

X'023' TIBFLAG1 DS XL1 Flag1

```

```

*-----*
*           Bits defined in TIBFLAG1         *
*-----*
*
TIBPUNCH EQU X'80' Punch SYSOUT file
TIB3800 EQU X'40' Virtual 3800 SYSOUT file
TIBSYSIN EQU X'20' SYSIN (job) file
TIBSAF EQU X'10' Store and forward file
TIBSAVR EQU X'08' Uncompressed record saved in tank
TIBMSGIN EQU X'04' Message in buffer (CONS TIB only)
TIBPUNCC EQU X'02' PUNCC=YES specified on tag
TIBOPCDJ EQU X'01' OPTCD=J specified on tag

```

```

X'024' TIBTANK DS XL(TANKLEN) Tank for TIB
TIBMLEN EQU *-TIB Length of TIB for message processor

```

For Files

```

X'000' TIBNEXT DS 1F Chain pointer
X'004' TIBBUFF DS 1F Buffer assigned to TIB
X'008' TIBSBUFA DS 1F Addr. to start next record in buffer
X'00C' TIBUSER DS XL8 User field for exit routines
X'014' TIBXAB DS 1F Address of XAB
X'018' TIBXABL DS 1H Length of XAB
X'01A' TIBREAS DS 1H Receiver cancel reason code
X'01C' TIBGPIPT DS 1F NJEFILE pointer
X'020' TIBTYPE DS XL1 Type flags

```

```

*-----*
*           Bits defined in TIBTYPE           *
*-----*
*
*   NOTE: TIBTYPE is the flag which is not changed for each new file
*
TIBCONS EQU X'80' Console type processor
TIBTHROT EQU X'40' Using message throttle (console TIB only)
TIBPRIN EQU X'20' SYSIN stream permanently opened
TIBPROUT EQU X'10' SYSOUT stream permanently opened

```

```

X'021' TIBRCB DS XL1 RCB for active file
X'022' TIBSTAT DS XL1 Status flags

```

```

*-----*
*           Bits defined in TIBSTAT           *
*-----*
*

```

	TIBINACT	EQU	X'80'	TIB not active
	TIBRJECT	EQU	X'40'	Reject perm or receiver cancel
	TIBABORT	EQU	X'20'	Abort sent for file
	TIBWAITS	EQU	X'10'	Wait-a-bit stream on
	TIBWAITR	EQU	X'08'	Waiting for reply
	TIBWOSYS	EQU	X'04'	TIB waiting for other systems RECV
	TIBUNUSE	EQU	X'02'	TIB is unusable (other side
	*			... has issued a permanent reject
	TIBTCOMP	EQU	X'01'	Transmission complete
X'023'	TIBFLAG1	DS	XL1	Flag1

	* Bits defined in TIBFLAG1 *			

	TIBPUNCH	EQU	X'80'	Punch SYSOUT file
	TIB3800	EQU	X'40'	Virtual 3800 SYSOUT file
	TIBSYSIN	EQU	X'20'	SYSIN (job) file
	TIBSAF	EQU	X'10'	Store and forward file
	TIBSAVR	EQU	X'08'	Uncompressed record saved in tank
	TIBMSGIN	EQU	X'04'	Message in buffer (CONS TIB only)
	TIBPUNCC	EQU	X'02'	PUNCC=YES specified on tag
	TIBOPCDJ	EQU	X'01'	OPTCD=J specified on tag
X'024'	TIBTANK	DS	XL(TANKLEN)	Tank for TIB
	TIBMLEN	EQU	**TIB	Length of TIB for message processor
X'12C'	TIBTANKX	DS	XL4	Extra for length of 260 byte FCBS
	*			
X'130'	TIBAXTNK	DS	1F	Address of auxiliary tank
X'134'	TIBSTRM	DS	XL1	NJI stream number
X'135'	TIBTRCV	DS	XL1	Current TRC value for 3800
X'136'	TIBFCSM	DS	XL1	FCSMASK for active file
X'137'	TIBFLAG2	DS	XL1	Flag2

	* Bits defined in TIBFLAG2 *			

	TIBFIRST	EQU	X'80'	First entry to transmitter
	TIBFLUSH	EQU	X'40'	FLUSH command issued for file
	TIBNOCC	EQU	X'20'	SYSOUT file - send without carriage control
	TIBFIX	EQU	X'10'	Fix length records
	TIBEOF	EQU	X'08'	EOF sent
	TIBXABA	EQU	X'04'	Valid XAB indicator
	TIBWPERM	EQU	X'02'	Awaiting permission to RIF
	TIBALTMV	EQU	X'01'	Record in aux tank
X'138'	TIBFLAG3	DS	XL1	Flag byte 3

	* Bits defined in TIBFLAG3 *			

	* TIBFLAG3 is not reset until file is closed *			
	TIBFHD	EQU	X'80'	Hold file at close time
	TIBRENQ	EQU	X'40'	Reenqueue the file at close time
	TIBGWJH	EQU	X'20'	Job Header sent on GPI
	TIBGWDSH	EQU	X'10'	Dataset Header sent on GPI
	TIBGWJT	EQU	X'08'	Job Trailer sent on GPI
	TIBINTRC	EQU	X'04'	TRC's MUST BE SENT
	TIBHAVE	EQU	X'02'	Have record on GPI
	TIBTAG1	EQU	X'01'	Have looked at first TAG
	*			record of file
X'139'		DS	XL1	Reserved
X'13A'	TIBLFLAG	DS	XL1	Flag byte for *LIST process

	* Bits defined in TIBLFLAG *			

	TIBLISTP	EQU	X'80'	TIB pertains to *LIST link
	TIBDISTL	EQU	X'40'	In process of handling unprocessed headers
	TIBLSAF	EQU	X'20'	This is really a S&F file
	TIBLDHDR	EQU	X'10'	Headers already generated
	TIBLNOCN	EQU	X'08'	Remember no counter allowed
	TIBLDLON	EQU	X'04'	Remember we did headers

	TIBLSYSI	EQU	X'02'	Imbedded SYSIN hdrs in file
	TIBLJTRL	EQU	X'01'	Already processed trailer
X'13B'	TIBFLEN	DS	XL1	Fixed length for file
X'13C'		DS	6X	Reserved
X'142'	TIBAXSZ	DS	1H	Size of auxiliary tank
X'144'	TIBSPID	DS	CL4	EBCDIC spool ID of active file
X'148'	TIBCOSTV	DS	F	Cost value for stream disp
	*			
	*			The following fields are intended for the list processing function
	*			
X'14C'	TIBLRECS	DS	F	Number of records to skip
	*			... until next *LIST entry
	TIBLSSAV	EQU	TIBLRECS,4	Use for work area, also
X'150'	TIBTRECS	DS	F	Number of records for trailer
X'154'	TIBDSANC	DS	F	Anchor for dataset headers
X'158'	TIBFREDS	DS	F	Anchor for free elements
X'15C'	TIBLTLEN	DS	H	Prospective length of DSH
X'15E'	TIBAXLSV	DS	H	Use this for TNKLEN savearea
X'160'	TIBAXTSV	DS	F	Use this for AUXTNK savearea
X'164'	TIBOVDSH	DS	F	Address of overflow DSH
X'168'	TIBOVTNK	DS	F	Pointer to temporary tank
X'16C'	TIBRMBER	DS	F	Remember the recs in DSH
X'170'	TIBOVSIK	DS	H	Size of overflow DSH
	*			
	*			The following fields are intended to facilitate
	*			implementation of neighborly behavior
	*			
X'172'	TIBLDSNO	DS	H	Dataset Header number
X'174'	TIBSNDSDH	DS	F	Pointer to area containing
	*			..list of dshs currently sent
	TIBLSTAG	EQU	TIBSNDSDH,4	Also doubles as area with
	*			*LIST TAG slot prototype
X'178'	TIBSNDNO	DS	H	Number of dshs being sent
X'17A'	TIBNGFLG	DS	AL1	Flag for neighborly behavior

	*	Bits defined in TIBNGFLG		*

	*			
	TIBNDOIT	EQU	X'80'	Do neighborly behavior
	TIBNSKIP	EQU	X'40'	Skipping datasets mode
	TIBNSDSH	EQU	X'20'	Sent DSH but waiting for matching data
	TIBNSDAT	EQU	X'10'	Sending data to matching DSH
	TIBNSKDS	EQU	X'08'	Skipped a DSH, re-enqueue
	TIBCUSEC	EQU	X'04'	Current DSH is a companion
	*			
X'17B'	TIBSFLAG	DS	AL1	Flag that S&F DSH status

	*	Bits defined in TIBSFLAG		*

	*			
	TIBGENER	EQU	X'80'	General Section Processed
	TIBVMSEC	EQU	X'40'	RSCS section processed
	TIBS3800	EQU	X'20'	3800 Section processed
	TIBSOTHR	EQU	X'10'	Other sections present
	TIBOVFLO	EQU	X'08'	Sending distlist overflow
X'17C'	TIBAGENE	DS	F	Address of GENERAL Section
X'180'	TIBAVMSE	DS	F	Address of RSCS section
X'184'	TIBA3800	DS	F	Address of 3800 section
X'188'	TIBAOTHR	DS	F	Address of other sections
X'18C'	TIBREGS	DS	7F	Register SAVEAREA
X'1A8'	TIBNPAGE	DS	F	Number of 'begin page' sent
X'1AC'	TIBNBYTE	DS	F	Number of bytes sent
	*	RDR	DSECT=NO,LABELS=YES	RDR parm block for TIB
X'1B0'	TIBRDR	RDR	DSECT=NO,LABELS=YES	RDR parm block for TIB
	*			
	*	Spool input table		
	*			
X'1B0'	RDRTAG	DC	A(0)	Address of input tag
X'1B4'	RDRFIOA	DC	A(0)	Address of file I/O area
X'1B8'	RDRLINK	DC	A(0)	Addr of link table (non-NJE)
X'1BC'	RDRSPNXT	DC	A(0)	Next CCW in page buffer
X'1C0'	RDRSPNUM	DC	A(0)	No. of recs in page buffer
X'1C4'	RDRDAREA	DC	A(0)	Addr of default input area
X'1C8'	RDRAREA	DC	A(0)	Address of input area
X'1CC'	RDRLRECL	DC	AL2(0)	Logical record length
X'1CE'	RDRMAX	DC	AL2(0)	Max logical record length
X'1D0'	RDRCCWOP	DC	AL1(0)	CCW opcode

XABHDR

X'1D1'	RDRIFLG2	DC	AL1(0)	Second input flag
	*			
	*		Bits defined in RDRIFLG2	
	*			
	RDRSPECO	EQU	X'80'	Special case to OPTIMIZE
	RDRCALWT	EQU	X'40'	Caller wants to wait for *SPL ... reads to complete
X'1D2'	RDRMAX	DC	AL2(0)	Previous max record length
X'1D4'	RDRIFLG	DC	AL1(0)	Input flags
	*			
	*		Bits defined in RDRIFLG	
	*			
	RDRFILLD	EQU	X'80'	Spool buffer present in FIOA
	RDRNOP	EQU	X'40'	Tag rec converted for real reader
	RDRDCIP	EQU	X'20'	Data chained CCW process in progress
	RDROPT	EQU	X'10'	CCW optimization indicator
	RDRUNTRN	EQU	X'08'	Original untruncated length
	RDRWSIN	EQU	X'04'	Workstation input
	RDRLDFCB	EQU	X'02'	3211-Type load FCB acceptable
	RDREOF	EQU	X'01'	EOF encountered on last read
X'1D5'	RDRNFLG	DC	AL1(0)	Output flags
	*			
	*		Bits defined in RDRNFLG	
	*			
	IGNORNXT	EQU	X'80'	Data chained and at max
	DCMRGAHD	EQU	X'40'	Look to merge a CD CCW
	OVOPT	EQU	X'20'	Override printer (RJE)
	RDR1SPLK	EQU	X'08'	1st splk of file indicator
	RDRALT	EQU	X'04'	Alternate input area in use
	RDRDATAS	EQU	X'02'	Data encountered in spool buffer
	RDRIMCMD	EQU	X'01'	CCW is an immediate command
	RDRRDR	EQU	RDRTAG,8	Address of tag and FIOA
X'1D6'	RDRCOUNT	DC	AL2(0)	CCW Data count
	RDRLEN	EQU	*-TIBRDR	Length of RDRPARMS
X'1D8'	TIBTAGR	DS	CL(LTAGR)	Contains the tag record
X'260'	DMT01200	DS	0H	Command origin (half-word aligned)
X'260'	TIBCQUAL	DC	AL1(0)	Origin qualifier
X'261'	TIBCFLAG	DC	X'00'	Flags (bits as MSGBFLAG)
X'262'	TIBCRSPC	DC	H'0'	Response counter
X'264'	TIBCNODE	DC	CL8' '	Origin node
X'26C'	TIBCUSER	DC	CL8' '	and userid
X'274'	TIBCSIG	DC	CL6' '	Response signature
	TIBCORIG	EQU	DMT01020,*-DMT01020	Symbol and length of whole thing
X'27C'		DS	0F	Pad to multiple of fullword
	TIBLEN	EQU	*-TIB	Length of TIB

XABHDR

The Print Services Facility™/VM (PSF) uses an external attribute buffer (XAB) when it processes output files that are destined for an all-points-addressable printer.

If a file contains an Output Processing Section or a nonblank PRMODE indicator (field NDHGPMDE in the General Section of the Data Set Header), an XAB must be associated with it. RSCS creates a header for the XAB; its format is shown below. See [z/VM: CP Programming Services](#) for more information about the XAB format.

X'000'	XABDATL	DS	XL2	Length of entire XAB
X'002'	XABRSVD	DS	XL2	Reserved
X'004'	XABHDRL	DS	XL2	Length of XAB header
X'006'	XABHDRV	DS	0XL34	Data area within XAB header

XABDATL

Contains the length of the entire block (the header and data for this XAB section, including the reserved field and two-byte length field).

XABHDRL

Contains the length of the header and the two-byte header length field.

XABHDRV

Contains the header PSF places into the front of the XAB. The value of that header is: IBM - PSF/VM
- TEXT UNITS BLOCK - LEVEL 0.0.0

XAB data

Contains the variable length data contents of the XAB as created by PSF.

NJE Header Formats

The following sections describe the format of NJE job headers, data set headers, and job trailers. The NHDTR macro creates mapping DSECTs for NJE header, trailer, and data set headers. The following pages describe the format of each NJE header record created by RSCS.

```

*-----*
*      Type Codes Used for Subsystem Sections and Special      *
*      Sections in All Headers and Trailers                    *
*-----*

NTYPGEN EQU X'00'      General section
NTYPSUB EQU X'80'      Subsystem section
NTYPASP EQU X'81'      ASP      subsystem section
NTYPHASP EQU X'82'      HASP      subsystem section
NTYPJES1 EQU X'83'      JES/RES subsystem section
NTYPJES2 EQU X'84'      JES2      subsystem section
NTYPJES3 EQU X'85'      JES3      subsystem section
NTYPPWR EQU X'86'      POWER/VS subsystem section
NTYPVNET EQU X'87'      RSCS      subsystem section
NTYPDSTM EQU X'89'      Data stream section (data set header only)
NTYPTACT EQU X'89'      Accounting section (job trailer only)
NTYPJHSC EQU X'8A'      Scheduling section (job header only)
NTYPUSE EQU X'C0'      User section

```

Job Header Format

```

NJH      DSECT      Network job header record
*
*      Block control information
*
X'000' NJHLEN DC AL2(NJHLEN)      Length of entire block
X'002' NJHFLAGS DC X'00'          Flags
X'003' NJHSEQ DC BL.1'0',AL.7(0)  Transmission sequence indicator
      NJHLBCI EQU *-NJH          Length of block control information
*
*      General Section
*
X'004' NJHG DS 0F                Start of general section

```

X'004'	NJHGLLEN	DC	AL2(NJHGLLEN)	Length of general section
X'006'	NJHGFLGS	DS	0XL2	Section type flags
X'006'	NJHGTYPE	DC	AL1(NTYPGEN)	ID for general section
X'007'	NJHGMOD	DC	AL1(NJHG\$MOD)	Modifier
	NJHG\$MOD	EQU	X'00'	Value of modifier
X'008'	NJHGJID	DC	Y(0)	Job identifier
X'00A'	NJHGJCLS	DC	C'A'	Job class
X'00B'	NJHGMCLS	DC	C'A'	Message class
X'00C'	NJHGFLG1	DC	X'00'	Flags
X'00D'	NJHGPRIO	DC	AL1(0)	Selection priority
X'00E'	NJHGORGQ	DC	AL1(0)	Origin node system qualifier
X'00F'	NJHGJCPY	DC	AL1(0)	Job copy count
X'010'	NJHGLNCT	DC	AL1(0)	Job line count
X'011'		DC	X'00'	Reserved
X'012'	NJHGOPS	DC	AL2(0)	Hop count
X'014'	NJHGACCT	DC	CL8'	Networking account number
X'01C'	NJHGJNAM	DC	CL8'	Job name
X'024'	NJHGUSID	DC	CL8'	Originating user ID
X'02C'	NJHGPASS	DS	CL8	Password
X'034'	NJHGNPAS	DS	CL8	New password
X'03C'	NJHGETS	DC	FL8'0'	Entry time/date stamp
X'044'	NJHGORGX	DC	CL8'	Origin node name
X'04C'	NJHGORGX	DC	CL8'	Origin remote name
X'054'	NJHGXEQN	DC	CL8'	Execution node name
X'05C'	NJHGXEQU	DC	CL8'	Execution user ID (VM)
X'064'	NJHGPRTN	DC	CL8'	Default print node name
X'06C'	NJHGPRTR	DC	CL8'	Default print remote name
X'074'	NJHGPUNN	DC	CL8'	Default punch node name
X'07C'	NJHGPUNR	DC	CL8'	Default punch remote name
X'084'	NJHGFORM	DC	CL8'	Job forms
X'08C'	NJHGICRD	DC	F'0'	Input card count
X'090'	NJHGETIM	DC	F'0'	Estimated execution time
X'094'	NJHGETLN	DC	F'0'	Estimated output lines
X'098'	NJHGETCD	DC	F'0'	Estimated output cards
X'09C'	NJHGPRGN	DC	CL20'	Programmer's name
X'0B0'	NJHGR00M	DC	CL8'	Programmer's room number
X'0B8'	NJHGDEPT	DC	CL8'	Programmer's department
X'0C0'	NJHGBLDG	DC	CL8'	Programmer's building number
X'0C8'	NJHGNREC	DC	F'0'	Record count on output xmission
X'0CC'	NJHGEND	DS	0F	End of general section
	NJHGLLEN	EQU	*-NJHG	Length of general section
	NJHLLN	EQU	*-NJH	Length of entire block
* Recommended format for a user section				
*				
X'0CC'	NJHU	DS	0F	Start of user section
X'0CC'	NJHULEN	DC	AL2(NJHULEN)	Length of user section
X'0CE'	NJHUFLGS	DS	0BL2	Section type flags
X'0CE'	NJHUTYPE	DC	AL1(NTYPUSE)	ID for user section --
* Bits 0-1 must be B'11'				
* Bits 2-7 can be anything				
X'0CF'	NJHUMOD	DC	AL1(NJHU\$MOD)	Modifier --
	NJHU\$MOD	EQU	X'00'	Mod value can be anything
X'0D0'	NJHUCODE	DC	CL4'	SHARE/GUIDE installation code
* Place user information fields				
* between 'NJHUCODE' & 'NJHUEND'				
X'0D4'	NJHUEND	DS	0F	End of user section
	NJHULEN	EQU	*-NJHU	Length of user section
* Bits defined in general section, NJHGFLG1				
*				
	NJHGF1PR	EQU	X'80'	Do not recompute priority
	NJHGF1JN	EQU	X'40'	NJHGJID field is set
	NJHGF1CF	EQU	X'08'	Suppress forwarding msg
	NJHGF1CA	EQU	X'04'	Suppress acceptance msg

Job Trailer Format

	NJT	DSECT		Network job trailer record
X'000'	NJTLEN	DC	AL2(NJTLEN)	Length of entire block
X'002'	NJTFLAGS	DC	X'00'	Flags
X'003'	NJTSEQ	DC	BL.1'0',AL.7(0)	Transmission sequence indicator
	NJTLBCI	EQU	*-NJT	Length of block control information
	*			
	*	General section		
	*			
X'004'	NJTG	DS	0F	Start of general section
X'004'	NJTGLLEN	DC	AL2(NJTGLLEN)	Length of general section
X'006'	NJTGFLGS	DS	0XL2	Section type flags
X'006'	NJTGTYP	DC	AL1(NTYPGEN)	ID for general section
X'007'	NJTGMOD	DC	AL1(NJTG\$MOD)	Modifier
	NJTG\$MOD	EQU	X'00'	Value of modifier
X'008'	NJTGFLG1	DC	X'00'	Flags
X'009'	NJTGXCLS	DC	C'A'	Actual execution class
X'00A'		DC	XL2'0'	Reserved
X'00C'	NJTGSTRT	DC	FL8'0'	Execution start time/date
X'014'	NJTGSTOP	DC	FL8'0'	Execution stop time/date
X'01C'	NJTGACPU	DC	F'0'	Actual CPU time
X'020'	NJTGALIN	DC	F'0'	Actual output lines
X'024'	NJTGACRD	DC	F'0'	Actual output cards
X'028'	NJTGEXCP	DC	F'0'	EXCP count
X'02C'	NJTGI XPR	DC	AL1(0)	Initial XEQ selection priority
X'02D'	NJTGA XPR	DC	AL1(0)	Actual XEQ selection priority
X'02E'	NJTGIOPR	DC	AL1(0)	Initial output selection priority
X'02F'	NJTGAOPR	DC	AL1(0)	Actual output selection priority
X'030'	NJTGEN	DS	0F	End of general section
	NJTGLLEN	EQU	*-NJTG	Length of general section
	*			
	*	Recommended format for an accounting section		
	*			
X'030'	NJT	DS	0F	Start of accounting section
	NJTSLLEN	DC	AL2(NJTSLLEN)	Length of accounting section
	NJTSLFSG	DS	0XL2	Section type flags
X'032'	NJTSTYP	DC	AL1(NTYPTACT)	ID for accounting section
X'033'	NJTSMOD	DC	AL1(NJT\$MOD)	Modifier
	NJT\$MOD	EQU	X'00'	Value of modifier
X'034'	NJTAPAG	DC	F'0'	Page data page count
X'038'	NJTSABYT	DC	F'0'	Number of bytes transmitted
	NJTSEND	DS	0F	End of accounting section
	NJTLLLEN	EQU	*-NJT	Length of entire block
	*			
	*	Recommended format for a user section		
	*			
X'03C'	NJTU	DS	0F	Start of user section
X'03C'	NJTULLEN	DC	AL2(NJTULLEN)	Length of user section
X'03E'	NJTUFLGS	DS	0XL2	Section type flags
X'03E'	NJTUTYPE	DC	AL1(NTYPUSE)	ID for user section --
				Bits 0-1 must be B'11'
				Bits 2-7 can be anything
X'03F'	NJTUMOD	DC	AL1(NJTU\$MOD)	Modifier --
	NJTU\$MOD	EQU	B'00'	Mod value can be anything
X'040'	NJTUCODE	DC	CL4' '	SHARE/GUIDE installation code
				Place user information fields
				between 'NJTUCODE' & 'NJTUEND'
X'044'	NJTUEND	DS	0F	End of user section
	NJTULLEN	EQU	*-NJTU	Length of user section

Data Set Header Format

	NDH	DSECT		Network data set header record
X'000'	NDHLEN	DC	AL2(NDHLEN)	Length of entire block
X'002'	NDHFLAGS	DC	X'00'	Flags
X'003'	NDHSEQ	DC	BL.1'0',AL.7(0)	Transmission sequence indicator
	NDHLBCI	EQU	*-NDH	Length of block control information
	*			
	*	General section		
	*			
X'004'	NDHG	DS	0F	Start of general section
X'004'	NDHGLEN	DC	AL2(NDHGLEN)	Length of general section
X'006'	NDHGFLGS	DS	0XL2	Section type flags
X'006'	NDHGTYP	DC	AL1(NTYPGEN)	ID for general section
X'007'	NDHGMOD	DC	AL1(NDHG\$MOD)	Modifier
	NDHG\$MOD	EQU	B'00000000'	Value of modifier
X'008'	NDHGNODE	DC	CL8' '	Destination node name
X'010'	NDHGRMT	DC	CL8' '	Destination remote name
X'018'	NDHGPROC	DC	CL8' '	PROC invocation name
X'020'	NDHGSTEP	DC	CL8' '	Step name
X'028'	NDHGDD	DC	CL8' '	DDNAME
X'030'	NDHGDSNO	DC	H'0'	Data set number
X'032'	NDHGSEC	DC	AL1(0)	Security level
X'033'	NDHGCLAS	DC	C'A'	Output class
X'034'	NDHGNREC	DC	F'0'	Record count

NJE Data Set Header

```

X'038' NDHGFLG1 DC    X'00'          Flags
*
*          Bits defined in general section, NDHGFLG1
*
NDHGF1SP EQU    X'80'          Spin data set
NDHGF1HD EQU    X'40'          Hold data set at destination
NDHGF1LG EQU    X'20'          Job log indicator
NDHGF1OV EQU    X'10'          Page overflow indicator
NDHGF1IN EQU    X'08'          Punch interpret indicator

X'039' NDHGRCFM DC    X'00'          RECFM
X'03A' NDHGLREC DC    H'0'          Max logical record length
X'03C' NDHGDSCT DC    AL1(1)        Data set copy count
X'03D' NDHGFCBI DC    AL1(0)        3211 FCB index
X'03E'          DC    XL2'00'        Reserved
X'040' NDHGFORM DC    CL8' '        Forms ID
X'048' NDHGFCB  DC    CL8' '        FCB ID
X'050' NDHGUCS  DC    CL8' '        UCS ID
X'058' NDHGXWTR DC    CL8' '        External writer ID
X'060' NDHGDESU DC    CL8' '        Reserved
X'068' NDHGFLG2 DC    X'00'          Second flag byte
*
*          Bits defined in general section, NDHGFLG2
*
NDHGF2PR EQU    X'80'          Dataset is to be printed
NDHGF2PU EQU    X'40'          Dataset is to be punched
NDHGF2NM EQU    X'20'          File name/type can be taken
*          ... from NDHGPROC/STEP

X'069' NDHGUCS0 DC    X'00'          UCS option byte
*
*          Bits defined in general section, NDHGUCS0
*
NDHGUCS0 EQU    X'80'          Block data check option
NDHGUCSF EQU    X'40'          UCS fold option

X'06A'          DC    XL2'00'        Reserved
X'06C' NDHGPMD E DC    CL8' '        Process mode specified by user
X'074' NDHGEND  DS    0F           End of general section
      NDHGLLEN EQU    *-NDHG        Length of general section
      NDHLLN  EQU    *-NDH          Length of entire block
*
*          RSCS subsystem section
*
X'074' NDHV     DS    0F           Start of RSCS section
X'074' NDHVLEN DC    AL2(NDHVLEN)   Length of RSCS section
X'076' NDHVFLGS DS    0XL2          Section type flags

X'076' NDHVTYPE DC    AL1(NTYPVNET) ID for RSCS section
X'077' NDHVMOD  DC    AL1(NDHV$MOD) Modifier
      NDHV$MOD EQU    X'00'        Value of modifier
X'078' NDHVFLG1 DC    X'00'        Flags
X'079' NDHVCLAS DC    C'A'         VM/CP spool file class
X'07A' NDHVDEV  DC    X'00'        VM/CP origin device type
X'07B' NDHVPGLE DC    X'00'        VM/CP virt 3800 page length
X'07C' NDHVDIST DC    CL8' '        VM/CP distribution code
X'084' NDHVFNAM DC    CL12' '       VM/CP file name
X'090' NDHVFTYP DC    CL12' '       VM/CP file type
X'09C' NDHVPRIO DC    AL2(0)        VM/CP transmission priority
X'09E' NDHVVRSN DC    X'00'        RSCS version number of
*          system creating header
X'09F' NDHVRELN DC    X'00'        RSCS release number of
*          system creating header
X'0A0' NDHVTAGR DC    CL136' '      User supplied tag record
X'128' NDHVD SNO DC    AL2(0)        List proc dataset counter
X'12C' NDHVEND  DS    0F           End of RSCS section
      NDHVLEN EQU    *-NDHV        Length OF RSCS section
*
*          Bits defined in RSCS section, NDHVFLG1
*
NDHVLIST EQU    X'80'          File created by *LIST processor
NDHVFIRS EQU    X'40'          First DSH of its kind
NDHVPERS EQU    X'20'          Personalized section
NDHVF1CF EQU    X'08'          Suppress forwarding msgs
NDHVF1CA EQU    X'04'          Suppress acceptance msgs
*
* NOTE: The following two flags are only for use within the
*       list processor and must not be forwarded.
*
NDHVF1SP EQU    X'02'          Suspend all active datasets
*          and open an overflow dataset

```

	NDHVF1RS	EQU	X'01'	Resume all suspended output
	*			
	*	3800	printer characteristics general section (optional)	
	*			
X'12C'	NDHA	DS	0F	Start of 3800 char section
X'12C'	NDHALEN	DC	Y(NDHALLEN)	Length of 3800 char section
X'12E'	NDHAFLGS	DS	0XL2	Flags and modifier
X'12E'	NDHATYPE	DC	AL1(NTYPGEN)	ID for general section
X'12F'	NDHAMOD	DC	AL1(NDHA\$MOD)	Modifier
	NDHA\$MOD	EQU	X'80'	Value of modifier (3800 char)
X'130'	NDHAFLG1	DC	X'00'	Flags
	*			
	*	Bits defined in 3800 characteristics general section, NDHAFLG1		
	*			
	NDHAF1J	EQU	X'80'	'OPTCD=J' specified
	NDHAF1BR	EQU	X'40'	'BURST=YES' specified
	NDHAF1BN	EQU	X'20'	'BURST=NO' specified
	NDHAF1BD	EQU	X'00'	Take default burst setting
	*			
X'131'	NDHAFLCT	DC	AL1(0)	Flash count
X'132'	NDHATREF	DC	X'00'	Table reference character
X'133'		DC	X'00'	Reserved
X'134'	NDHATAB1	DC	CL8' '	Translate table 1
X'13C'	NDHATAB2	DC	CL8' '	Translate table 2
X'144'	NDHATAB3	DC	CL8' '	Translate table 3
X'14C'	NDHATAB4	DC	CL8' '	Translate table 4
X'154'	NDHAFLSH	DC	CL8' '	Flash cartridge ID
X'15C'	NDHAMODF	DC	CL8' '	Copy modification ID
X'164'	NDHACPYG	DC	XL8'00'	Copy groups
X'16C'	NDHAEND	DS	0F	End of 3800 char section
	NDHALLEN	EQU	*-NDHA	Length of 3800 char section
	*			
	*	Data stream characteristics section		
	*			
X'16C'	NDHS	DS	0F	Start of data stream section
X'16C'	NDHSLEN	DC	AL2(NDHSLEN1)	Length of data stream section
X'16E'	NDHSFLGS	DS	0BL2	Flags and modifiers
X'16E'	NDHSTYPE	DC	AL1(NTYPDSTM)	ID for stream section
X'16F'	NDHSMOD	DC	AL1(NDHS\$OUT)	Modifier
	NDHS\$OUT	EQU	B'00000000'	Value of modifier (output SWBS)
X'170'	NDHSFLEN	DC	Y(NDHSLEN1)	Subsection fixed length
X'172'	NDHSFLG1	DC	X'00'	Data stream flag
	*			
	*	Bits defined in stream section, NDHSFLG1		
	*			
	NDHSCPDS	EQU	X'80'	Data set contains at least one CPDS record
	*			
X'173'		DC	X'00'	Reserved
X'174'	NDHSJDVT	DC	XL8'00'	JDVT name
X'17C'	NDHSNSTR	DC	XL4'00'	Page data page count
X'180'	NDHSGPID	DC	XL8'00'	Output name for data set
	NDHSLEN1	EQU	*-NDHS	Length of fixed data stream
X'188'	NDHSOPTB	DS	0H	Start of prefix area
X'188'	NDHSPRID	DC	CL4'SJPF'	Prefix identifier
X'18C'	NDHSVERS	DC	X'02'	Version of prefix
X'18D'	NDHSPLEN	DS	AL1(NDHSLEN2)	Length of prefix = NDHSLEN2
X'18E'	NDHSDLEN	DS	XL2	Length of variable section
X'190'	NDHSVERB	DC	CL8'OUTPUT'	Constant
X'198'	NDHSVRBL	DS	CL8	Diagnostic field
X'1A0'	NDHSFLG2	DS	XL1	Second flag byte
	NDHSCONT	EQU	X'80'	Indicates this is a continuation of the previous OPTB
	*			
X'1A1'	NDHSPARM	DS	XL1	Processed fields byte
X'1A2'		DS	XL2	Reserved
	NDHSLEN2	EQU	*-NDHSOPTB	Length of prefix area
	NDHSTXTU	EQU	*	Start of text unit area
	*			
	*	Record characteristics change general section		
	*			
X'1A4'	NDHC	DS	0F	Start of char change general section
X'1A4'	NDHCLEN	DC	AL2(NDHCLEN)	Length of char change general sect
X'1A6'	NDHCFGLS	DS	0XL2	Section type flags
X'1A6'	NDHCTYPE	DC	AL1(NTYPGEN)	ID for general section
X'1A7'	NDHCMOD	DC	AL1(NDHC\$MOD)	Modifier
	NDHC\$MOD	EQU	X'40'	Value of modifier (char change)
X'1A8'	NDHCFGL1	DC	X'00'	Flags
X'1A9'	NDHCRCFM	DC	X'00'	RECFM
X'1AA'	NDHCLREC	DC	AL2(0)	Maximum LRECL
X'1AC'	NDHCEND	DS	0F	End of char change general section
	NDHCLEN	EQU	*-NDHC	Length of char change general sect

Record Formats

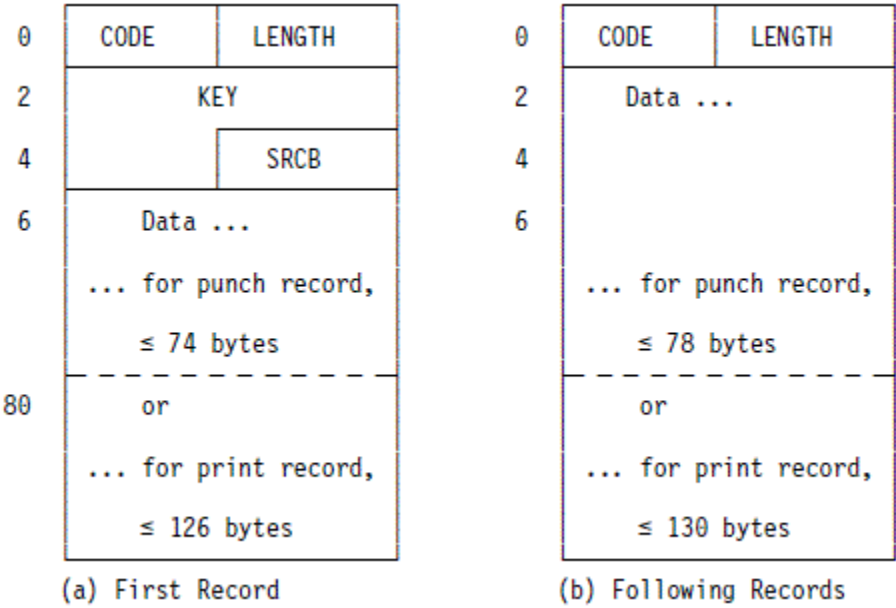
* * Recommended format for a user section *				
X'1AC'	NDHU	DS	0F	Start of user section
X'1AC'	NDHULEN	DC	AL2(NDHULEN)	Length of user section
X'1AE'	NDHUFLGS	DS	0XL2	Section type flags
X'1AE'	NDHUTYPE	DC	AL1(NTYPUSER)	ID for user section -- Bits 0-1 must be B'11' BITS 2-7 can be anything
X'1AF'	NDHUMOD	DC	AL1(NDHU\$MOD)	Modifier --
	NDHU\$MOD	EQU	X'00'	Mod value can be anything
X'1B0'	NDHUCODE	DC	CL4' '	SHARE/GUIDE installation code place user information fields between 'NDHUCODE' & 'NDHUEND'
X'1B4'	NDHUEND	DS	0F	End of user section
	NDHULEN	EQU	*-NDHU	Length of user section

Record Formats

The following sections describe the format of NOP, spanned, and segmented records.

Coded NOP Records

RSCS uses coded NOPs to place certain records from store-and-forward files into CP spool. The link drivers call DMTNUSCN to produce the Coded NOP records.



CODE

Is a 1-byte code that identifies the record:

- X'01'
Network headers
- X'02'
All others
- X'80'
Last segment

LENGTH

Is the length of data in this segment.

KEY

Is a 3-byte field, randomly generated, that matches the contents of the TAGKEY field in the TAG record. It identifies the record as a Coded NOP record.

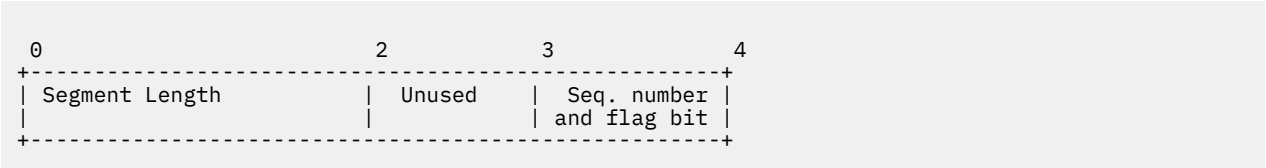
SRCB

Is the SRCB with which the record was originally received.

Segmented Header Formats

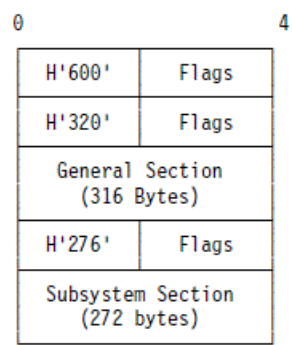
NJE headers for the DMTNET link driver and the DMTSNE session driver can be up to 32,767 bytes long. Headers longer than 256 bytes are transmitted as segmented headers, with each segment a maximum of 256 bytes long.

When segmented headers are transmitted, each segment contains a 4-byte control section that contains the length of that segment, a segment number, and a flag indicating if this is the last segment. The following is an example of segmented header formats.

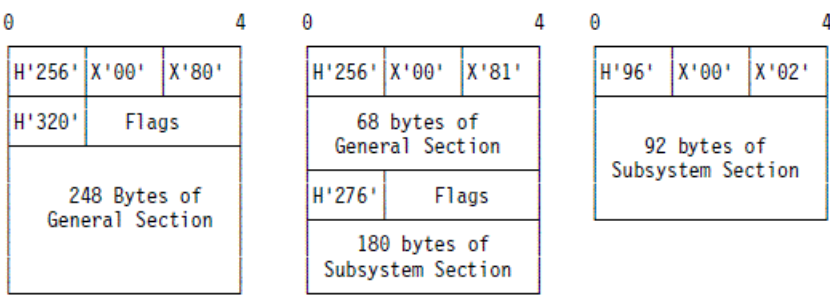


The remaining 252 bytes contain the actual header data.

For example, assume that you have a 600-byte header in the following format:



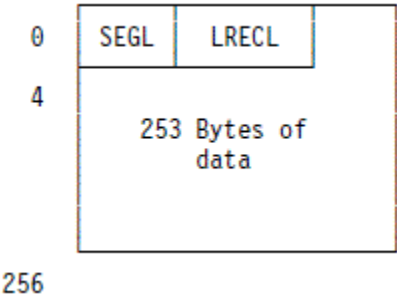
RSCS would send this header in the following form:



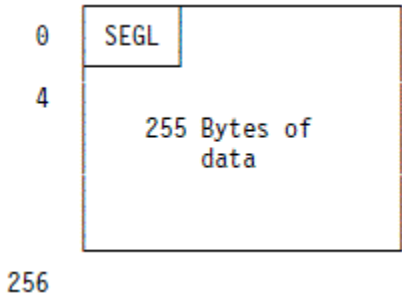
Spanned Record Format

Spanned record segments for NJE-type and SNANJE-type links have the following format:

First Segment



Remaining Segments



SEGL
Contains the length of segment sent (field is 1 byte long)

LRECL
Contains the logical record length of record after it is unspanned (field is 2 bytes long)

RSCS places the machine operation code or carriage control character for records that it is originating into the first byte of the data area in the first segment (as it does for unspanned records). The format of spanned records differs from that of unspanned records. Unspanned records contain a one-byte record length preceding the data or machine operation code.

TCPNJE Record Formats

The following sections describe the format of the TCPNJE control record and data block headers.

Control Record Format

Control records are exchanged between both sides of a TCPNJE-type link. After the TCP/IP connection is established, these records must be the first data exchanged by the TCPNJE links.

X'000'	Type	
X'008'	RHost	
X'010'	RIP	OHost ...
X'018' /	... OHost	OIP
X'020'	R	

TCPNJE Record Formats

F

Flags used to pass information about this record.

R

Reserved

LN

Length of data record, binary 16-bit value. The length does not include the length of the TTR header itself. If the length in a TTR is zero, this is the end-of-block marker.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

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*

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

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 document 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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

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

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information may 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.

Programming Interface Information

This book primarily documents information that is NOT intended to be used as Programming Interfaces of z/VM.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/VM. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

PI

<...Programming Interface information...>

PI end

Within such marked sections, information that is NOT intended to be used as Programming Interfaces of z/VM is identified by the following marking:

NOT Programming Interface Information

End of NOT Programming Interface Information
--

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on [IBM Copyright and trademark information](http://www.ibm.com/legal/copytrade) (<https://www.ibm.com/legal/copytrade>).

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

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

- The section entitled **IBM Websites** at [IBM Privacy Statement](https://www.ibm.com/privacy) (<https://www.ibm.com/privacy>)

- Cookies and Similar Technologies (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see [z/VM: General Information](#).

Where to Get z/VM Information

The current z/VM product documentation is available in [IBM Documentation - z/VM \(https://www.ibm.com/docs/en/zvm\)](https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- [z/VM: License Information](#), GI13-4377
- [z/VM: General Information](#), GC24-6286

Installation, Migration, and Service

- [z/VM: Installation Guide](#), GC24-6292
- [z/VM: Migration Guide](#), GC24-6294
- [z/VM: Service Guide](#), GC24-6325
- [z/VM: VMSES/E Introduction and Reference](#), GC24-6336

Planning and Administration

- [z/VM: CMS File Pool Planning, Administration, and Operation](#), SC24-6261
- [z/VM: CMS Planning and Administration](#), SC24-6264
- [z/VM: Connectivity](#), SC24-6267
- [z/VM: CP Planning and Administration](#), SC24-6271
- [z/VM: Getting Started with Linux on IBM Z](#), SC24-6287
- [z/VM: Group Control System](#), SC24-6289
- [z/VM: I/O Configuration](#), SC24-6291
- [z/VM: Running Guest Operating Systems](#), SC24-6321
- [z/VM: Saved Segments Planning and Administration](#), SC24-6322
- [z/VM: Secure Configuration Guide](#), SC24-6323

Customization and Tuning

- [z/VM: CP Exit Customization](#), SC24-6269
- [z/VM: Performance](#), SC24-6301

Operation and Use

- [z/VM: CMS Commands and Utilities Reference](#), SC24-6260
- [z/VM: CMS Primer](#), SC24-6265
- [z/VM: CMS User's Guide](#), SC24-6266
- [z/VM: CP Commands and Utilities Reference](#), SC24-6268

- [z/VM: System Operation](#), SC24-6326
- [z/VM: Virtual Machine Operation](#), SC24-6334
- [z/VM: XEDIT Commands and Macros Reference](#), SC24-6337
- [z/VM: XEDIT User's Guide](#), SC24-6338

Application Programming

- [z/VM: CMS Application Development Guide](#), SC24-6256
- [z/VM: CMS Application Development Guide for Assembler](#), SC24-6257
- [z/VM: CMS Application Multitasking](#), SC24-6258
- [z/VM: CMS Callable Services Reference](#), SC24-6259
- [z/VM: CMS Macros and Functions Reference](#), SC24-6262
- [z/VM: CMS Pipelines User's Guide and Reference](#), SC24-6252
- [z/VM: CP Programming Services](#), SC24-6272
- [z/VM: CPI Communications User's Guide](#), SC24-6273
- [z/VM: ESA/XC Principles of Operation](#), SC24-6285
- [z/VM: Language Environment User's Guide](#), SC24-6293
- [z/VM: OpenExtensions Advanced Application Programming Tools](#), SC24-6295
- [z/VM: OpenExtensions Callable Services Reference](#), SC24-6296
- [z/VM: OpenExtensions Commands Reference](#), SC24-6297
- [z/VM: OpenExtensions POSIX Conformance Document](#), GC24-6298
- [z/VM: OpenExtensions User's Guide](#), SC24-6299
- [z/VM: Program Management Binder for CMS](#), SC24-6304
- [z/VM: Reusable Server Kernel Programmer's Guide and Reference](#), SC24-6313
- [z/VM: REXX/VM Reference](#), SC24-6314
- [z/VM: REXX/VM User's Guide](#), SC24-6315
- [z/VM: Systems Management Application Programming](#), SC24-6327
- [z/VM: z/Architecture Extended Configuration \(z/XC\) Principles of Operation](#), SC27-4940

Diagnosis

- [z/VM: CMS and REXX/VM Messages and Codes](#), GC24-6255
- [z/VM: CP Messages and Codes](#), GC24-6270
- [z/VM: Diagnosis Guide](#), GC24-6280
- [z/VM: Dump Viewing Facility](#), GC24-6284
- [z/VM: Other Components Messages and Codes](#), GC24-6300
- [z/VM: VM Dump Tool](#), GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- [z/VM: DFSMS/VM Customization](#), SC24-6274
- [z/VM: DFSMS/VM Diagnosis Guide](#), GC24-6275
- [z/VM: DFSMS/VM Messages and Codes](#), GC24-6276
- [z/VM: DFSMS/VM Planning Guide](#), SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (<https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support>), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/iaa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See *z/VM Performance Data Pump*.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See *Data Pump Messages*.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- *z/VM: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Environmental Record Editing and Printing Program

- Environmental Record Editing and Printing Program (EREP): Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ifc2000_v2r5.pdf), GC35-0152
- Environmental Record Editing and Printing Program (EREP): User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ifc1000_v2r5.pdf), GC35-0151

Related Products

XL C++ for z/VM

- *XL C/C++ for z/VM: Runtime Library Reference*, SC09-7624
- *XL C/C++ for z/VM: User's Guide*, SC09-7625

z/OS

IBM Documentation - z/OS (<https://www.ibm.com/docs/en/zos>)

Additional Publications

- *IBM 7171 ASCII Device Attachment Control Unit Reference Manual and Programming Guide*, GA37-0021
- *IPDS Reference*, S544-3417
- *Systems Network Architecture: Formats*, GA27-3136
- *Systems Network Architecture: Sessions Between Logical Units*, GC20-1868
- *Systems Network Architecture: Technical Overview*, GC30-3073
- *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *VTAM: Programming*, SC31-6496
- *VTAM: Resource Definition Reference*, SC31-6498
- *z/OS: MVS JCL Reference*, SA22-7597

Index

Special Characters

*MSG system service [44](#)
&-symbols [101](#)

Numerics

3270P printer link driver
 command processing [78](#)
 description [77](#)
 initialization [77](#)
 processing [77](#)
 terminating [78](#)
4K optimization, storage [113](#)
7171 ASCII device attachment control unit [82](#)
9370 ASCII subsystem controller [82](#)

A

abend
 capturing [38](#)
 console messages [141](#)
 dump processing [142](#)
 ESTAE routines [38](#)
access method control block (ACB) [57](#)
accounting
 data structures [212](#)
 routines, networking links [68](#)
ACNTBUFF format [212](#)
AID byte [85](#)
ASCII printer and plotter link driver
 command processing [84](#)
 description [82](#)
 initialization [83](#)
 processing [83](#)
 terminating [85](#)
assembling spanned record segments [69](#)
attention identifier (AID) byte [85](#)
attention interrupt processing [85](#)
AUTHBLOK
 description [32](#)
 format [212](#)
authorization table [212](#)
auto-answer task
 exit points [63](#)
 initialization [62](#)
 invoking links [63](#)
 processing sign-on records [63](#)
auto-start task
 allocating ports [52](#)
 initialization [52](#)
 ITO processing [53](#)
 RETRY processing [53](#)
 termination [53](#)

B

BACKSPACE command
 command element format [237](#)
 processing routine [178](#)
bind image [55](#)
BSC
 networking links
 initialization [71](#)
 record compression [67](#)
 terminating [72](#)
 protocols [63](#)
buffer
 output, management [69](#)
 transmitting [69](#)
BUFFER macro [245](#)

C

capturing task abends [38](#)
CDEF macro [119](#)
CHANGE command
 command element format [234](#)
 processing [48](#)
changing link states [111](#)
channel table [28](#)
checks, program [142](#)
child group [18](#)
CID (communication identifier) [57](#)
CLOSE command
 command element format [232](#)
 processing [49](#)
CLOSEIN requests [50](#)
CLOSEOUT requests [50](#)
CMNDAREA
 basic format [231](#)
 element formats
 Type A0 [232](#)
 Type A1 [232](#), [233](#)
 Type A2 [234](#)
 Type C0 [235](#)
 Type C1 [235](#)
 Type C2 [235](#)
 Type E0 [236](#)
 Type L0 [236](#)
 Type L1 [237](#)
 Type L2 [238](#)
 Type L3 [238](#)
 Type V1 [239](#)
 Type V2 [239](#)
coded NOP records
 creating [68](#)
 format [262](#)
 NJE header reconstruction [68](#)
column masks [122](#)
command
 authorization table [32](#), [212](#)

- command (*continued*)
 - defining syntax [119](#)
 - finding definitions [123](#)
 - identifying syntax variations [120](#)
 - parsing [124](#)
 - processing
 - 3270P link driver [78](#)
 - ASCII printer and plotter link driver [84](#)
 - auto-start task [52](#)
 - communications task [45](#)
 - event scheduler task [54](#)
 - networking link driver tasks [68](#)
 - SNA3270P session driver [81](#)
 - SNARJE workstation task [97](#)
 - spool manager task [48](#)
 - TCPASCII printer and plotter link driver [87](#)
 - TN3270E link driver [79](#)
 - task, GCS [33](#)
- communication identifier (CID) [57](#)
- communications task
 - command processing [45](#)
 - configuration file processing [43](#)
 - creating system tasks [44](#)
 - exit 0 (initialization) [35](#)
 - exit 1 (termination) [46](#)
 - initialization [43](#)
 - termination [46](#)
- compilers, message [138](#)
- compressed load maps [157](#)
- compressing records [67](#)
- configuration file
 - processing [43](#)
- configuration file processing [43](#)
- console
 - abend messages [141](#)
 - spooling [141](#)
 - task, GCS [33](#)
- control record format, TCPNJE [264](#)
- control records, processing [70](#)
- control units, ASCII device [82](#)
- conversion repositories
 - structure [135](#)
- conversion repository, message [135](#)
- conversion routines, data and numeric [110](#)
- CRV table
 - description [17](#)
 - format [201](#)
- CTC networking links
 - initialization [71](#)
 - record compression [67](#)
 - sample trace data [143](#), [145](#)
 - terminating [72](#)
- CVT
 - description [16](#)
 - format [202](#)
 - subcommand, debugging [158](#)

D

- data areas
 - command and request elements [231](#)
 - control blocks [201](#)
 - locating in dumps [142](#)
 - networking [245](#)

- data areas (*continued*)
 - overview [15](#)
- data block header, TCPNJE [265](#)
- data block record header, TCPNJE [265](#)
- data conversion routines [110](#)
- data record format, trace [225](#)
- data set headers, NJE
 - building [65](#)
 - combining [67](#)
 - distribution list processing [100](#)
 - format [259](#)
 - receiving [66](#)
 - trace example [144](#)
- data tracing structures [224](#)
- DDEF macro [123](#)
- ddname
 - allocation [27](#), [110](#)
 - destination file [44](#)
- debugging considerations
 - abend processing [141](#)
 - compressed load map [157](#)
 - forcing a dump [157](#)
 - output example [171](#)
 - using subcommands [157](#), [169](#)
- decompressing records [68](#)
- defining command and statement syntax [119](#)
- definitions, finding [123](#)
- degraded mode [25](#)
- DEST
 - description [20](#)
 - format [204](#)
 - statement, processing [44](#)
- destination identifier file [44](#)
- destinations, message [131](#)
- Diagnose codes
 - testing if supported [38](#)
 - X'00' [43](#)
 - X'08' [46](#), [51](#)
 - X'14' [50](#)
 - X'B4' [51](#)
- disk file interface routine [109](#)
- distribution lists, processing [99](#)
- DRAIN command
 - command element format [236](#)
 - processing routine [178](#)
- dump formatting routines [199](#)
- Dump Viewing Facility
 - dump processing [142](#)
- DUMPSCAN subcommands [157](#), [169](#)
- DWA subcommand [158](#)
- dynamic port allocation [52](#)

E

- ECB
 - structure [40](#)
 - use with RPLs [57](#)
- EMSG setting [128](#)
- ENABLE command
 - command processing [62](#)
- ENABLE command processing [62](#)
- EQUATE
 - description [28](#)
 - finding entries [112](#)

- EQUATE (*continued*)
 - format [205](#)
- ESTAE exit routines [38](#)
- EVEBLOK
 - description [31](#)
 - format [225](#)
- event scheduler task
 - command queue [15](#)
 - initialization [53](#)
 - SCHEDULE command processing [54](#)
- EXEC processor task
 - command queue [15](#)
 - exec request format [236](#)
 - initialization [55](#)
- exit points cross-reference to calling modules [198](#)
- exit points, processing
 - auto-answer task [63](#)
 - command processing
 - exit 19 (command screening) [45](#)
 - exit 29 (unknown command) [45](#)
 - communications task
 - exit 0 (initialization) [44](#)
 - exit 1 (termination) [46](#)
 - exit 26 (link state change accounting) [111](#)
 - exit 35 (dump processing) [38](#)
 - message processing
 - exit 27 (message request screening) [131](#)
 - exit 28 (message language selection) [132](#)
 - NJE header processing
 - exit 11 (NJE job header creation) [65](#)
 - exit 12 (NJE data set header creation) [66](#)
 - exit 13 (NJE job trailer creation) [66](#)
 - exit 14 (NJE job header reception) [66](#)
 - exit 15 (NJE data set header reception) [66](#)
 - exit 16 (NJE job trailer reception) [67](#)
 - exit 41 (NJE job header post-processing) [66](#)
 - exit 42 (NJE data set header post-processing) [67](#)
 - exit 43 (NJE job trailer post-processing) [67](#)
 - NOTIFY link driver
 - exit 22 (NOTIFY driver note selection) [101](#)
 - exit 23 (NOTIFY driver note editing) [102](#)
 - exit 36 (NOTIFY driver purge) [102](#)
- EXITBLOK [29](#)
- external attribute buffer (XABHDR) [256](#)

F

- fanout processing [100](#)
- FCB table structure [32](#)
- file processing
 - data areas [213](#)
 - overview [21](#)
 - spool manager task [50](#)
- file request block [26](#)
- file work area [26](#)
- FILREQ
 - description [26](#)
 - use in disk file interface routine [109](#)
- filtering columnar messages [125](#)
- FILWORKA
 - description [26](#)
 - use in disk file interface routine [110](#)
- finding command and statement definitions [123](#)
- FLUSH command

- FLUSH command (*continued*)
 - command element format [238](#)
 - processing routine [45](#)
- FORCE command
 - command element format [235](#)
 - command processing [52](#)
 - processing routine [178](#)
- forcing a dump [157](#)
- FORM table
 - description [32](#)
 - format [213](#)
- format table entry, ITRACE [224](#)
- formatting
 - messages [132](#)
 - routines, dump [199](#)
- FREE command
 - command element format [236](#)
 - processing routine [178](#)
- FWDSpace command
 - command element format [237](#)
 - processing routine [178](#)

G

- GATEWAY link driver
 - gateway service macros [74](#)
 - initialization [73](#)
 - structure [73](#)
- GCS
 - command task [33](#)
 - console task [33](#)
 - dump processing
 - considerations [143](#)
 - finding active tasks [143](#)
 - tracing state blocks [143](#)
 - macros [34](#)
 - program management [33](#)
 - task management [33](#)
- generics, locating [109](#)
- GENIO requests, GCS [114](#)

H

- HASHBLOK
 - description [112](#)
 - format [227](#)
 - indexing routines [112](#)
- HDRTRL [245](#)
- header formats
 - data block [265](#)
 - data block record [265](#)
 - segmented [263](#)
- header record format, trace [224](#)
- header/trailer data area [245](#)
- HOLD command
 - command element format [236](#)
 - processing routine [178](#)
- honorary group, definition [18](#)
- host mode, MRJE link driver [95](#)

I

- I/O processing

- I/O processing (*continued*)
 - input spool routines [114](#)
 - interface routines [114](#)
 - output spool routines [115](#)
- indexing routines, HASHBLOK [112](#)
- indirection [135](#)
- initializing
 - 3270P link driver [77](#)
 - ASCII printer and plotter link driver [83](#)
 - auto-answer task [62](#)
 - auto-start task [52](#)
 - BSC and CTC link drivers [71](#)
 - communications task [43](#)
 - event scheduler task [53](#)
 - EXEC processor task [55](#)
 - GATEWAY link driver [73](#)
 - LISTPROC link driver [99](#)
 - LPD daemon links [89](#)
 - LPR links [90](#)
 - MRJE workstation link driver [95](#)
 - NOTIFY link driver [101](#)
 - port redirector task [60](#)
 - RJE link driver [93](#)
 - SCT task [57](#)
 - SNA3270P session driver [80](#)
 - SNANJE session driver [70](#)
 - SNARJE workstation session driver [96](#)
 - spool manager task [47](#)
 - TCPASCII printer and plotter link driver [86](#)
 - TCPNJE link driver [72](#)
 - TN3270E link driver [79](#)
 - UFT links [103](#)
 - UFTD daemon links [106](#)
- input spool routines, I/O [114](#)
- interrupt handling, I/O [114](#)
- IOTABLE
 - format [227](#)
 - subcommand, debugging [159](#)
 - use with I/O interface routines [114](#)
- issuing messages [131](#)
- ITO
 - command element format [235](#)
 - command processing [52](#)
 - enrollment process [53](#)
- ITRACE format table entry [224](#)
- ITRACE subcommand [161](#)
- ITRACFRM format [224](#)
- ITRACHDR format [224](#)
- ITRACREC format [225](#)
- IUCV
 - *MSG system service [44](#)
 - interrupt processing [44](#)
 - use with port redirector task [61](#)

J

- job headers, NJE
 - building [65](#)
 - format [257](#)
 - receiving [66](#)
 - trace example [144](#)
- job trailers, NJE
 - building [66](#)

- job trailers, NJE (*continued*)
 - format [259](#)
 - receiving [67](#)
 - trace example [144](#)

K

- keyword
 - command processing [124](#)
 - table, sample [116](#)

L

- LDEF macro [120](#)
- Line Printer Daemon (LPD) Link Driver
 - description [88](#)
 - initialization [89](#)
 - terminating [90](#)
- line trace formats [143](#)
- link driver tasks
 - equate entries [28](#)
 - LISTPROC [99](#)
 - networking [65](#)
 - NOTIFY [101](#)
 - summary table [5](#)
 - workstation [93](#)
- link state changes [111](#)
- LINKS subcommand [165](#)
- LINKTABL
 - description [17](#)
 - format [205](#)
- LISTEN request processing [61](#)
- LISTPROC link driver
 - initialization [99](#)
 - processing [99](#)
 - terminating [101](#)
- load map, compressed [157](#)
- loading exit routines [111](#)
- locating generics [109](#)
- log file, RSCS Interchange server [171](#)
- logical units [55](#)
- LOGON request [58](#)
- LOSTERM exit routine [57](#)
- LPR link driver
 - description [90](#)
 - initialization [90](#)
 - terminating [92](#)

M

- macros
 - gateway service [74](#)
 - GCS
 - IUCVCOM [44](#)
 - IUCVINI [44](#)
 - SCHEDX [33](#)
 - parsing and syntax
 - CDEF [119](#)
 - DDEF [123](#)
 - LDEF [120](#)
 - PDEF [120](#)
 - RSCSCMDS [119](#)
 - RSCSSTMT [119](#)

macros (*continued*)

summary table [34](#)

VTAM

RECEIVE [59](#)

SCIP [58](#)

TPEND [59](#)

management routines, storage [113](#)

MCOMP exec [138](#)

MCONV exec [138](#)

message

columnar

processing [130](#)

QUERY command responses [125](#)

repository entries [137](#)

compilers [138](#)

consoleabend [141](#)

conversion repository [135](#)

CRI support [128](#)

data structures [30](#)

exit points

exit 27 (message request screening) [131](#)

exit 28 (message language selection) [132](#)

formatting [132](#)

indirection [135](#)

issuing [131](#)

national language support [128](#)

preparing to issue [130](#)

processing [129](#)

request element [240](#)

routing codes [127](#)

structure [127](#)

subscriptions [30](#), [129](#)

substitutions, processing [133](#)

text

description [128](#)

structure [136](#)

translation repository [136](#)

message examples, notation used in [xxi](#)

midnight event [54](#)

mixed RCB feature [69](#)

module summary, RSCS [175](#)

modules that call exit points [198](#)

MONITENT

description [30](#)

format [228](#)

MRJE workstation link driver

description [94](#)

initialization

host mode [95](#)

remote mode [95](#)

processing [95](#)

terminating [96](#)

MSGBLOK

description [30](#)

format [240](#)

use in command processing [124](#)

use in disk file interface routine [109](#)

use with CRI prefix [128](#)

MSGLINE 31

MSGSKIP parameter [69](#)

MSGWA

acquiring [130](#)

description [31](#)

multiple data set headers [67](#)

N

national language support [128](#)

NCC area [246](#)

NDWA

subcommand, debugging [165](#)

use in networking links [65](#)

NETDATA

conversion routines [115](#)

records supported by RSCS [115](#)

use in NOTIFY link driver [102](#)

network connection control area [246](#)

Network Job Entry (NJE)

common data structures [65](#)

header and trailer formats [257](#)

networking equates (NJEEQU) [246](#)

networking link drivers

accounting routines [68](#)

BSC and CTC link driver [71](#)

command processing [68](#)

common data structures [65](#)

GATEWAY link driver [73](#)

LISTPROC link driver [99](#)

processing control records [70](#)

processing sign-on records [68](#)

receiving buffers [69](#)

record compression [67](#)

SNANJE session driver [70](#)

TCPNJE link driver [72](#)

transmitting buffers [69](#)

NIB (node initialization block) [57](#)

NJE headers

formats [257](#)

processing [65](#)

NJEEQU [246](#)

NMR

format [248](#)

processing [67](#)

node initialization block (NIB) [57](#)

NODE, description [20](#)

nonintelligent workstations [93](#)

NOP records [262](#)

notation used in message and response examples [xxi](#)

NOTIFY link driver

generating notes [101](#)

initializing [101](#)

purging files [102](#)

use of NETDATA format routines [102](#)

NSEXIT request [58](#)

numeric conversion routines [110](#)

O

obtaining record segments [70](#)

OPENIN requests [50](#)

OPENINTA request [51](#)

OPENOUT requests [50](#)

operating requirements [3](#)

ORDER command

command element format [232](#)

output spool routines, I/O [115](#)

overview

command processing [9](#)

data structures [15](#)

overview (*continued*)

- establishing links [8](#)
- establishing SNA sessions [8](#)
- file processing [7](#)
- message processing [10](#), [129](#)
- RSCS initialization [4](#)
- task communication [11](#)
- task descriptions [3](#)

P

- PAFBLOK [46](#), [121](#)
- parent group, definition [18](#)
- parsing
 - commands and statements [124](#)
 - general routines (DMTMPT) [116](#)
 - macros
 - CDEF [119](#)
 - DDEF [123](#)
 - LDEF [120](#)
 - PDEF [120](#)
 - syntax variations [120](#)
- pause characters [111](#)
- PDEF macro [120](#)
- phone numbers, validating [111](#)
- PORT
 - description [21](#)
 - format [210](#)
- port redirector task
 - initialization [60](#)
 - receiving LISTEN requests [61](#)
 - termination [62](#)
- PRDBLOK
 - format [218](#)
 - use with port redirector task [61](#)
- PRDTYPE field values [61](#)
- preface [xix](#)
- primary LU
 - SNA control task [55](#)
 - SNANJE session driver [71](#)
- printer link drivers
 - 3270P link driver [77](#)
 - ASCII printer and plotter link driver [82](#)
 - LPR [90](#)
 - LPR client [88](#)
 - SNA3270 session driver [80](#)
 - TCPASCII printer and plotter link driver [85](#)
 - TN3270E link driver [78](#)
 - UFT [103](#)
 - UFTD Daemon [105](#)
- private messages [127](#)
- private section, distribution record [100](#)
- processing message substitutions [133](#)
- processing sockets [88](#)
- program checks [142](#)
- programmable workstations [93](#)
- programs, filter [125](#)
- propagating QUERY commands [126](#)
- PURGE command
 - command element format [232](#)
 - processing [49](#)

Q

- QBLOCK [39](#)
- QSABLOK [113](#)
- QUERY command
 - columnar messages [125](#), [128](#)
 - PDEF options [122](#)
 - processing [125](#)
 - propagating [126](#)

R

- RCB
 - mixed, feature [69](#)
 - processing [70](#)
- RDEVBLOK
 - format [242](#)
 - use in file requests [50](#)
- reader interrupts, processing [47](#)
- READY command
 - command element format [236](#)
 - processing routine [178](#)
- real group, definition [18](#)
- RECEIVE request [59](#)
- reconstructing headers [66](#)
- record characteristics change section (RCCS) [65](#)
- record format
 - accounting [212](#)
 - coded NOP [262](#)
 - control record, TCPNJE [264](#)
 - data block header, TCPNJE [265](#)
 - data block record header, TCPNJE [265](#)
 - segmented header [263](#)
 - spanned [263](#)
- register save area [228](#)
- RELREQ request [58](#)
- remote mode, MRJE link driver [95](#)
- REORDER command
 - command element format [232](#)
 - processing [49](#)
- repositories, message
 - conversion [135](#)
 - syntax [119](#)
 - translation [136](#)
- request parameter list
 - starvation mode [60](#)
 - use in SNA control task [57](#)
 - use in SNANJE session driver [71](#)
- requirements, operational [3](#)
- RERNBLOK [20](#)
- REROUTE
 - description [20](#)
 - format [210](#)
- RESBLOK
 - description [25](#)
 - spool manager task processing [51](#)
- response examples, notation used in [xxi](#)
- RESQBLOK [25](#)
- RETRY
 - command element format [235](#)
 - command processing [52](#)
 - enrollment processing [53](#)
- RIB
 - format [249](#)

- RIB (*continued*)
 - subcommand, debugging [166](#)
 - use in networking links [65](#)
- RJE workstation link driver
 - description [93](#)
 - initialization [93](#)
 - processing [93](#)
 - terminating [94](#)
- root group, description [18](#)
- ROUTEGRP
 - description [18](#)
 - format [211](#)
- ROUTES subcommand [167](#)
- routines, dump formatting [199](#)
- routing codes, message [127](#)
- RSCS Data Interchange Manager
 - problem solving
 - using DEBUG [171](#)
 - using REXX traces [171](#)
 - using server log file [171](#)
- RSCSCMDS macro [119](#)
- RSCSDDEF macro [119](#), [123](#)
- RSCSSTMT macro [119](#)

S

- SAFTAG format [214](#)
- SAVEAREA format [228](#)
- SCB (string control bytes) [67](#), [68](#)
- SCHEDULE command
 - EVEBLOK format [225](#)
 - processing [54](#)
- SCIP request [58](#)
- searching control blocks [109](#)
- secondary LU
 - SNA control task [55](#)
 - SNANJE session driver [71](#)
- segmented header formats [263](#)
- SEPBLOK format [213](#)
- session control request [58](#)
- session driver tasks
 - SNA3270P printer session driver [80](#)
 - SNANJE session driver [70](#)
 - SNARJE workstation session driver [96](#)
 - summary table [6](#)
- SHOWMASK macro [122](#)
- sign-on records
 - auto-start task processing [63](#)
 - BTC and CTC link driver processing [71](#)
 - I records
 - description [68](#)
 - trace sample [145](#)
 - J records
 - description [68](#)
 - trace sample [145](#)
 - MRJE link driver [95](#)
 - processing, networking links [68](#)
- SIMLOGON macro [56](#)
- SMSG command [44](#)
- SNA
 - establishing sessions [55](#)
- SNA control task
 - attaching SNA3270P driver [80](#)
 - attaching SNARJE driver [96](#)

- SNA control task (*continued*)
 - establishing SNA sessions [55](#)
 - initializing [57](#)
 - maintaining RSCS/VTAM interface [59](#)
 - session cleanup [60](#)
 - VTAM exit routines [57](#)
- SNA3270P printer session driver
 - command processing [81](#)
 - description [80](#)
 - initializing [80](#)
 - processing [81](#)
 - RECEIVE ANY request [59](#)
 - terminating [82](#)
- SNANJE session driver
 - initialization [70](#)
 - sample trace trace [148](#)
 - termination [71](#)
- SNARJE workstation session driver
 - initialization [96](#)
 - processing [97](#)
 - RECEIVE ANY request [59](#)
 - terminating [98](#)
- SOCKBLOK format [218](#)
- SOCKCBLK format [219](#)
- SOCKET format [220](#)
- socket processing [88](#)
- spanned records
 - format [263](#)
 - segments, assembling [69](#)
- special messages, trapping [44](#)
- special purpose link drivers
 - LISTPROC link driver [99](#)
 - NOTIFY link driver [101](#)
- SPLINK, CP [114](#)
- spool interface routines [114](#)
- spool manager task
 - accepting files [47](#)
 - command processing [48](#)
 - exit points
 - exit 2 (spool file accept accounting) [48](#)
 - exit 21 (spool file accept/reject) [48](#)
 - exit 3 (spool file send accounting) [50](#)
 - exit 31 (sort priority change) [48](#)
 - exit 4 (spool file purge accounting) [49](#)
 - exit 5 (spool file receive accounting) [51](#)
 - exit 6 (TAG priority change) [48](#)
 - initialization [47](#)
 - managing unit record devices [51](#)
 - processing reader interrupts [47](#)
- spooling the console [141](#)
- START command
 - command element format [236](#), [239](#)
 - processing routine [178](#)
- starvation mode, RPL [60](#)
- state blocks
 - description [33](#)
 - tracing [143](#)
- statements, configuration file
 - finding definitions [123](#)
 - parsing [124](#)
 - syntax, defining [119](#)
- STOP command
 - command element format [239](#)
 - processing routine [179](#)

- storage
 - initialization, NJE link drivers [68](#)
 - management routines [113](#)
- store-and-forward
 - command processing [9](#)
 - file processing [7](#)
 - tag element [214](#)
- string control bytes [67](#), [68](#)
- subcommands
 - CVT [158](#)
 - DWA [158](#)
 - IOTABLE [159](#)
 - ITRACE [161](#)
 - LINKS [165](#)
 - NDWA [165](#)
 - RIB [166](#)
 - ROUTES [167](#)
 - TAGQUE [168](#)
 - TIB [169](#)
- substitutions, message [133](#)
- summary
 - dump format routines [199](#)
 - GCS macros [34](#)
 - link driver tasks [5](#)
 - NETDATA format records [115](#)
 - RCB processing [70](#)
 - routing codes, message [127](#)
 - RSCS modules [175](#)
 - session driver tasks [6](#)
 - system tasks [4](#)
- supporting national languages [128](#)
- synchronized tasks [13](#)
- syntax diagrams, how to read [xix](#)
- syntax, defining [119](#)
- SYSIDENT format [204](#)
- SYSIN job headers [65](#)
- SYSOUT
 - record tank [252](#)
 - spanned record [252](#)
- system events [54](#)
- system tasks
 - auto-start task [52](#)
 - communications task [43](#)
 - creation [44](#)
 - event manager task [53](#)
 - EXEC processor task [55](#)
 - port redirector task [60](#)
 - SNA control task [55](#)
 - spool manager task [46](#)
 - summary table [4](#)

T

- table display work area [130](#)
- TABWA area [130](#)
- TACTIVE subcommand [143](#)
- TAG element
 - description [23](#)
 - format [214](#)
- TAGAREA
 - description [21](#)
 - format [216](#)
- TAGQUE subcommand [168](#)
- TANBLOK [31](#), [53](#)

- TASHADOW element
 - description [23](#)
 - format [217](#)
- task abends, capturing [38](#)
- task communication
 - direct interface [13](#)
 - queued interface [14](#)
- task control block [33](#)
- task management
 - starting system tasks [35](#)
 - use of GCS facilities [33](#)
- task number block [31](#)
- task overview [3](#)
- task table service routines [117](#)
- TASKBLOK
 - creating [117](#)
 - description [27](#), [117](#)
 - format [211](#)
 - locating [118](#)
 - removing [117](#)
- TASTORAG element
 - description [22](#)
 - format [217](#)
- TCP/IP
 - command processing routine [179](#)
 - connections
 - LPD link driver [88](#)
 - LPR link driver [90](#)
 - TCPASCII link driver [85](#)
 - TCPNJE link driver [72](#)
 - UFD link driver [105](#)
 - UFT link driver [103](#)
 - port redirector task [60](#)
 - related control blocks [218](#)
- TCPASCII link driver
 - command processing [87](#)
 - description [85](#)
 - initialization [86](#)
 - processing [86](#)
 - socket processing [88](#)
 - terminating [88](#)
- TCPNJE link driver
 - initialization [72](#)
 - record formats [264](#)
 - sample trace data [149](#)
 - termination [73](#)
- telecommunications buffer [245](#)
- TEMPLATE file [101](#)
- terminating
 - 3270P link driver [78](#)
 - ASCII printer and plotter link driver [85](#)
 - auto-start task [53](#)
 - BSC and CTC networking links [72](#)
 - GATEWAY-type links [75](#)
 - LISTPROC link driver [101](#)
 - LPD link driver [90](#)
 - LPR link driver [92](#)
 - MRJE workstation link [96](#)
 - port redirector task [62](#)
 - RJE link driver [94](#)
 - RSCS [46](#)
 - SNA control task [60](#)
 - SNA3270P session driver [82](#)
 - SNANJE session driver [71](#)

- terminating (*continued*)
 - SNARJE workstation session driver [98](#)
 - TCPASCII link driver [88](#)
 - TCPNJE link driver [73](#)
 - TN3270E link driver [80](#)
 - UFT link driver [105](#)
 - UFTD link driver [107](#)

TIB

- format [252](#)
- subcommand, debugging [169](#)
- use in networking links [65](#)

- time-of-day clock conversion routines [110](#)

- TN3270E Printer Link Driver

- command processing [79](#)
- description [78](#)
- initialization [79](#)
- processing [79](#)
- terminating [80](#)

TOD

- clock conversion routines [110](#)

- TOD clock conversion routines [110](#)

- top-level group, definition [18](#)

- TPEND request [59](#)

- TRACE command

- command element format [236](#)
- processing routine [179](#)

- trace data format [143](#)

- trace data record format [225](#)

- trace header record format [224](#)

- trace structures [224](#)

- tracing RSCS Interchange problems [171](#)

- tracing state blocks [143](#)

- TRANSFER command

- command element format [233](#)
- processing [49](#)
- syntax structure, example [120](#)

- translation repositories

- structure [136](#)

- translation repository [136](#)

- transmission algorithms [48](#), [51](#)

- trapping special messages [44](#)

- TTB, data block header [265](#)

- TTR, data block record header [265](#)

U

- unique connections, identifying [72](#)

- unit record devices

- input spool routines [114](#)

- output spool routines [115](#)

- Unsolicited File Transfer (UFT) Driver

- description [103](#)

- initialization [103](#)

- terminating [105](#)

- Unsolicited File Transfer Daemon (UFTD) Driver

- description [105](#)

- initialization [106](#)

- terminating [107](#)

- user events [54](#)

V

- validating phone numbers [111](#)

- variations, command syntax [120](#)

- VM Batch [96](#)

- VMDUMP command [157](#)

- VTAM

- exit routines [57](#)

- RSCS/VTAM interface [59](#)

W

- workstation link drivers

- MRJE link driver [94](#)

- RJE link driver [93](#)

- SNARJE session driver [96](#)

X

- XABHDR (external attribute buffer) [256](#)



Product Number: 5741-A09

Printed in USA

GC24-6316-73

