

z/VM  
7.3

*RSCS Networking  
Exit Customization*



**Note:**

Before you use this information and the product it supports, read the information in [“Notices” on page 371.](#)

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

<b>Figures.....</b>	<b>xi</b>
<b>Tables.....</b>	<b>xiii</b>
<b>About This Document.....</b>	<b>xv</b>
Intended Audience.....	xv
Syntax, Message, and Response Conventions.....	xv
Where to Find More Information.....	xviii
Links to Other Documents and Websites.....	xviii
<b>How to provide feedback to IBM.....</b>	<b>xix</b>
<b>Summary of Changes for z/VM: RSCS Networking Exit Customization.....</b>	<b>xxi</b>
SC24-6317-73, z/VM 7.3 (December 2023).....	xxi
SC24-6317-73, z/VM 7.3 (September 2022).....	xxi
SC24-6317-01, z/VM: 7.2 (September 2020).....	xxi
SC24-6317-00, z/VM: 7.1 (September 2018).....	xxi
<b>Chapter 1. Introduction.....</b>	<b>1</b>
Types of RSCS Exits.....	1
RSCS Exit Facility.....	1
Transmission Algorithms.....	4
ASCII Printer and Plotter Exits.....	4
Gateway Programming Interface.....	5
LPR Exits.....	5
LPD Exits.....	6
UFT Exits.....	7
UFTD Exits.....	7
Loadable Link Drivers.....	8
Sample Exit Routines.....	8
<b>Chapter 2. Customizing RSCS.....</b>	<b>9</b>
Selecting Which Exits to Use.....	9
RSCS Exit Facility.....	9
Transmission Algorithms.....	11
ASCII Printer and Plotter Exits.....	11
Gateway Programming Interface.....	12
LPD Exits.....	12
LPR Exits.....	12
UFT Exits.....	12
UFTD Exits.....	12
Loadable Link Drivers.....	12
Exit Routine Considerations.....	13
Language Requirements.....	13
Code Attributes.....	13
31-Bit Enablement.....	13
Return Codes.....	14
Writing Exit Routines.....	14
Calling Conventions.....	14

Linkage Conventions.....	16
Restoring Registers.....	16
Using RSCS Facilities.....	17
Issuing Messages.....	17
Calling Exit Routines.....	18
Example 1: Defining Printing Shifts.....	18
Example 2: Using Two Exit Routines.....	19
Example 3: Creating a New Command.....	22
Storage Considerations.....	25
Using GCS Macros.....	25
Quick Storage Allocation Routines.....	25
Example 4: Mapping a Work Area.....	26
Packaging Considerations.....	27
Identifying Entry Points.....	28
Sharing Information.....	30
User Fields.....	31
Link-Editing Considerations.....	33
Common Problems and Solutions.....	33
Distribution Considerations.....	34
Specifying the Order of the Exit Routines.....	35
Tracing Exit Routines.....	35
Using Sample Exit Packages.....	36
Enabling Sample Exit Routines.....	36
Summary of Sample Packages.....	36
<b>Chapter 3. IBM-Defined Exit Points.....</b>	<b>39</b>
Usage Conventions.....	39
Standard Entry Conditions.....	39
Standard Exit Conditions.....	39
Standard Return Codes.....	39
Data Areas.....	40
Accounting Records.....	40
Exit 0 – Initialization.....	41
Exit 1 – Termination.....	43
Exit 2 – Spool File Accept Accounting.....	45
Exit 3 – Spool File Send Accounting.....	47
Exit 4 – Spool File Purge Accounting.....	49
Exit 5 – Spool File Receive Accounting.....	50
Exit 6 – TAG Priority Change.....	52
Exit 7 – Auto-Answer Sign-On Time Out.....	54
Exit 8 – Auto-Answer Unrecognizable Data.....	55
Exit 9 – Auto-Answer Sign-On Validation.....	57
Exit 10 – Auto-Answer Sign-On Reject.....	59
Exit 11 – NJE Job Header Creation.....	61
Exit 12 – NJE Data Set Header Creation.....	64
Exit 13 – NJE Job Trailer Creation.....	67
Exit 14 – NJE Job Header Reception.....	69
Exit 15 – NJE Data Set Header Reception.....	72
Exit 16 – NJE Job Trailer Reception.....	74
Exit 17 – Separator Page Selection.....	76
Exit 18 – Separator Page Generation.....	78
Exit 19 – Command Screening.....	81
Exit 21 – Spool File Accept/Reject.....	83
Exit 22 – NOTIFY Driver Note Selection.....	85
Exit 23 – NOTIFY Driver Note Editing.....	86
Exit 24 – Spooling CP Command Screening.....	88
Exit 25 – Post-CP Command Screening.....	91

Exit 26 – Link State Change Accounting.....	94
Exit 27 – Message Request Screening.....	96
Exit 28 – Message Language Selection.....	98
Exit 29 – Unknown Command.....	99
Exit 30 – Reroute Interception.....	101
Exit 31 – Sort Priority Change.....	103
Exit 32 – NMR Reception.....	105
Exit 33 – User Parm Processing.....	107
Exit 34 – Spool Manager Command.....	109
Exit 35 – Dump Processing.....	110
Exit 36 – NOTIFY Driver Purge.....	112
Exit 37 – NJE Job Header Transmission.....	113
Exit 38 – NJE Data Set Header Transmission.....	115
Exit 39 – NJE Job Trailer Transmission.....	117
Exit 40 – NJE Record Reception.....	119
Exit 41 – NJE Job Header Post-Processing.....	121
Exit 42 – NJE Data Set Header Post-Processing.....	123
Exit 43 – NJE Job Trailer Post-Processing.....	125
Exit 44 – Link Termination.....	127
Exit 45 – Output Page Accounting.....	128
Exit 46 – Verification of Page Accounting.....	130
Exit 47 – Driver Initialization.....	132
Exit 48 – Verification of Output Page Error.....	134
<b>Chapter 4. Transmission Algorithm Processing.....</b>	<b>137</b>
Specifying a Transmission Algorithm.....	137
Transmission Algorithm Programming Considerations.....	137
External Transmission Algorithms.....	138
Open Request Processing.....	138
Accept Request Processing.....	139
Select Request Processing.....	141
Internal Transmission Algorithms.....	142
Programming Considerations.....	142
Transmission Algorithm 0.....	143
Transmission Algorithm 1.....	143
Transmission Algorithms 2 - F.....	144
Packaging Transmission Algorithms.....	144
Installing External Transmission Algorithms.....	145
Installing Internal Transmission Algorithms.....	145
<b>Chapter 5. ASCII Printer and Plotter Exit Processing.....</b>	<b>147</b>
ASCII Exit Programming Considerations.....	147
Required Values.....	147
Entry Conditions.....	148
Exit Conditions.....	148
ASCII Exit Routines.....	148
Initialization Routine.....	148
TAG Processing Routine.....	150
Record Processing Routine.....	151
Device Reset Routine.....	152
Message Processing Routine.....	153
Attention Interrupt Processing Routine.....	154
Termination Routine.....	155
Sample ASCII Printer and Plotter Exit Modules.....	156
Printer Exit Modules.....	157
ASCPSE Routine.....	158
ASCPSE Configuration File.....	160

ASCXONE Routine.....	163
ASCXONE Configuration file.....	165
IBM XY/749 Plotter Exit Module.....	165
Nicolet Zeta 8 Plotter Exit Module.....	166
Sending Files with Sample Exit Routines.....	166
<b>Chapter 6. Gateway Programming Interface.....</b>	<b>167</b>
Gateway Program.....	167
Entry Conditions.....	167
Exit Conditions.....	168
Return Codes.....	169
Programming Considerations.....	169
Work Area Considerations.....	169
Link-Editing Considerations.....	169
Program Structure.....	170
Types of Work.....	170
Scheduling Work.....	171
Supported NJE Sub Record Control Byte Values.....	172
Reason Code Responses.....	173
Gateway Service Macros.....	174
NJEABORT.....	175
NJECLOSE.....	176
NJECONCT.....	177
NJEDSCON.....	178
NJEGET.....	179
NJEOPEN.....	180
NJEPUT.....	182
NJERJECT.....	183
NJE File Control Block Fields.....	184
NJEFILE.....	185
NJEFILED.....	186
<b>Chapter 7. TCP/IP LPR Exit Points.....</b>	<b>187</b>
LPR Programming Considerations.....	187
Required Values.....	187
Entry Conditions.....	188
Printer Flag Fields.....	188
Print Record Vector.....	189
Exit Conditions.....	189
LPR Exit Routines.....	189
LPR Initialization Routine.....	189
LPR TAG Processing Routine.....	191
LPR Record Processing Routine.....	193
LPR End of File Routine.....	195
LPR Control File Routine.....	197
LPR Termination Routine.....	199
Sample LPR Exit Routines.....	200
LPRXONE Routine.....	201
LPRXONE Configuration file.....	203
LPRXPSE Routine.....	204
LPRXPSE Configuration file.....	206
<b>Chapter 8. TCP/IP LPD Exit Points.....</b>	<b>211</b>
LPD Programming Considerations.....	211
Required Values.....	211
Entry Conditions.....	212
Order of the Control File and Data File.....	212

Response Messages.....	212
Print Record Vector.....	212
Exit Conditions.....	213
LPD Exit Routines.....	213
LPD Initialization Routine.....	213
LPD Print Command Processing Routine.....	214
LPD Print Job Command Processing Routine.....	216
LPD Data Processing Routine.....	218
LPD End of File Routine.....	220
LPD Control File Routine.....	222
LPD Termination Routine.....	223
Sample LPD Exit Routine.....	224
LPDXMANY Routine.....	224
LPDXMANY Configuration file.....	226
Using an LPD-Type Link as a Print Server.....	230
<b>Chapter 9. TCP/IP UFT Exit Points.....</b>	<b>231</b>
UFT Programming Considerations.....	231
Required Values.....	231
Entry Conditions.....	232
UFT Commands.....	232
Data Record Vector.....	232
Exit Conditions.....	233
UFT Exit Routines.....	233
UFT Initialization Routine.....	233
UFT TAG Processing Routine.....	234
UFT Record Processing Routine.....	235
UFT End of File Routine.....	237
UFT Command Routine.....	238
UFT Termination Routine.....	239
Sample UFT Exit Routine.....	240
UFTXOUT Routine.....	241
UFTXOUT Configuration File.....	242
<b>Chapter 10. TCP/IP UFTD Exit Points.....</b>	<b>245</b>
UFTD Programming Considerations.....	245
Required Values.....	245
Entry Conditions.....	246
Order of the UFT Commands and Data.....	246
Response Messages.....	246
Data Record Vector.....	246
Exit Conditions.....	247
UFTD Exit Routines.....	247
UFTD Initialization Routine.....	247
UFTD Connect Processing Routine.....	248
UFTD Command Processing Routine.....	250
UFTD Data Processing Routine.....	252
UFTD End of File Routine.....	254
UFTD Termination Routine.....	256
Sample UFTD Exit Routine.....	257
UFTXIN Routine.....	257
UFTXIN Configuration File.....	259
<b>Chapter 11. RSCS Macros.....</b>	<b>265</b>
Specifying Parameters.....	265
Program Structure Macros.....	265
BRC – Branch on Return Code.....	266

EXITCALL – Providing an Exit Point.....	267
HASHBLOK – Defining a Hash Table.....	270
INSTALIT – Adding a Record Format Table.....	272
ITFORMAT – Building a Format Table.....	273
ITRACE – Tracing an Event.....	274
PARDSECT – Defining a Keyword Table.....	279
PAREND – Defining the End of a Keyword Table.....	281
PARKEY – Defining a Keyword.....	282
PAROPT – Defining Options for a Keyword.....	284
QSABLOK – Defining a Storage Request.....	286
RCALL – Passing Control to a Routine.....	288
RENTY – Defining a Module Entry Point.....	290
REXIT – Defining a Module Return Point.....	295
RMOD – Defining a Module.....	297
RMSG – Issuing a Message.....	299
RWORK – Defining the Start of a Module Work Area.....	302
RWORKEND – Defining the End of a Module Work Area.....	303
SOCKET – Using the TCP/IP Socket Interface.....	304
SOCKET Function Descriptions.....	307
Invoking the SOCKET Macro.....	307
ACCEPT.....	307
BIND.....	307
CANCEL.....	308
CLOSE.....	308
CONNECT.....	308
DSECT.....	308
FCNTL.....	308
GETCLIENTID.....	309
GETHOSTBYNAME.....	309
GETHOSTID.....	309
GETHOSTNAME.....	309
GETPEERNAME.....	310
GETSOCKNAME.....	310
GETSOCKOPT.....	310
GIVESOCKET.....	311
INITIALIZE.....	311
IOCTL.....	311
LISTEN.....	312
READ.....	312
RECV.....	312
RECVFROM.....	313
SELECT.....	313
SEND.....	314
SENDTO.....	314
SETSOCKOPT.....	315
SHUTDOWN.....	315
SOCKET.....	316
TAKESOCKET.....	316
TERMINATE.....	316
WRITE.....	316
Control Block Macros.....	318
<b>Chapter 12. Supported Routines in the CRV.....</b>	<b>321</b>
Executable Entry Points.....	321
DMTAXMRQ.....	321
DMTBPLLX.....	321
DMTCOMDG.....	322



DMTCOMDQ.....	322
DMTCOMFI.....	322
DMTCOMGG.....	323
DMTCOMGN.....	323
DMTCOMHG.....	324
DMTCOMLK.....	324
DMTCOMNQ.....	324
DMTCOMSM.....	325
DMTCOMTE.....	325
DMTCOMTS.....	326
DMTDDLEP.....	326
DMTHASHA.....	327
DMTHASHB.....	327
DMTHASHC.....	327
DMTHASHD.....	328
DMTHASHF.....	328
DMTHASHG.....	328
DMTHASHS.....	329
DMTIOTHD.....	329
DMTIOTST.....	329
DMTLOGCL.....	329
DMTLOGEP.....	330
DMTMANDE.....	330
DMTMGFFM.....	330
DMTMGXEP.....	331
DMTMPTBP.....	331
DMTMPTCK.....	332
DMTMPTGD.....	332
DMTMPTGP.....	332
DMTPAREP.....	333
DMTPRDDQ.....	334
DMTPRDNQ.....	334
DMTQSAAB.....	334
DMTQSAFA.....	335
DMTQSAUB.....	335
DMTRDREP.....	335
DMTRDROP.....	336
DMTRERSC.....	337
DMTRESLO.....	337
DMTRESUN.....	337
DMTSEPBL.....	338
DMTSOKET.....	338
DMTTASKA.....	339
DMTTASKD.....	340
DMTTASKF.....	340
DMTTASKG.....	340
DMTUROEP.....	341
DMTUROFL.....	341
Nonexecutable Entry Points.....	341

## **Chapter 13. Message Repositories.....345**

Conversion Repository.....	345
Naming Convention.....	345
Repository Structure.....	345
Control Statements.....	345
Message Definition Statement.....	347
Message Fields.....	348

Translation Repository.....	356
Naming Convention.....	356
Repository Structure.....	356
MCOMP and MCONV – Compiling Message Repositories.....	360
<b>Chapter 14. Customizing the RSCS Data Interchange Manager.....</b>	<b>363</b>
Creating Exit Routines.....	363
Using Accounting Exits.....	363
Using Command Exits.....	364
Using Format Recognition Exits.....	364
Using Security Exits.....	365
<b>Appendix A. DSECTs Generated by Mapping Macros.....</b>	<b>367</b>
<b>Notices.....</b>	<b>371</b>
Programming Interface Information.....	372
Trademarks.....	372
Terms and Conditions for Product Documentation.....	372
IBM Online Privacy Statement.....	373
<b>Bibliography.....</b>	<b>375</b>
Where to Get z/VM Information.....	375
z/VM Base Library.....	375
z/VM Facilities and Features.....	376
Prerequisite Products.....	378
Related Products.....	378
Additional Publications.....	378
<b>Index.....</b>	<b>379</b>

---

# Figures

1. Invoking an Exit Point.....	3
2. Defining Shifts with an EVENTS CONFIG File.....	18
3. Exit 31 Routine: Stopping Large Files from Printing.....	18
4. PLIMIT LKEDCTRL File.....	19
5. Exit 0 Routine: Reading the JOBNAME CONFIG File.....	20
6. Exit 11 Routine: Placing the Job Name in Job Header.....	21
7. JOBNAME LKEDCTRL File.....	21
8. Exit 29 Routine: Implementing the TYPE Command (Part 1 of 3).....	22
9. Exit 29 Routine: Implementing the TYPE Command (Part 2 of 3).....	23
10. Exit 29 Routine: Implementing the TYPE Command (Part 3 of 3).....	24
11. TYPE LKEDCTRL File.....	24
12. Exit 14 Routine: Allocating and Deallocating a Work Area.....	26
13. Exit 1 Routine: Freeing Storage Acquired by the Exit 14 Routine.....	27
14. Packaging One Source Module (SECURE ASSEMBLE).....	28
15. SECURE LKEDCTRL File.....	28
16. Specifying Aliases in the Source Module (SECURE ASSEMBLE).....	28
17. Sample Link-Edit Control File: Creating Two Load Modules.....	29
18. Link-edit Control File: Creating One Load Module.....	29
19. Identifying Entry Points (ACCNTI ASSEMBLE).....	30
20. Identifying Entry Points in the PROFILE GCS.....	30
21. Example Macro to Map Exit Utility Routine Addresses.....	32
22. Exit 0 Routine: Installing a Utility Routine Package.....	32
23. Finding and Calling a Utility Routine.....	32

24. Sample RSCS CONFIG File.....	34
25. SECURE CONFIG File for Security Exit Package.....	35
26. Sample Configuration File Statements.....	35
27. Sample External Transmission Algorithm.....	144
28. Sample LKEDCTRL File: External Transmission Algorithm.....	145
29. Specifying the WORK Parameter.....	169
30. Basic Structure of Gateway Program.....	170
31. Defining Keywords.....	283
32. Sample of a SOCKET Macro Invocation.....	307

---

# Tables

1. Examples of Syntax Diagram Conventions..... xvi

2. Required RENT and SAVAREA Settings..... 14

3. Sample Exit Routine Packages.....37

4. Printer Flag Fields and Values..... 188

5. Macros That Map RSCS Data Areas..... 367



# About This Document

---

This document describes the exit facilities of IBM® Remote Spooling Communications Subsystem (RSCS) Networking for z/VM. This information can help you:

- Plan for customizing RSCS
- Implement and test your exit routines
- Use RSCS macros in your exit routines

## Intended Audience

---

This information is intended for programmers who are responsible for initializing and modifying RSCS. You should be familiar with the concepts, terminology, and use of z/VM®, RSCS, and the Group Control System (GCS). An understanding of networking protocols, Virtual Telecommunications Access Method (VTAM®), and TCP/IP is also helpful.

Attention
Only experienced programmers should attempt to use the programming interfaces described in this document. Writing an exit routine requires thorough knowledge of telecommunication protocols, systems programming, and RSCS programming techniques.
If you attempt to customize RSCS by writing exit routines, transmission algorithms, or gateway programs without having this knowledge, you run the risk of seriously degrading your system's performance and causing system failure.
When you use the RENTRY macro for exit routines, you must use specific options. See “Calling Conventions” on page 14 for a list of the required options and see “Example 4: Mapping a Work Area” on page 26 for an explanation of work area usage. Failure to follow these guidelines may result in abend conditions.

## 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 xvi](#).

Table 1. Examples of Syntax Diagram Conventions

Syntax Diagram Convention	Example
<b>Keywords and Constants</b> <p>A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown.</p> <p>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.</p>	<p>» KEYWORD «</p>
<b>Abbreviations</b> <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>
<b>Symbols</b> <p>You must specify these symbols exactly as they appear in the syntax diagram.</p>	<p>* Asterisk</p> <p>:</p> <p>Colon</p> <p>,</p> <p>Comma</p> <p>=</p> <p>Equal Sign</p> <p>-</p> <p>Hyphen</p> <p>()</p> <p>Parentheses</p> <p>.</p> <p>Period</p>
<b>Variables</b> <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>



Table 1. Examples of Syntax Diagram Conventions (continued)

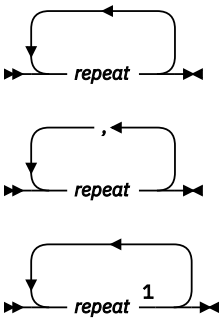
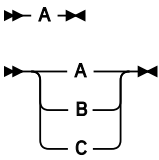
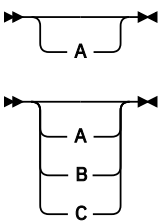
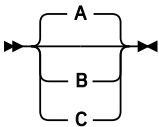
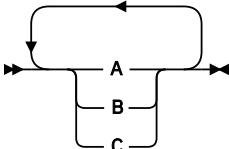
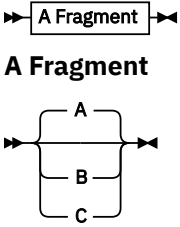
Syntax Diagram Convention	Example
<p><b>Repetitions</b></p> <p>An arrow returning to the left means that the item can be repeated.</p> <p>A character within the arrow means that you must separate each repetition of the item with that character.</p> <p>A number (1) by the arrow references a syntax note 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>Notes:</p> <p><sup>1</sup> Specify <i>repeat</i> up to 5 times.</p>
<p><b>Required Item or Choice</b></p> <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><b>Optional Item or Choice</b></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><b>Defaults</b></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><b>Repeatable Choice</b></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>	

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	Example
<b>Syntax Fragment</b> 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. In this example, the fragment is named "A Fragment."	 <p>The example shows two diagrams. The first is a box labeled 'A Fragment' with arrows at both ends. The second is a diagram labeled 'A Fragment' with a large bracket on the left and three sub-elements labeled A, B, and C stacked vertically, each with its own bracket on the left and a small arrow on the right.</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 more information about RSCS and other z/VM topics, see [“Bibliography” on page 375.](#)

## 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 Exit Customization

---

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.

## SC24-6317-73, z/VM 7.3 (December 2023)

---

This edition includes terminology, maintenance, and editorial changes.

## SC24-6317-73, 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.

## SC24-6317-01, z/VM: 7.2 (September 2020)

---

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

## SC24-6317-00, z/VM: 7.1 (September 2018)

---

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



# Chapter 1. Introduction

RSCS Networking for z/VM is a general purpose spooling subsystem. RSCS is designed to be flexible so that you can customize it to meet the changing needs of your installation and network.

Using exit facilities and control files, such as the configuration file and EVENTS file, you can set up and tailor the network where RSCS functions. For information about these files, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

You can use the exit facility to modify RSCS processing to meet any special functional requirements for your installation. This document describes how you can use the exit facility to customize RSCS.

## Attention

The writing of exit routines and the installation of a new exit point require a thorough knowledge of systems programming and RSCS programming conventions. If, without having this knowledge, you attempt to write exit routines or install new exit points, you risk seriously degrading the performance of your system or causing system problems. Therefore, only an experienced programmer should attempt to use RSCS exits to customize or modify RSCS processing.

## Types of RSCS Exits

An *exit* is a procedure outside the main RSCS program that, under certain conditions, receives control from RSCS. An exit consists of an exit point and an exit routine.

An *exit point* is a location in RSCS source code from which processing passes to an exit routine. An *exit routine* is the code that you supply to customize standard RSCS processing.

RSCS provides the following categories of exits, which you can use to customize specific processing areas:

- RSCS exit facility
- Transmission algorithms
- ASCII printer and plotter exits
- Gateway programming interface
- TCP/IP line printer remote (LPR) exits
- TCP/IP line printer daemon (LPD) exits
- TCP/IP unsolicited file transfer (UFT) exits
- TCP/IP unsolicited file transfer daemon (UFTD) exits

RSCS also provides facilities for creating additional types of link drivers and customizing RSCS messages.

Chapter 2, “Customizing RSCS,” on page 9 contains information about writing exit routines.

## RSCS Exit Facility

The RSCS exit facility is an interface between RSCS and your routines. When you use this exit facility, you can modify certain RSCS processes without directly altering RSCS source code. Your exit routines remain independent of RSCS code. This can simplify maintenance and increase the portability of your exit routines.

Using the RSCS exit facility, you can customize different areas of RSCS processing, including:

- Abnormal end (abend)
- Auto-answer sign-on
- Commands
- Messages

- Network job entry (NJE) header
- Separator page output
- Spool file
- Start and end

**You do not need to supply exit routines as part of standard RSCS processing.** Use of the RSCS exit facility is optional. If you do not implement an exit point, it is transparent during standard RSCS processing.

### How the RSCS Exit Facility Works

The RSCS exit facility can have 256 exit points, each identified by a number from 0 to 255. The *nnn* value specified on the EXITCALL macro corresponds to the exit point number. EXITCALL macros define and identify the numbered exit points in the RSCS source code. See [“EXITCALL – Providing an Exit Point” on page 267](#).

To implement an exit point, you must define it with an EXIT statement in the RSCS configuration file and supply an exit routine. You can enable or disable the exit point by specifying the ON or OFF operands, respectively, on the EXIT statement. You can also dynamically change the exit point's status with the EXIT command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

When you enable an exit point, RSCS checks for associated exit routines. You can associate several exit routines with one exit point by specifying each exit routine on the EXIT statement. RSCS passes control to the first exit routine specified. The remaining exit routines receive control in the order they appear on the EXIT statement. For example in the following EXIT statement, exit routine EXITSTE1 receives control before exit routine EXITSTE2 and so on.

```
EXIT 102 ON EXITSTE1 EXITSTE2 EXITSTE3 ...
```

This process continues until each specified exit routine is called. However, if an exit routine issues a nonzero return code, RSCS will not call the subsequent exit routines in the list.

[Figure 1 on page 3](#) shows the relationship between EXITCALL macros, EXIT statements, and an exit routine. The EXIT statement defines the exit number *nnn* at label XYZ for the EXITCALL macro in RSCS code. The statement also identifies EXITSTE1 as the entry point of the exit routine for exit point *nnn*.



**Attention:** When writing exit routines, you must ensure that data areas shared by multiple exits are handled as described in [“Writing Exit Routines” on page 14](#) and [“Sharing Information” on page 30](#). If the guidelines are not followed, data will not be shared properly, possibly leading to an RSCS outage.



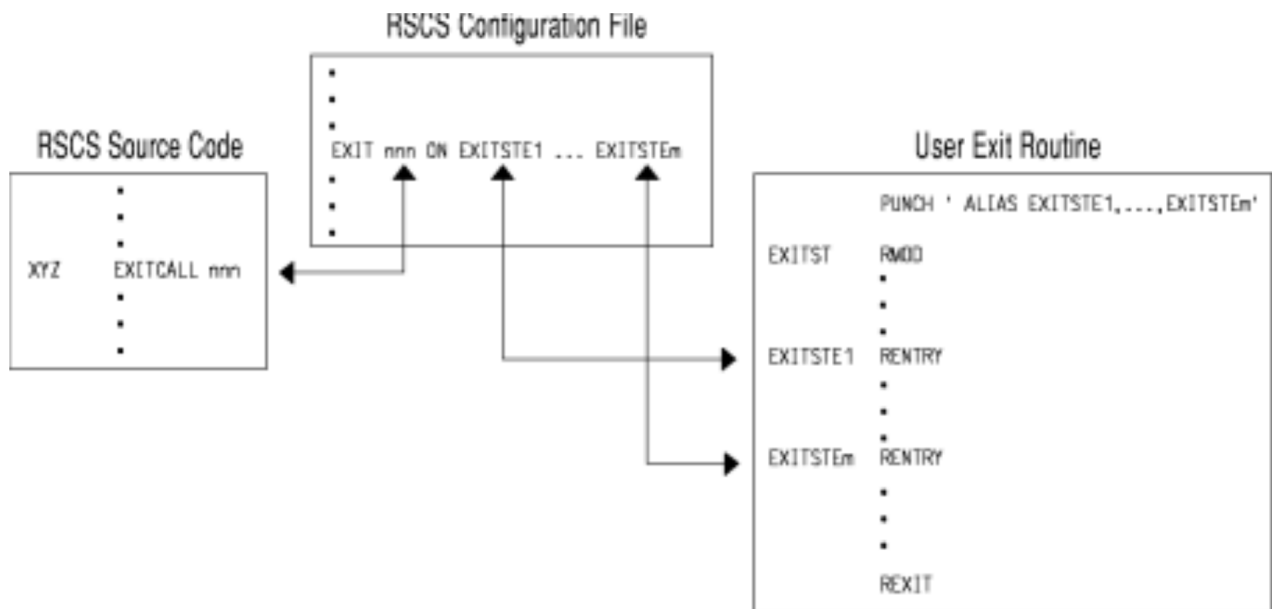


Figure 1. Invoking an Exit Point

When an exit point is disabled or not defined by an `EXIT` statement, RSCS bypasses the exit point. It does not pass control to an associated exit routine. RSCS then continues its standard processing. If an exit point is enabled but RSCS cannot find any exit routines, RSCS also continues its processing.

## IBM-Defined Exit Points

RSCS contains 48 *IBM-defined exits* that are identified by `EXITCALL` macros in the RSCS source code. In this document, each IBM-defined exit point is identified by its number (for example, Exit 0, Exit 1, and so on). Chapter 3, “IBM-Defined Exit Points,” on page 39 contains information about each IBM-defined exit point.

## Installation-Defined Exit Points

You can also establish your own exit points if the IBM-defined exits do not meet your needs. To use these exits, called *installation-defined exits*, add `EXITCALL` macros from within RSCS source code to your exit routine code.

## Invoking the RSCS Exit Facility

The following table summarizes the steps you need to take to load the exit facility. Chapter 2, “Customizing RSCS,” on page 9 contains more information about writing exit routines; it also contains some examples of how to start the exit facility.

1. Define and enable an exit point with an `EXIT` statement or command.
2. Use `VMFHLASM` to assemble the module that contains the exit routine.
3. Use `VMFLKED` to link-edit the resultant text file into a load library.
4. Specify the load library name on the `GCS GLOBAL` command in the `PROFILE GCS` file.

The `RSCSEXIT LOADLIB` contains all of the IBM-supplied sample exit routines. To use one of these exit routines, you must specify the name of this load library in the `PROFILE GCS` file.

5. Initialize RSCS.

As an alternative to building your own exit load library, you can add your exit module to the `RSCSEXIT LOADLIB` and use `VMSES/E` to rebuild the library.

## Transmission Algorithms

RSCS supplies two transmission algorithms: DMTAXAG0 and DMTAXAG1. **You do not need to provide additional transmission algorithms.** However, as the needs of your installation change, you can modify these transmission algorithms or create your own, packaged in a separate load library. These transmission algorithms are called *external* transmission algorithms. For compatibility, RSCS also provides facilities for *internal* transmission algorithms. See [Chapter 4, “Transmission Algorithm Processing,”](#) on [page 137](#) for more information about external and internal transmission algorithms.

### How Transmission Algorithms Work

RSCS uses a transmission algorithm when a networking link starts, a file is enqueued on a networking link, and when the link tries to get a file for transmission.

A transmission algorithm determines the stream on which a *multistreaming* link sends a file. Multistreaming allows RSCS to send more than one file at a time over a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, and TCPNJE-type). RSCS can send several small files at the same time it sends a large file; each file is sent concurrently on different transmission streams. This reduces the time small files wait to be sent on a link.

### Invoking Transmission Algorithms

Transmission algorithms are called by specifying operands on the PARM statement or the START and DEFINE commands for a networking link. The TA operand identifies the transmission algorithm used for the link. The STREAMS operand specifies the number of transmission streams. NJE-type, SNANJE-type, TCPNJE-type, and LISTPROC-type links can use up to seven transmission streams to send files; GATEWAY-type links can use up to 32 streams.

The TAPARM operand specifies an 80-character string for use as input to a transmission algorithm. If you specify the same transmission algorithm for more than one link, you can supply a different set of control values for each link. For more information, see [z/VM: RSCS Networking Operation and Use](#).

See [“Packaging Transmission Algorithms”](#) on [page 144](#) for information about installing external and internal transmission algorithms.

## ASCII Printer and Plotter Exits

ASCII-type and TCPASCII-type links use the ASCII printer and plotter exit routines to build specific data streams to communicate with ASCII printers and plotters. On ASCII-type links, these devices can be connected to RSCS by a controller, such as the IBM 7171 ASCII Device Attachment Control Unit or the 9370 ASCII Subsystem Controller. On TCPASCII-type links, the devices can be connected to a terminal server in a TCP/IP network.

**Note:** Throughout this book, *ASCII exits* means exits for both ASCII-type and TCPASCII-type links.

ASCII printer and plotter exit routines are independent of RSCS code. RSCS provides sample ASCII printer and plotter exit routines. Use these samples to choose the appropriate data translation needed for a specific ASCII printer or plotter. You can also create other processing modules, using the sample modules as guides.

**You need to code ASCII printer and plotter exit routines only if your installation uses ASCII-type or TCPASCII-type links.** If your installation uses an ASCII-type or TCPASCII-type link, you must provide an exit routine to customize the link driver for the ASCII device. See [“Sample ASCII Printer and Plotter Exit Modules”](#) on [page 156](#) for more information.

### How ASCII Printer and Plotter Exits Work

The ASCII or TCPASCII link driver calls an ASCII printer and plotter exit to build specific data streams, which are sent to a printer or plotter. These exit routines are called at seven points in the ASCII or TCPASCII link driver's operational cycle:

- Attention interrupt processing

- Initialization
- End-of-file processing
- Message handling
- Output record translation
- TAG record processing for spool files
- Termination

Your exit routine can contain six subroutines that provide additional processing instructions for each of the six points. You need only four of these subroutines in your exit routines; two are optional. Generally, ASCII printer and plotter exit routines respond to conditions set by the CP TAG, SPOOL, and CLOSE commands, and to the characteristics of a particular printer or plotter.

For more information, see [Chapter 5, “ASCII Printer and Plotter Exit Processing,” on page 147.](#)

## Invoking ASCII Printer and Plotter Exits

To start ASCII printer and plotter exit routines, take the following steps:

1. Create one or more exit routines and identify each exit routine by an entry point name.
2. Use VMFHLASM to assemble the module containing the exit routines.
3. Use VMFLKED to link-edit the resultant text file into a load library (or add your exit module to RSCSEXIT LOADLIB and use VMSES/E to rebuild the library).
4. Specify the load library on the GCS GLOBAL command.
5. Specify the module name on the EXIT operand of the DEFINE or START command or the PARM statement for the ASCII-type or TCPASCII-type link.
6. Initialize RSCS.

## Gateway Programming Interface

The gateway programming interface (GPI) lets you create routines that enable RSCS to communicate with systems using various network protocols. The GPI is made up of the GATEWAY-type links, gateway programs, and gateway service macros. Gateway programs, like other exit routines, are external from the rest of the RSCS source code. For more information, see [Chapter 6, “Gateway Programming Interface,” on page 167.](#)

## Invoking a Gateway Program

To start a gateway program and the gateway programming interface, perform these steps:

1. Create a gateway program, including gateway service macros, and identify its entry point name.
2. Use VMFHLASM to assemble the module containing the gateway program.
3. Use VMFLKED to link-edit the resultant text file into a load library (or add your exit module to RSCSEXIT LOADLIB and use VMSES/E to rebuild the library).
4. Specify the load library on the GCS GLOBAL command.
5. Specify the module name of the gateway program on the EXIT operand of the DEFINE or START command or on the PARM statement for the GATEWAY-type link.
6. Initialize RSCS.

## LPR Exits

The LPR link drivers use exits to build data streams that are sent to a line printer daemon in a TCP/IP network. These exits are also used to control which remote host and port in a TCP/IP network will receive the transmission.

**You need to code LPR exit routines only if your installation uses LPR-type links.** If your installation uses an LPR-type link, you must provide an exit routine for the link driver. RSCS provides sample LPR exit routines; see [“Sample LPR Exit Routines” on page 200](#) for more information.

### How LPR Exits Work

The LPR exits are called at six points in the operational cycle of the LPR link drivers:

- Control file processing
- End-of-file processing
- Initialization
- Record processing
- TAG record processing for spool files
- Termination

For more information, see [Chapter 7, “TCP/IP LPR Exit Points,” on page 187](#).

### Invoking LPR Exits

To start LPR exit routines, perform these steps:

1. Create one or more exit routines and identify each exit routine by an entry point name.
2. Use VMFHLASM to assemble the module containing the exit routines.
3. Use VMFLKED to link-edit the resultant text file into a load library (or add your exit module to RSCSEXIT LOADLIB and use VMSES/E to rebuild the library).
4. Specify the load library on the GCS GLOBAL command.
5. Specify the module name on the EXIT operand of the DEFINE or START command or the PARM statement for the LPR-type link.
6. Initialize RSCS.

## LPD Exits

The LPD link drivers use exits to build data streams that are received from an LPR client in a TCP/IP network. These exits are also used to control the destination node and user ID the file will be delivered to.

**You need to code LPD exit routines only if your installation uses LPD-type links.** If your installation uses an LPD-type link, you must provide an exit routine for the link driver. For more information, see [“Sample LPD Exit Routine” on page 224](#).

### How LPD Exits Work

The LPD exits are called at seven points in the operational cycle of the LPD link drivers:

- Control file processing
- Data processing
- End-of-file processing
- Initialization
- Print command processing
- Print job command processing
- Termination

For more information, see [Chapter 8, “TCP/IP LPD Exit Points,” on page 211](#).

### Invoking LPD Exits

To start LPD exit routines, perform these steps:

1. Create one or more exit routines and identify each exit routine by an entry point name.
2. Use VMFHLASM to assemble the module containing the exit routines.
3. Use VMFLKED to link-edit the resultant text file into a load library (or add your exit module to RSCSEXIT LOADLIB and use VMSES/E to rebuild the library).
4. Specify the load library on the GCS GLOBAL command.
5. Specify the module name on the EXIT operand of the DEFINE or START command or the PARM statement for the LPD-type link.
6. Initialize RSCS.

## UFT Exits

The UFT link drivers use exits to build data streams that are sent to a UFT daemon in a TCP/IP network. These exits are also used to control which remote host, user, and port in a TCP/IP network will receive the transmission.

**You need to code UFT exit routines only if your installation uses UFT-type links.** If your installation uses a UFT-type link, you must provide an exit routine for the link driver. For more information, see [“Sample UFT Exit Routine” on page 240.](#)

## How UFT Exits Work

The UFT exits are called at six points in the operational cycle of the UFT link drivers:

- End-of-file processing
- Initialization
- Record processing
- TAG record processing for spool files
- Termination
- UFT command processing

For more information, see [Chapter 9, “TCP/IP UFT Exit Points,” on page 231.](#)

## Invoking UFT Exits

To start UFT exit routines, perform these steps:

1. Create one or more exit routines and identify each exit routine by an entry point name.
2. Use VMFHLASM to assemble the module containing the exit routines.
3. Use VMFLKED to link-edit the resultant text file into a load library (or add your exit module to RSCSEXIT LOADLIB and use VMSES/E to rebuild the library).
4. Specify the load library on the GCS GLOBAL command.
5. Specify the module name on the EXIT operand of the DEFINE or START command or the PARM statement for the UFT-type link.
6. Initialize RSCS.

## UFTD Exits

The UFTD link drivers use exits to build data streams that are received from a UFT client in a TCP/IP network. These exits are also used to control the destination node and user ID the file will be delivered to.

**You need to code UFTD exit routines only if your installation uses UFTD-type links.** If your installation uses a UFTD-type link, you must provide an exit routine for the link driver. For more information, see [“Sample UFTD Exit Routine” on page 257.](#)

### How UFTD Exits Work

The UFTD exits are called at six points in the operational cycle of the UFTD link drivers:

- Command processing
- Connect processing
- Data processing
- End-of-file processing
- Initialization
- Termination

For more information, see [Chapter 10, “TCP/IP UFTD Exit Points,”](#) on page 245.

### Invoking UFTD Exits

To start UFTD exit routines, perform these steps:

1. Create one or more exit routines and identify each exit routine by an entry point name.
2. Use VMFHLASM to assemble the module containing the exit routines.
3. Use VMFLKED to link-edit the resultant text file into a load library (or add your exit module to RSCSEXIT LOADLIB and use VMSES/E to rebuild the library).
4. Specify the load library on the GCS GLOBAL command.
5. Specify the module name on the EXIT operand of the DEFINE or START command or the PARM statement for the UFTD-type link.
6. Initialize RSCS.

### Loadable Link Drivers

RSCS provides the LINKTYPE configuration statement for you to define additional types of link drivers. You must supply the routine for the new type of driver; this routine is also separate from the RSCS code. Like other exit routines, you can use RSCS facilities, such as macros and other routines, within the link driver routine.

You identify the new link driver to RSCS by specifying a LINKTYPE statement in the RSCS configuration file. You must also specify the entry point name of your link driver routine. For more information about this statement, see [z/VM: RSCS Networking Planning and Configuration](#).

### Sample Exit Routines

---

RSCS supplies sample exit routines and exit packages for use with the IBM-defined exit points, the gateway programming interface, and the ASCII, LPD, LPR, UFT, and UFTD exits. For more information, see [“Using Sample Exit Packages”](#) on page 36 or see the section about supplied sample packages in the RSCS program directory.

## Chapter 2. Customizing RSCS

This section contains information to help you write exit routines. It contains a summary of all exit categories, describes programming considerations, and provides suggestions for implementing exits.

### Selecting Which Exits to Use

Each exit category can be used to customize different areas of RSCS. Before altering standard RSCS functions or adding new functions to RSCS, you must determine the type of exit you will need to use. You may find that you need to use more than one type of exit to accomplish the desired changes in RSCS function. Each type of exit in RSCS has a specific purpose. If you use an exit for other than its intended purpose, you can increase the risk of performance degradation and system failure.

### RSCS Exit Facility

The RSCS exit facility allows you to modify and extend the processing of standard RSCS functions. The following table lists the IBM-defined exit points in the RSCS exit facility. For more information, see [Chapter 3, “IBM-Defined Exit Points,”](#) on page 39.

Exit	Name	Function
0	Initialization	Perform additional initialization processing.
1	Termination	Perform additional termination processing.
2	Spool File Accept Accounting	Create an accounting record for each spool file RSCS receives from a local user.
3	Spool File Send Accounting	Create or modify an accounting record for each spool file RSCS transmits on a link.
4	Spool File Purge Accounting	Create an accounting record for each spool file deleted by the PURGE command.
5	Spool File Receive Accounting	Create or modify the accounting record for each file RSCS receives on a link.
6	TAG Priority Change	Ensure that the TAG priority option is not misused or change the priority of a file before RSCS queues it on one or more links.
7	Auto-answer Sign-on Time Out	Create an accounting record when the sign-on time out for an auto-answer port expires.
8	Auto-answer Unrecognizable Data	Create an accounting record when RSCS receives unrecognizable data from an auto-answer port.
9	Auto-answer Sign-on Validation	Perform additional processing when RSCS receives a valid sign-on card from an auto-answer port.
10	Auto-answer Sign-on Reject	Perform additional processing when RSCS receives a valid sign-on card, which was rejected by the associated link, from an auto-answer port.
11	NJE Job Header Creation	Perform additional processing of the NJE headers created by RSCS.
12	NJE Data Set Header Creation	Perform additional processing of the data set headers created by RSCS.
13	NJE Job Trailer Creation	Perform additional processing of NJE trailers created by RSCS.

Exit	Name	Function
14	NJE Job Header Reception	Perform additional processing of the NJE header, as received by RSCS, before RSCS updates the TAG element.
15	NJE Data Set Header Reception	Perform additional processing of the data set header, as received by RSCS, before RSCS updates the TAG element.
16	NJE Job Trailer Reception	Perform additional processing of the NJE trailer, as received by RSCS, before RSCS updates the TAG element.
17	Separator Page Selection	Select the separator page style for printed files.
18	Separator Page Generation	Create an alternative header or trailer page for printed files.
19	Command Screening	Determine if RSCS should process a command or modify a command before processing it.
20	(no longer available)	Exit 27 and Exit 28 replace the functions of Exit 20, which was defined in RSCS V2.3.
21	Spool File Accept/Reject	Determine criteria for accepting or rejecting incoming spool files.
22	NOTIFY Driver Note Selection	Determine if a NOTIFY-type link issues a note to the originator of a misdirected file.
23	NOTIFY Driver Note Editing	Modify the note that the NOTIFY-type link sends to the originator of a misdirected file.
24	Spooling CP Command Screening	Examine and, optionally, modify the CP commands run by the RSCS spool manager task.
25	Post-CP Command Screening	Examine the return codes from the CP commands run by the spool manager task. Use this exit with Exit 24.
26	Link State Change Accounting	Create accounting records when a link changes state.
27	Message Request Screening	Modify, log, or suppress an RSCS message.
28	Message Language Selection	Change the language which RSCS uses to issue a message.
29	Unknown Command	Examine or process any command that RSCS does not recognize.
30	Reroute	Reroute data traffic using criteria defined at your installation.
31	Sort Priority Change	Change the sort priority of a file's tag shadow elements before they are queued on links, or prevent files from being transmitted on a link.
32	NMR Reception	Determine if RSCS should process a command or message element received on a networking link, or modify the element before RSCS processes it.
33	User Parm Processing	Process values specified on the UPARM operand of the DEFINE command.
34	Spool Command Screen	Determine if the spool manager task should process a request from another task, or modify the command element before it is processed by the spool manager task.
35	Dump Processing	Determine if RSCS should request a dump when a task abends.
36	NOTIFY Driver Purge	Determine if RSCS should purge a file queued on a NOTIFY-type link after its requested retention period has expired.
37	NJE Job Header Transmission	Examine job headers before RSCS sends a store-and-forward file on a networking link.



Exit	Name	Function
38	NJE Dataset Header Transmission	Examine data set headers before RSCS sends a store-and-forward file on a networking link.
39	NJE Job Trailer Transmission	Examine job trailers before RSCS sends a store-and-forward file on a networking link.
40	NJE Record Reception	Examine records, other than NJE headers, as RSCS receives a file on a networking link.
41	NJE Job Header Post-Processing	Perform additional processing of the NJE header, as received by RSCS, after RSCS updates the TAG element.
42	NJE Data Set Header Post-Processing	Perform additional processing of the data set header, as received by RSCS, after RSCS updates the TAG element.
43	NJE Job Trailer Post-Processing	Perform additional processing of the NJE trailer, as received by RSCS, after RSCS updates the TAG element.
44	Link Termination	Perform special processing needed for a print output link.
45	Output Page Accounting	Perform accounting processes for printed output.
46	Verification of Page Accounting	Perform adjustments to accounting information for output printed on 3270P-type link.
47	Driver Initialization	Perform any required initialization for an SNA3270P link driver.
48	Verification of Output Page Error	Perform any special processing if an error occurs while an SNA3270P link driver is processing a file.

## Transmission Algorithms

Transmission algorithms specify the way in which RSCS allocates multiple streams to send files over a networking link. IBM supplies transmission algorithms 0 and 1, which perform the following functions:

### TA 0

Selects any file and assigns it to any available stream.

### TA 1

Selects files based on the number of records or spool file blocks in the file. The transmission algorithms can use one to seven streams. The TAPARM value specifies the upper and lower limit for each defined stream. The threshold value for transmission algorithm 1 is 99,999,999 file records.

You can modify transmission algorithm 0 or 1, or define your own transmission algorithms to meet your installation's needs. You can also supply transmission algorithms in a separate load library. RSCS also provides facilities which you can use to define new transmission algorithms 2 - F. For more information, see [Chapter 4, "Transmission Algorithm Processing," on page 137](#).

## ASCII Printer and Plotter Exits

ASCII printer and plotter exits specify the protocol used when an ASCII-type or TCPASCII-type link communicates with a specific ASCII device. Sample exit routines are located in each of the ASCII printer and plotter processing modules. You can also write other processing modules, using the sample modules as guides. IBM supplies sample exit routine modules for the following ASCII devices:

### ASCXDWRE

LA120 DECwriter Printer from DEC

### ASCXSPWE

NEC 3515 Spinwriter Printer

### ASCXDSOE

DS180 Matrix Printer from Datasouth

### **ASCXPROP**

IBM Proprinter

### **ASCX749E**

IBM Instruments XY/749 Multipen Digital Plotter

### **ASCXZETE**

Nicolet Zeta 8 Plotter

### **ASCXPSE**

PostScript® printers

### **ASCXONE**

Generic (non-PostScript) printers

For more information, see [Chapter 5, “ASCII Printer and Plotter Exit Processing,”](#) on page 147.

## Gateway Programming Interface

The gateway programming interface (GPI) specifies how RSCS uses a customer-defined networking protocol. Unlike the ASCII link driver, which provides many ASCII link driver functions and calls exit routines to perform specific tasks, the gateway link driver calls one exit routine that does all link driver tasks. The gateway programming interface provides high-level access to RSCS and NJE-related services through various macros supplied by IBM. For more information, see [Chapter 6, “Gateway Programming Interface,”](#) on page 167.

## LPD Exits

LPD exit routines build data streams received from an LPR client in a TCP/IP network. These exit routines enable you to modify how files are received. IBM supplies sample exit routines for use with the LPD exits. For more information, see [Chapter 8, “TCP/IP LPD Exit Points,”](#) on page 211.

## LPR Exits

LPR exit routines build device-specific data streams that are sent to a line printer daemon in a TCP/IP network. These exit routines enable you to modify how files are printed. IBM supplies sample exit routines for use with the LPR exits. For more information, see [Chapter 7, “TCP/IP LPR Exit Points,”](#) on page 187.

## UFT Exits

UFT exit routines build data streams that are sent to a UFT daemon in a TCP/IP network. These exit routines enable you to modify how files are delivered. IBM supplies sample exit routines for use with the UFT exits. For more information, see [Chapter 9, “TCP/IP UFT Exit Points,”](#) on page 231.

## UFTD Exits

UFTD exit routines build data streams received from a UFT client in a TCP/IP network. These exit routines enable you to modify how files are received. IBM supplies sample exit routines for use with the UFTD exits. For more information, see [Chapter 10, “TCP/IP UFTD Exit Points,”](#) on page 245.

## Loadable Link Drivers

The loadable link driver facility lets you define a new RSCS link driver without using the GPI. You can use the LINKTYPE configuration statement to define a new link driver type, associate an entry point name to it, and define its basic characteristics to RSCS. The link driver routine's characteristics should be consistent with the RSCS interpretation of the link type's defined characteristics. For more information about the LINKTYPE statement, see [z/VM: RSCS Networking Planning and Configuration](#).

## Exit Routine Considerations

The following sections contain information you will need to write exit routines.

### Language Requirements

All RSCS source code is written in basic assembler language (BAL). Your exit routines should also be written in BAL.

### Code Attributes

Exit routine code may be *serially reusable* or *reentrant*. A serially reusable exit routine can be used repeatedly. However, the current processing of the routine must be completed before the code can be used again. A reentrant exit routine can be called from more than one task at the same time. Each task that calls the exit routine has its own set of registers and program status word. However, all tasks use the same base register for the routine and the same data areas that are part of the routine.

When writing a routine for a reentrant exit point, you must ensure that the exit routine does not contain self-modifying code. You must also ensure that any storage used as a work area is based on registers that are specific to each task (not the base register). You can use locking mechanisms to control access to common data areas within an exit routine.

In the RSCS exit facility, the calling environment determines if an exit routine must be reentrant or serially reusable. For example, exit routines called from the RSCS spool manager task do not have to be reentrant, while those called from link driver tasks must be reentrant. The reentrancy requirements for each exit point are described in [Chapter 3, “IBM-Defined Exit Points,”](#) on page 39.

Your transmission algorithms can be written to be serially reusable.

Gateway programs and ASCII, LPR, LPD, UFT, and UFTD exit routines can be nonreusable, serially reusable, or reentrant. Base your decision on the characteristics of your exit routine and the information in [“Link-Editing Considerations”](#) on page 33.

### 31-Bit Enablement

RSCS is enabled to use 31-bit addressing and must run in an ESA-mode virtual machine. Your exit routines should also be enabled to use 31-bit addressing. They should also tolerate ESA mode and be able to run in an ESA-mode virtual machine.

If you are migrating from RSCS V3.1 or earlier, or creating new exit routines, you should take the following actions to ensure that your exit routines are 31-bit compliant:

- Update assembler instructions for 31-bit compliance.

For example, you should update the following instructions within your exit routines. For more information about other programming changes that are required for 31-bit compliance, see [Enterprise Systems Architecture/390 Principles of Operation \(publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf\)](http://publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf).

Instruction	Changes for 31-Bit Compliance
BAL	Change to BAS.
BALR	Change to BASR; use IPM to obtain a condition code in a register.
CCW	Change to format 1.
CSW	Change to SCSW.
DC and DS	Ensure any addresses specified are 4 bytes long.
ICM and STCM	Verify any instructions that process an address in a register and have a mask that uses that address.

Instruction	Changes for 31-Bit Compliance
LA	No longer clears the high-order byte of a register; only the high-order <b>bit</b> is cleared.
<ul style="list-style-type: none"> <li>• Ensure the options specified on the GETMAIN and FREEMAIN macros are compatible with 31-bit addressing. For more information, see <a href="#">z/VM: Group Control System</a>.</li> <li>• Reassemble your exit routines with the macro libraries that are supplied with RSCS.</li> <li>• Specify the AMODE 31 RMODE ANY options on the %LEPARMS statement of an LKEDCTRL file or on the :OPTIONS. statement of a VMSES/E build list.</li> </ul>	

## Return Codes

When an exit routine returns control to RSCS, it must contain a return code in register 15. The return code tells RSCS about the type of processing the exit routine performed. It also tells RSCS how it is to continue its processing. The number of return codes and their meanings to RSCS varies for each type of exit routine. “Standard Return Codes” on page 39 describes the guidelines each exit routine should follow when issuing a return code. For specific information, see the return codes section of each exit point description.

## Writing Exit Routines

The following sections contain code and macro samples to show the concepts that are involved in writing exit routines. Although these samples focus on the RSCS exit facility, the concepts can also be applied to transmission algorithms, ASCII, LPD, LPR, UFT, and UFTD exit routines, and gateway link driver routines.



**Attention:** Before attempting to write any RSCS exit routines, you should completely review this section and understand the implications described.

## Calling Conventions

The RMOD macro must be the first statement in any module containing exit routine code; it identifies the name of the module, **which must contain 6 characters**. Depending on the options specified, RMOD may generate other statements (TITLE, CSECT) and eye-catchers to identify the module. For more information, see “RMOD – Defining a Module” on page 297.

The RENTRY macro identifies entry points within the module. The entry point name specified on the macro **must** contain the 6-character name specified on RMOD and 2 characters to identify the entry point. If the same 6-character name is not specified, external routines will not be able to access the entry point. Here, the resulting entry point will be accessible only by other routines in the module that contains the RENTRY macro. RENTRY may also be used to generate additional statements and establish savearea conventions. For more information, see “RENTY – Defining a Module Entry Point” on page 290.

The combination of SAVAREA and RENT parameters on the RENTRY macro may be critical to the performance of an exit routine. Table 2 on page 14 identifies the **required** values for each category of exit routine. If you do not use the following values, your exit routine must get and free its own save areas. If you do not manage your save areas properly, you may cause RSCS to abend.

Table 2. Required RENT and SAVAREA Settings

Exit Category	Reentrant/Reusable	RENT and SAVAREA Settings
Exit Facility	Reentrant Serially Reusable	RENT=YES, SAVAREA=PREALLOC RENT=NO, SAVAREA=PREALLOC
Transmission Algorithm	Serially Reusable	RENT=NO, SAVAREA=PREALLOC
ASCII	Reentrant Serially Reusable	RENT=YES, SAVAREA=PREALLOC RENT=NO, SAVAREA=PREALLOC

Table 2. Required RENT and SAVAREA Settings (continued)

Exit Category	Reentrant/Reusable	RENT and SAVAREA Settings
Gateway Program	Reentrant Serially Reusable Nonreentrant	RENT=YES , SAVAREA=YES RENT=NO , SAVAREA=YES
Loadable Link Driver	Reentrant Serially Reusable	RENT=YES , SAVAREA=YES RENT=NO , SAVAREA=YES
LPD	Reentrant Serially Reusable	RENT=YES , SAVAREA=PREALLOC RENT=NO , SAVAREA=PREALLOC
LPR	Reentrant Serially Reusable	RENT=YES , SAVAREA=PREALLOC RENT=NO , SAVAREA=PREALLOC
UFT	Reentrant Serially Reusable	RENT=YES , SAVAREA=PREALLOC RENT=NO , SAVAREA=PREALLOC
UFTD	Reentrant Serially Reusable	RENT=YES , SAVAREA=PREALLOC RENT=NO , SAVAREA=PREALLOC

The SAVAREA=PREALLOC option means that RSCS provides the exit routine with a previously allocated save area chain. If you do not specify SAVAREA=PREALLOC, you must manage the save area chain yourself. Also, if you need a work area for the exit routine, it is recommended that you use DMTQSA to get the storage for the work area. For more information, see [“Example 4: Mapping a Work Area” on page 26](#).

The listing that is produced when you assemble the exit routine can identify errors on RENTRY macro specifications. For example, you may find that the macro generates incorrectly or that unnecessary processing is taking place.

## Save Area Format

When the calling routine passes control to another routine, register 13 contains the address of the save area. This save area, which is mapped by the SAVEAREA macro, has the following format.

Word	Displacement	Contents
1	0	First 3 bytes are reserved; the fourth byte contains the number of saveareas in the chain
2	4	Address of the previous save area (stored by the calling program)
3	8	Address of the next save area (stored by the current program)
4	12	Register 14 (return address)
5	16	Register 15 (entry point address)
6	20	Register 0
7	24	Register 1
8	28	Register 2
9	32	Register 3
10	36	Register 4
11	40	Register 5

Word	Displacement	Contents
12	44	Register 6
13	48	Register 7
14	52	Register 8
15	56	Register 9
16	60	Register 10
17	64	Register 11
18	68	Register 12
19	72	Work area word 1
20	76	Work area word 2
21	80	Work area word 3
22	84	Work area word 4
23	88	Work area word 5
24	92	Work area word 6
25	96	Work area word 7
26	100	Work area word 8

The fourth byte in Word 1 of the save area tells the exit routine about the number of save areas that remain in the chain. Each time an exit routine calls another routine, one of the save areas in the chain is used; if that routine calls another routine, another save area is used.

If an exit routine calls an entry point that has SAVAREA=PREALLOC specified on its RENTRY macro and the indicator is pointing at the last save area in a chain, a protection exception will occur. RSCS will then create a dump, with R1 pointing at the last save area in the chain.

## Linkage Conventions

RSCS uses the following standard Operating System (OS) linkage conventions when calling exit routines:

### **R0**

May contain a function code

### **R1**

Points to a parameter list

### **R13**

Points to a save area

### **R14**

Contains the return address

### **R15**

Contains the address of the routine.

When an exit routine calls the RCALL macro to call another routine whose entry point is defined by the RENTRY macro, including any RSCS routines that exit routines can access, standard RSCS calling conventions are ensured.

## Restoring Registers

As the default, the REXIT macro restores registers 2 - 12 before returning to the calling routine. Generally, however, you should also restore registers 0 and 1. This is especially important if several exit routines are associated with an exit point or if RSCS repeatedly calls an exit routine. If your exit routine issues return code 0, which tells RSCS to call the next associated exit routine, you should specify REGS=(0,12) on

the REXIT macro. This ensures that registers 0 and 1 are restored before the exit routine returns control to RSCS. If not, RSCS may pass altered registers when it calls the next exit routine; this may produce unpredictable results.

## Using RSCS Facilities

Exit routines can use many of the facilities RSCS provides. These facilities include some RSCS data structures, macros, and various utility routines. Exit routines can also coordinate their processing with other RSCS facilities, such as the event scheduler (see [Figure 2 on page 18](#)). Often, these facilities can simplify or enhance the function of an exit routine.

### Data Areas

The communications vector table (CVT) and the common routines vector table (CRV) are particularly important to exit routines. The CVT, which is passed to all exit routines and transmission algorithms, contains pointers to other RSCS data areas, counters, certain command queues, and flags.

Using the TCRVTAB field in the CVT, an exit routine can find the CRV. The CRV contains the addresses of some RSCS routines that can be accessed by an exit routine.

The TUSER field in the CVT can be used to share information among different exit points of an exit package. [“Sharing Information” on page 30](#) describes how to allow multiple exit packages to utilize this field. If the guidelines are not followed, data will not be shared properly, possibly leading to an RSCS outage.

Depending on their intended function, an exit routine may also be passed pointers to other data structures. For more information, see the entry conditions section in the description of each exit routine. For a list of RSCS data areas supported as programming interfaces, see [Appendix A, “DSECTs Generated by Mapping Macros,” on page 367](#).

### Macros

RSCS provides macros that you can use within your exit routines. With these macros, you can define entry points for your exit routines, determine how exit routines access information, and issue messages. For more information about these macros, see [Chapter 11, “RSCS Macros,” on page 265](#).

RSCS also provides gateway service macros that provide functions for gateway programs. Using these macros, a gateway program can send, receive, and examine files and network job entry (NJE) records from a GATEWAY-type link. [“Gateway Service Macros” on page 174](#) contains more information about these macros.

### Routines

Using the CRV, exit routines can call RSCS routines to perform functions, such as queuing and dequeuing request elements, searching tables, issuing messages, and managing storage. For more information about the RSCS routines supported for use by exit routines, see [Chapter 12, “Supported Routines in the CRV,” on page 321](#).

DMTAXA contains transmission algorithm 0 and transmission algorithm 1. For compatibility with previous releases of RSCS, DMTAXA also contains entry points for transmission algorithms 2 - F. For more information, see [“Internal Transmission Algorithms” on page 142](#).

## Issuing Messages

Exit routines can use the RMSG macro to issue messages from the message repositories supplied by RSCS. RMSG can also be called to issue messages from other repositories, which you can create at your installation. [Chapter 13, “Message Repositories,” on page 345](#) describes the structure of the message repositories. Your exit routine can use RMSG to set various MSGBLOK fields to describe the selected message. RSCS then uses this information to issue the message. For more information, see [“RMSG – Issuing a Message” on page 299](#).

## Calling Exit Routines

After you have created an exit routine, you should assemble the file, link-edit it, and create a load library. To assemble the file, use the VMFHLASM exec and specify a control file, if needed. To link-edit assembled files into a load library, you can use the VMFLKED exec and specify a link-edit control file, as needed. This control file, with file type LKEDCTRL, contains statements that describe the characteristics of the load library. [Figure 4 on page 19](#), [Figure 7 on page 21](#), and [Figure 11 on page 24](#) are examples of link-edit control files. For more information, see [“Link-Editing Considerations” on page 33](#). Alternatively, you can add your exit routine to the IBM-supplied RSCSEXIT LOADLIB and use VMSES/E to rebuild the library.

### Example 1: Defining Printing Shifts

This example demonstrates the conventions described in the preceding sections. For this example, assume that you want to prevent files larger than 5000 records from printing on any printers attached to your local node during prime shift. This example uses the RSCS event scheduling facilities and Exit 31 (Sort Priority Change).

To define the daily prime shift as the period between 8 AM and 5 PM, you use the RSCS SCHEDULE command. You then create the EVENTS CONFIG file in [Figure 2 on page 18](#). For more information, see [z/VM: RSCS Networking Planning and Configuration](#).

```
*----- EVENTS CONFIG File -----*
*
*      Date      Time      Days-of-      Range
*      Date      Time      the-week     Low   High   Command
*-----*
*      *      *      *      *      *
*      *      *      *      *      *
*      *      *      *      *      *
*      *      *      *      *      *
*      *      *      *      *      *
*      *      *      *      *      *
*      *      *      *      *      *
```

Figure 2. Defining Shifts with an EVENTS CONFIG File

To prevent files that exceed 5000 records from printing during prime shift, you create an exit routine for Exit 31 ([Figure 3 on page 18](#)).

```
1 PLIMIT  RMOD
*
    USING CVT,R6          Get CVT addressability
    USING TAG,R7          Get TAG addressability
    USING LINKTABL,R9     Get LINKTABL addressability
2 PLIMITEP RENTRY RENT=NO,SAVAREA=PREALLOC,ARGS=(@CVT,@TAG,@SHAD,@LINK, X
    &REORDER)
*
    L    R6,@CVT          Get CVT address
    L    R7,@TAG          Get TAG address
    L    R8,@LINK         Get LINKTABL address
    SR   R15,R15          Set zero return code
    TM   LACTYP1,LNET     Is this a networking driver?
    BO   PLEXIT           Yes ... only interested in printers
    CLC  TSHIFT,=F'1'     Is this prime shift
    BNE  PLEXIT           No ... only interested in shift 1
    CLC  TAGRECNM,=F'5000' Does file exceed 5000 records?
    BNH  PLEXIT           No ... let it print
    LA   R15,8            Set return code of 8
    PLEXIT EQU *
    REXIT RC=(R15),REGS=(0,12) Return to caller
*
    CVT   DSECT=YES      Communications Vector Table
    LINKTABL DSECT=YES    Link table entry
    TAG    DSECT=YES      Tag slot entry
    END
```

Figure 3. Exit 31 Routine: Stopping Large Files from Printing



The RMOD macro (1) identifies the name of the module, PLIMIT. The RENTRY macro defines an entry point in the module (2). The first 6 characters of the entry point name (PLIMITEP) **must** match the module name (PLIMIT) specified on RMOD. This causes RSCS to create an ENTRY statement that makes this exit routine an externally visible entry point.

The ARGS option on the RENTRY macro defines a temporary mapping DSECT for the parameter list that RSCS provides to this exit routine. Because Exit 31 is not a reentrant exit point, the RENT=NO option is specified on the RENTRY macro of the exit routine code. You should specify RENT=NO for serially reusable exit points and RENT=YES for reentrant exit points.

RSCS also passes a pointer to the CVT to the exit routine. The LINKTABL and TAG DSECTs in the PLIMIT module define mappings for data areas that describe the characteristics of a link and a file, respectively.

## Invoking the Exit Routine

After assembling the source file containing the Exit 31 routine (PLIMIT ASSEMBLE), you also create a link-edit control file. The link-edit control file in [Figure 4 on page 19](#) is used for this example.

```
* PLIMIT Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY PLIMIT
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE PLIMIT
ALIAS PLIMITEP
NAME PLIMIT
```

Figure 4. PLIMIT LKEDCTRL File

You then take the following steps to install the exit routine and enable Exit 31:

1. Issue the following command to assemble the PLIMIT ASSEMBLE file:

```
vmfhlasm plimit rscsv3
```

2. Issue the following command to build the PLIMIT LOADLIB:

```
vmflked plimit
```

3. Copy the PLIMIT LOADLIB to a disk that RSCS can access.
4. Update the GCS GLOBAL command in the RSCS PROFILE GCS file to include the PLIMIT load library (do *not* add a second GLOBAL command).
5. Add the following statement to the RSCS CONFIG file:

```
EXIT 31 ON PLIMITEP
```

6. Restart the RSCS virtual machine.

## Example 2: Using Two Exit Routines

This example demonstrates communication between two IBM-defined exit points: Exit 0 and Exit 11, which is a reentrant exit point. It also shows how exit routines can be link-edited into one load library. This example reads a file called JOBNAM CONFIG from a disk. The only token on the first line of the file is used as the job name for all jobs that originate on the local node and are sent to MVS.

### Exit 0

The Exit 0 routine ([Figure 5 on page 20](#)) reads the JOBNAM CONFIG file and saves the job name, which is later accessed by the Exit 11 routine. In this example, the exit routine does not check for errors in the JOBNAM CONFIG file.

```

3a JOBNO0  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
*
3b          USING CVT,R9          Get CVT addressability
          ENTRY JOBNO0NM          Externalize the JOBNAME field
JOBNO00EP RENTRY RENT=NO,SAVAREA=PREALLOC,ARGS=(@CVT)
*
          L      R9,@CVT          Get CVT pointer from plist
          MVC     FILRNAME,=CL8'JOBNAME'
          MVC     FILRTYPE,=CL8'CONFIG '
*
          MVI     FILROPTS,FILRSIMB Don't want IMBED support
          LA      R1,FILREQ        Point at file request block
3c          RCALL DMTCOMFI        Get first record from the file
          LTR     R15,R15          File there?
          BNZ     JNEXIT          No ... skip processing
          LA      R2,8             Use maximum 8 bytes
          C       R0,=F'8'        Record exceed 8 bytes?
          BH      LENGTHOK        Yes ... use first 8 bytes
          LR      R2,R0           Get the length
          LENGTHOK EQU *
          BCTR    R2,0            Get length - 1
          EX      R2,MOVENAME      Move in the job name
*          MVC     JOBNAME(0),0(R1) Executed by above line
          OI      FILROPTS,FILRCLOS Ask to close file
3c          RCALL DMTCOMFI        Close the file
          EQU     *
          JNEXIT  REXIT RC=0,REGS=(0,12) Return to caller
*
          MOVENAME MVC JOBNAME(0),0(R1) Executed to move job name
*
          JOBNO0NM EQU *
5a          JOBNAME DC CL8' '    Place to store the job name
*
          FILREQ DSECT=NO         File request block for COMFI
          ORG     FILRAMB         Back to MSGBLOK pointer
4          DC      A(MSGBLOK)    Supply an empty message block
          ORG     ,
*
          MSGBLOK DSECT=NO,VARS=2 Message block for use with COMFI
*
          CRV      DSECT=YES
          CVT      DSECT=YES
          END

```

Figure 5. Exit 0 Routine: Reading the JOBNAME CONFIG File

Because the CRVBASE and CRVCALL parameters are specified on the RMOD macro (**3a**), the RCALL macros (**3c**) use the CRV to locate the DMTCOMFI routine. The USING statement (**3b**) establishes CVT addressability, using register 9.

DMTCOMFI is a general purpose RSCS routine that reads files from a disk. The exit routine passes a pointer to a MSGBLOK in the FILREQ block (**4**). DMTCOMFI can use this MSGBLOK to issue diagnostic messages to the RSCS console if problems occur when the file is read.

Because Exit 0 is serially reusable, the job name can be defined in the JOBNAME data area (**5a**) within the exit routine. RENT=NO and SAVAREA=PREALLOC are specified on the RENTRY macro. This means that the exit routine does not need to be reentrant and that RSCS has previously allocated a save area chain. Also, REGS=(0,12) is specified on the REXIT macro to ensure that initial register settings are restored before any other exit routines are called.

## Exit 11

This example also requires an exit routine (Figure 6 on page 21) for Exit 11. Because Exit 11 is called from a networking link driver module and many networking links can be active at the same time, this exit routine must be reentrant.

```

JOB011  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
*
      USING  CVT,R9           Get CVT addressability
      USING  NJH,R8           Get job header addressability
      USING  TIB,R7           Get TIB addressability
JOB011EP RENTRY RENT=YES, SAVAREA=PREALLOC, ARGS=(@CVT,@LINK,@TIB,@JHDR,X
      @TAG,@XAB)
*
      L      R9,@CVT          Get CVT pointer from plist
      LM     R7,R8,@TIB       Get TIB and job header address
      TM     TIBFLAG1,TIBSYSIN Is this SYSIN (i.e a 'JOB')?
      BZ     JNEXIT           No ... don't bother setting job name
5b      L      R1,=V(JOB00NM)   Point at job name
      CLI    0(R1),C' '       Did exit 0 set job name to use?
      BE     JNEXIT           No ... leave things as we find them
      MVC    NJHGJNAM,0(R1)    Move in the job name
JNEXIT  EQU    *
      REXIT  RC=0,REGS=(0,12)  Return to caller
*
      CRV     DSECT=YES
      CVT     DSECT=YES
      NHDTR
      TANK
      NJEEQU
      TIB
      END

```

Figure 6. Exit 11 Routine: Placing the Job Name in Job Header

## Installing the Exit Routines

To build the load module for these exit routines, you use the link-edit control file in Figure 7 on page 21. Although the exit routines are in separate modules, they can be link-edited into one load module. As part of the same load module, the exit routines can share information. Entry points in the exit routine must be externalized by ALIAS statements; RSCS can then load the entry points when it initializes. The link-editor can then resolve the V-type reference (5b) in the Exit 11 routine to locate the JOBNAM data area in the Exit 0 routine.

```

* JOBNAM Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY JOBNAM
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE JOB00
INCLUDE JOB01
ALIAS JOB00EP
ALIAS JOB01EP
NAME JOBNAM

```

Figure 7. JOBNAM LKEDCTRL File

You then take the following steps to build the load module and install the exit routine:

1. Issue the following command to assemble the JOB00 ASSEMBLE file:

```
vmfhlasm jobn00 rscsv3
```

2. Issue the following command to assemble the JOB01 ASSEMBLE file:

```
vmfhlasm jobn01 rscsv3
```

3. Issue the following command to build the JOBNAM LOADLIB:

```
vmflked jobname
```

4. Copy the JOBNAM LOADLIB to an RSCS accessible disk.
5. Update the GCS GLOBAL command in the RSCS PROFILE GCS to include the JOBNAM load library (do not add a second GLOBAL command).

6. Add the following statements to the RSCS CONFIG file:

```
EXIT 00 ON JOBN00EP
EXIT 11 ON JOBN11EP
```

7. Restart RSCS.

### Example 3: Creating a New Command

To show how an exit routine can use some of the RSCS facilities, the following example creates a new command called TYPE. This command allows authorized alternate operators to display the contents of configuration files that reside on RSCS accessible disks.

The TYPE command responses must be compatible with any command response interface (CRI) options that may be specified when the command is issued. For more information about CRI prefixes, see [z/VM: RSCS Networking Operation and Use](#).

Because TYPE is not a standard RSCS command, this example uses Exit 29 (Unknown Command). This exit point can access the addresses of the CVT, the command element, command verb and text, an authorization block (if any), and a preformatted message request block. Using this information, you can write the exit routine shown in the following example.

```

  TPCMD  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
          USING CVT,R9           Map the CVT
          USING CMNDAREA,R7      Map the CMNDAREA
6  TPCMDEP RENTRY RENT=NO,SAVAREA=PREALLOC,ARGS=(@CVT,@CMD,@MSGB,@AUTH, X
          @VERB,@TEXT)
          SR      R15,R15        Set zero return code
          L       R9,@CVT        Get pointer to the CVT
          L       R2,@VERB       Get pointer to command verb
          CLC     CTYPE,0(R2)    Is it our command?
          BNE     EXIT0          Nope ... pass it along the line
          L       R6,@MSGB       Get preformatted MSGBLOK
          LTR     R0,R0          Is command from local console?
          BZ      TYPEOK         Yes ... it has authority
          ICM     R8,B'1111',@AUTH Do we have an AUTHBLOK?
          BZ      TYPM209        Nope ... issue message 209
          CLI     AUTHLINK-AUTHBLOK(R8),C' ' System authorized operator?
          BNE     TYPM209        Nope ... issue message 209
  TYPEOK EQU     *
          XC      FILREQ(FILRLN),FILREQ Zero out the request block
          ST      R6,FILRAMB     Anchor the MSGBLOK that was passed
          MVI     FILROPTS,FILRSIMB Suppress IMBED processing

```

Figure 8. Exit 29 Routine: Implementing the TYPE Command (Part 1 of 3)

```

*
* Set up GPLIST for MPTGP calls
*
      L      R7,@CMD          Point at the L3ALERT
      MVC    START,@TEXT      Set start of text
      XC      LENGTH,LENGTH    Say no previous token
      SR      R5,R5            Clear R5 for insert
      IC      R5,CMNDLEN        Get the element length - 1
      LA      R5,1(R7,R5)       Point past end of text
      ST      R5,TEND           Store end in plist
*
* Find the filename and filetype
*
      LA      R1,GPLIST         Point at plist
7      RCALL  DMTMPTGP          Frame the file name
      BRC     (NSI,TYPM204,TYPM204) Ok, not there, too long
      CLC     LENGTH,=F'8'      Check length against max
      BH      TYPM204            Ooops ... too bad for dad
      MVC     FILRNAME,PARM      Move in the file name
7      RCALL  DMTMPTGP          Frame the file type
      BRC     (NSI,TYPELOOP,TYPM204) Ok, not there, too long
      CLC     LENGTH,=F'8'      Check length against max
      BH      TYPM204            Ooops ... too bad for dad
      MVC     FILRTYPE,PARM      Set the filetype
7      RCALL  DMTMPTGP          Check for junk
      BRC     (TYPM204,NSI,TYPM204) Ok, not there, too long *
* Issue a message for every line in the file
*
TYPELOOP EQU *
8      LA      R1,FILREQ        Point at the FILREQ block
      RCALL  DMTCOMFI          Get the next record
      BRC     (NSI,EXIT8,EXIT8,EXIT8,EXIT8,EXIT8,EXIT8,EXIT8) X
      STM     R0,R1,TYPMSGV0     Store the length and pointer
9      RMSG   100,REPS=(=V(TYPMGCNX),=V(TYPMSGGNX)),BUFFER=(R6) X
      B       TYPELOOP          Loop till end
EXIT8    EQU *
      LA      R15,8              Set return code 8
EXIT0    EQU *
      REXIT   RC=(R15),REGS=(0,12) Return to calling routine
*
* Error messages
*
TYPM204  EQU *
      LA      R15,204            Get message number
      MVC     TYPMSGV0(4),LENGTH Store length of parameter
      MVC     TYPMSGV0+4(4),START And the pointer goes here
      B       MSGISSUE          Go issue the message
TYPM209  EQU *
      LA      R15,209            Get message number
      MVI     TYPMSGV0,X'00'      Say 'command'
      MVC     TYPMSGV1,TYPECMD     Set up TYPE command
MSGISSUE EQU *
10      RMSG   (R15),BUFFER=(R6) Issue the message
      B       EXIT8              And exit cleanly
*
* Data areas
*
      FILREQ  DSECT=NO           Build a file request block

```

Figure 9. Exit 29 Routine: Implementing the TYPE Command (Part 2 of 3)

```

*
*      Plist for DMTMPTGP calls
*
GPLIST  DC      A(PARM)           Pointer to 16 byte parm
START   DC      A(0)             Start of previous token
LENGTH  DC      A(0)             Length of previous token
TEND    DC      A(0)             End of text
PARM    DC      CL16' '          Place for the parm
CTYPE   DC      CL8'TYPE        Command name
TYPECMD DC      F'4',A(CTYPE)    Message version of TYPE command
*
*      DSECTs
*
      AUTHBLOK DSECT=YES          Authorization element
      CMNDAREA DSECT=YES          Command element
      CRV       DSECT=YES          Common Routines Vector
      CVT       DSECT=YES          Communications Vector Table
      MSGBLOK   DSECT=YES          Generate a MSGBLOK
TYPMSGV0 DS      CL8              First variable name
TYPMSGV1 DS      CL8              Second variable name
END

```

Figure 10. Exit 29 Routine: Implementing the TYPE Command (Part 3 of 3)

This exit routine calls the parsing routine, DMTMPTGP (7), and the RSCS file interface routine, DMTCOMFI (8). It also calls the RMSG macro to use the RSCS message facilities. When called, the exit routine receives a formatted message request block (MSGBLOK, 6). DMTCOMFI can then use this MSGBLOK to send any error messages about the specified file to the command originator. This MSGBLOK also allows the TYPE command to work with the CRI facilities.

In Figure 9 on page 23, RMSG is called to issue messages 204 and 209 (10) from the RSCS message repository. This example also shows RMSG specifying message 100, which is in an alternate repository called TYPMSG. The REPS option of the RMSG macro (9) passes the addresses of each alternate repository to RSCS.

For this example, the translation repository the exit routine uses to issue message 100 contains the following statement:

```
100: Text -- $(1)
```

The corresponding conversion repository entry for this message contains the following statement:

```
100 0      I: 1+4!0 AL: 1
```

## Installing the Exit Routine

After assembling the Exit 29 routine (Figure 8 on page 22) into a source file called TYPCMD ASSEMBLE, the link-edit control file shown in Figure 11 on page 24 will build the exit load module.

```

* TYPE Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY TYPE
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE TYPCMD
INCLUDE TYPMSG
INCLUDE TYPMGC
ENTRY TYPCMDEP
NAME TYPCMD

```

Figure 11. TYPE LKEDCTRL File

You then take the following steps to build the load module and install the exit routine:

1. Issue the following command to assemble the TYPCMD ASSEMBLE file:

```
vmfhiasm typcmd rscsv3
```

2. Issue the following command to compile the TYPMSG MSGS file:

```
mcomp typmsg rscsv3
```

3. Issue the following command to compile the TYPMGC MCONV file:

```
mconv typmgc rscsv3
```

4. Issue the following command to build the TYPE LOADLIB:

```
vmflkd type
```

5. Copy the TYPE LOADLIB to an RSCS accessible disk.
6. Update the GCS GLOBAL command in the RSCS PROFILE GCS to include the TYPE load library (do *not* add a second GLOBAL command).
7. Add the following statement to the RSCS CONFIG file:

```
EXIT 29 ON TYPCMD
```

8. Restart RSCS.

## Storage Considerations

To obtain storage, exit routines can use GCS storage management facilities or the quick storage allocation facilities provided by RSCS.

### Using GCS Macros

If an exit routine issues the GCS GETMAIN macro to obtain storage at Exit 0 or from another exit point, the Exit 1 routine should issue a FREEMAIN macro to release the storage. When RSCS shuts down, GCS frees all the storage that was acquired from nonpersistent subpools by GETMAIN macros. Storage that was acquired from persistent subpools remains allocated. For more information, see [z/VM: Group Control System](#).

However, RSCS does not free any storage acquired within an exit routine. If the exit routines within an exit package need one copy of a data area, you can define a copy of the data area in one of the modules that make up the exit package (see “Packaging Considerations” on page 27).

### Quick Storage Allocation Routines

If an exit package requires many copies of a data area or if it requires a work area for use with a reentrant exit point, you can use the RSCS quick storage allocation (QSA) routines to manage the storage. Your exit routines can use the CRV to access the QSA routines. The QSA routines provide an interface to the GCS GETMAIN and FREEMAIN macros and provide the following advantages:

- Exit routines can reuse storage without additional calls to GETMAIN and FREEMAIN; this is useful for work areas in reentrant exit routines.
- Storage can be automatically acquired in 4K pieces and subdivided to meet specific needs; this can also reduce GETMAIN calls.
- Storage can be automatically initialized using a specific template defined by a QSABLOK.
- Storage may be allocated from locations above or below the 16 MB line.

RSCS has predefined the following QSABLOKs for use by exit routines; each QSABLOK is accessible through the CRV.

#### DMTQSAEC

Issues a conditional GETMAIN macro to acquire 256-byte blocks of storage from a persistent subpool.

**DMTQSAEU**

Issues an unconditional GETMAIN macro to acquire 256-byte blocks of storage from a persistent subpool.

**Note:** When GETMAIN=RTYPE is used, a QSABLOK DSECT is needed. The DSECT will be generated when LENGTH=0 is used, or when QSABLOK is specified without parameters.

**Example 4: Mapping a Work Area**

This example shows how the QSA routines can be used with exit routines for Exit 14 and Exit 1. To use the QSA routines, you first create a macro to map the requested work area. The macro includes define constant (DC) instructions to show how each field should be initialized. In [Figure 12 on page 26](#), the macro WORK14 ([11a](#)) defines the requested storage for the Exit 14 routine.

```

EXIT14  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
*
11a      USING CVT,R9          Get CVT addressability
        USING WORK14,R8      Map the work area
EXIT14EP RENTRY RENT=YES,SAVAREA=PREALLOC,ARGS=(@CVT,@LINK,@RIB,@JOBH,X
        @TAG,@XAB,@USEC),ARGBASE=R2
12a      L      R9,@CVT      Get CVT address
        L      R0,V(EXIT01QS) Point at QSABLOK describing the area
        RCALL  DMTQSAAB      Call QSA to allocate buffer
        BRC    (NSI,NOSTOR)  Got it, no storage
        LR     R8,R1         Point at the work area
:
12a      L      R0,V(EXIT01QS) Point at QSABLOK describing the area
        LR     R1,R8         Point R1 at the work area
        RCALL  DMTQSAUB      Call QSA to deallocate buffer
        NOSTOR EQU *
        REXIT RC=0,REGS=(0,12) Return to caller
*
        CRV    DSECT=YES
        CVT    DSECT=YES
        WORK14 DSECT=YES
        END

```

*Figure 12. Exit 14 Routine: Allocating and Deallocating a Work Area*

When the exit routine calls DMTQSAUB to deallocate storage, DMTQSAUB does *not* issue a FREEMAIN macro to release the storage. Rather, DMTQSAUB returns the storage to a pool of free WORK14 areas, which can be reused by other exit routines.

Because the Exit 14 routine acquires storage indirectly, an Exit 1 routine must free the storage. (In contrast, because the Exit 11 routine in [Figure 6 on page 21](#) does not acquire storage, a corresponding Exit 1 routine is not needed.) In this example, the Exit 1 routine also stores the QSABLOK ([12b](#)), which describes the storage requirements, and the WORK14 macro ([11b](#)).



```

EXIT01  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
*
        USING CVT,R9          Get CVT addressability
        ENTRY EXIT01QS        Make point externally visible
EXIT01EP RENTRY RENT=YES,SAVAREA=PREALLOC,ARGS=(@CVT)
*
        L      R9,@CVT        Get CVT address
        LA     R0,EXIT01QS     Point at QSABLOK describing the area
        RCALL  DMTQSAFA        Free all WORK14 areas in pool
        REXIT  RC=0,REGS=(0,12) Return to caller
*
12b EXIT01QS QSABLOK LENGTH=WORK14LN, Storage area length is WORK14LN      X
        INIT=WORK14LN,        Initialize this many bytes          X
        OPT4K=YES,            Get 4K pages and subdivide them      X
        EYECAT=WORK14,        Place 'WORK14 ' at top of page      X
        GETMAIN=RCTYPE,       Use conditional GETMAIN (no abend)   X
        PERSIST=YES           Get persistent storage
*
11b      WORK14 DSECT=NO        Template to initialize from
*
        CRV      DSECT=YES
        CVT      DSECT=YES
        END

```

Figure 13. Exit 1 Routine: Freeing Storage Acquired by the Exit 14 Routine

When the Exit 14 routine calls DMTQSAAB (12a), it points R0 to a template (12b) that describes the characteristics of the required storage. This information includes its length and the type of the GETMAIN macros and subpools used to acquire the storage. If the INIT parameter is omitted or the value specified on it is less than the number specified on the LENGTH parameter, any bytes that have not been initialized are set to zeros (this is similar to GETMAIN).

The PERSIST=YES parameter indicates that the GETMAIN macro should acquire all storage from persistent subpools (subpool 243). For more information, see *z/VM: Group Control System*. YES is specified in Figure 13 on page 27 because DMTQSAAB is called by Exit 14, which is reentrant and is called from a link driver task. If a link driver task obtains storage, GCS can free the storage when the link terminates, even though another link driver task may be referencing it. Exit routines for Exit 0 or Exit 1, which are called only from the RSCS communications task, can call DMTQSAAB to obtain storage from nonpersistent subpools.

**Note:** Because DMTQSA updates the QSABLOK defined by, and residing in, the exit module, you cannot specify the RENT option when link-editing the exit. If you do, an abend X'0C4' will result.

## Packaging Considerations

The term *exit package* describes a collection of exit routines, which meet a common requirement, that are part of one load module. The content of the exit routines and your RSCS environment determine how you structure your exit packages and the link-editing options you specify. Using the GCS GLOBAL statement, you can identify up to 63 load libraries to GCS; this number is sufficient for most RSCS exit packages.

To create a package of common exit routines, you can place all the entry points in one module. You can then place any DSECTs that are specific to the set of exit routines at the bottom of the module. For example, to create a security exit package that uses exit points 0, 1, 14, 15, 19, 21, and 32, you could use the source file shown in Figure 14 on page 28.

```

SECURE  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
:
SECURE00 RENTRY RENT=NO,SAVAREA=PREALLOC
:
SECURE01 RENTRY RENT=NO,SAVAREA=PREALLOC
:
SECURE14 RENTRY RENT=YES,SAVAREA=PREALLOC
:
SECURE15 RENTRY RENT=YES,SAVAREA=PREALLOC
:
SECURE19 RENTRY RENT=NO,SAVAREA=PREALLOC
:
SECURE21 RENTRY RENT=NO,SAVAREA=PREALLOC
:
SECURE32 RENTRY RENT=YES,SAVAREA=PREALLOC
:
* Security related DSECTs
:
      END

```

Figure 14. Packaging One Source Module (SECURE ASSEMBLE)

If an exit package issues messages other than those supplied by RSCS, it should also include a message file and a conversion repository. For example, the exit package in [Figure 14 on page 28](#) uses the message file SECENG MSGS and conversion repository SECMGC MCONV to issue messages. The exit package, containing the exit routines and message files, is created using the link-edit control file in [Figure 15 on page 28](#).

```

* SECURE Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY SECURE
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE SECURE
INCLUDE SECENG
INCLUDE SECMGC
ALIAS SECURE00,SECURE01,SECURE14,SECURE15,SECURE19,SECURE21,SECURE32
NAME SECURE

```

Figure 15. SECURE LKEDCTRL File

## Identifying Entry Points

Every entry point name that is used as an RSCS exit routine must be identified to GCS. ALIAS statements in the link-edit control file identify the entry points that are used as an alias for a load module. In [Figure 15 on page 28](#), the entry point names SECURE00 - SECURE32 are aliases for the SECURE load module. As [Figure 16 on page 28](#) shows, aliases can also be specified at the top of the SECURE ASSEMBLE file.

```

      PUNCH ' ALIAS SECURE00,SECURE01,SECURE14,SECURE15,SECURE19'
      PUNCH ' ALIAS SECURE21,SECURE32'
SECURE  RMOD  CRVBASE=TCRVTAB,CRVCALL=YES

```

Figure 16. Specifying Aliases in the Source Module (SECURE ASSEMBLE)

You can use either method to specify the aliases; however, you can specify a maximum of 16 alias names in a load module. To identify more than 16 entry points in an exit package, you can divide the exit package into 2 or more load modules. You can also use the LOADCMD command and GCS IDENTIFY macro to identify the entry points to GCS.

## Separating Load Modules

If you divide your exit package, you may need to restructure it; exit routines may not be able to locate an entry point in another load module that does not have an alias. To obtain the addresses of routines in other load modules, exit routines can use the RSCS user fields (see [“User Fields” on page 31](#)). You can then use a link-edit control file (see [Figure 17 on page 29](#)) to create a load library that contains 2 load

modules. In this example, the ACCNTN and ACCNTR routines can use the TUSER field to access any utility routines in ACCUTL.

```
* ACCNTR Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY ACCNTR
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE ACCNTR
ALIAS ACCNTR00,ACCNTR01,ACCNTR02,ACCNTR03,ACCNTR04,ACCNTR05
ALIAS ACCNTR06,ACCNTR07,ACCNTR08,ACCNTR09,ACCNTR10,ACCNTR11
ALIAS ACCNTR12,ACCNTR13,ACCNTR14,ACCNTR15
NAME ACCNTR
*
INCLUDE ACCNTN
INCLUDE ACCUTL
ALIAS ACCNTN16,ACCNTR17,ACCNTR18,ACCNTR19,ACCNTR20,ACCNTR21
ALIAS ACCNTR22,ACCNTR23,ACCNTR24,ACCNTR25,ACCNTR26,ACCNTR27
ALIAS ACCNTR28,ACCNTR29,ACCNTR30,ACCNTR31
NAME ACCNTN
```

Figure 17. Sample Link-Edit Control File: Creating Two Load Modules

## Using GCS Facilities

If you use the LOADCMD command and IDENTIFY macro, most of your exit routines can remain in the load module. To create a load module for the exit routines in [Figure 17 on page 29](#), you could use the link-edit control file in [Figure 18 on page 29](#).

```
* ACCNTR Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY ACCNTR
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE ACCNTI
INCLUDE ACCNTR
INCLUDE ACCNTN
INCLUDE ACCUTL
ENTRY ACCNTIEP
NAME ACCNTI
```

Figure 18. Link-edit Control File: Creating One Load Module

In [Figure 18 on page 29](#), no alias names for the load module have been defined. However, GCS is informed of the entry point names by the GCS IDENTIFY macro, which is issued from the exit routine **(13)**. One IDENTIFY macro must be run for each entry point that was previously specified on an ALIAS statement. For this example, module ACCNTI contains the statements in [Figure 19 on page 30](#).

```

ACCNТИ  RMOD
ACCNТИEP RENTRY RENT=NO,SAVAREA=PREALLOC
        LA R2,ENTRYPTS      Point at table
        LA R3,ENTRYNUM      Get number of entries
IDENLOOP EQU *
        L R1,8(0,R2)        Get address of entry point
13 IDENTIFY EPLOC=0(R2),ENTRY=(1) Identify entry point to GCS
        LTR R15,R15          Did it go OK?
        BNZ IDENTERR        No ... scream and shout
        LA R2,12(0,R2)      On to next entry
        BCT R3,IDENLOOP     Loop through table
        SR R15,R15          Set zero return code
        B EXIT              Call it a day

*
IDENTERR EQU *
        MVC EPNAME,0(R2)    Set offending name
        WTO MF=(E,ERROR)    Issue message to console
        LA R15,4            Set bad return code
EXIT EQU *
        REXIT RC=(R15),REGS=(0,12) Return to GCS

*
ERROR WTO 'Bad rc found for entry point xxxxxxxx',MF=L
EPNAME EQU ERROR+4+35,8    Place to plug in epname
*
DS 0F
ENTRYPTS DC CL8'ACCNTR00',V(ACCNTR00),CL8'ACCNTR01',V(ACCNTR01)
        DC CL8'ACCNTR02',V(ACCNTR02),CL8'ACCNTR03',V(ACCNTR03)
        DC CL8'ACCNTR04',V(ACCNTR04),CL8'ACCNTR05',V(ACCNTR05)
        DC CL8'ACCNTR06',V(ACCNTR06),CL8'ACCNTR07',V(ACCNTR07)
        DC CL8'ACCNTR08',V(ACCNTR08),CL8'ACCNTR09',V(ACCNTR09)
        DC CL8'ACCNTR10',V(ACCNTR10),CL8'ACCNTR11',V(ACCNTR11)
        DC CL8'ACCNTR12',V(ACCNTR12),CL8'ACCNTR13',V(ACCNTR13)
        DC CL8'ACCNTR14',V(ACCNTR14),CL8'ACCNTR15',V(ACCNTR15)
*
        DC CL8'ACCNTN16',V(ACCNTR16),CL8'ACCNTN17',V(ACCNTR17)
        DC CL8'ACCNTN18',V(ACCNTR18),CL8'ACCNTN19',V(ACCNTR19)
        DC CL8'ACCNTN20',V(ACCNTR20),CL8'ACCNTN21',V(ACCNTR21)
        DC CL8'ACCNTN22',V(ACCNTR22),CL8'ACCNTN23',V(ACCNTR23)
        DC CL8'ACCNTN24',V(ACCNTR24),CL8'ACCNTN25',V(ACCNTR25)
        DC CL8'ACCNTN26',V(ACCNTR26),CL8'ACCNTN27',V(ACCNTR27)
        DC CL8'ACCNTN28',V(ACCNTR28),CL8'ACCNTN29',V(ACCNTR29)
        DC CL8'ACCNTN30',V(ACCNTR30),CL8'ACCNTN31',V(ACCNTR31)
ENTRYNUM EQU (*-ENTRYPTS)/12    Number of entries
END

```

Figure 19. Identifying Entry Points (ACCNТИ ASSEMBLE)

When using this approach, you must ensure that the code at entry point ACCNТИEP is processed *before* RSCS tries to access any of the entry points previously given aliases. To do so, you would add the commands shown in Figure 20 on page 30 to the PROFILE GCS file of the RSCS virtual machine.

```

'global loadlib rscs accntr' /* make load libraries known to GCS */
:
'loadcmd rscs dmtman' /* load RSCS load module */
'loadcmd accnti accnti' /* load ACCNТИ load module */
:
'accnti' /* call ACCNТИEP routine */
if rc <> 0 then do /* problems IDENTIFYing entry points? */
    say 'Bad return code 'rc' from ACCNТИEP'
    exit 4
end
'rscs init' /* start RSCS with INIT command */

```

Figure 20. Identifying Entry Points in the PROFILE GCS

At times, you may want to include several unrelated exit packages, as separate load modules, in one load library. To combine exit packages that do not need to exchange information, you could use a link-edit control similar to Figure 17 on page 29. Here, you do not have to create a load library for each exit package or specify each load library name on the GCS GLOBAL command in the PROFILE GCS file.

## Sharing Information

**Note:** Many IBM-supplied exit packages use several of the facilities described in this section.

Several exit routines may be needed to customize RSCS processing at your installation. Sometimes, these exit routines must exchange information to achieve the desired results. This section describes how several exit routines can communicate with each other.

You can create exit packages to provide standard routines that are often used by other exit routines. To do so, you could create the required routines and link-edit them into one load module. You can then place this load module in a load library and identify each routine within it by an ALIAS statement. Other exit routines or exit packages can then issue GCS LOAD macros to find the necessary routines in the load module.

To increase performance, however, you can use Exit 0 to locate the addresses of exit routines within the exit package. For this case, the Exit 0 routine creates a vector of the exit routine addresses (similar to the CRV). It also creates a mapping DSECT (in macro form) that refers to entries in the vector. The address of this vector should then be made available to all other exit packages.

## User Fields

Some commonly used RSCS control blocks contain special 8-byte fields called *user fields*. User fields let exit routines that reside in separate load modules pass information to each other. Exit routines can use these fields to establish additional exit-related fields or to point to additional work areas. RSCS provides user fields in the following control blocks:

User Field	Control Block
LUSER	Link table (LINKTABL)
NOTEUSER	NOTIFY link driver control block (NOTEBLOK)
MSGBUSER	Message request parameter list (MSGBLOK)
RIBUSER	Receiver information block (RIB)
SEPUSER	Separator page control block (SEPBLOK)
TAGFLAGU	TAG element
TAGUSER	TAG element
TIBUSER	Transmitter information block (TIB)
TUSER	Communications vector table (CVT)

Because the use of user fields may cause conflict with other exit packages, these fields should be used only when necessary. You should not use the user fields to pass information between exit routines that are in the same load module. If you use an RSCS user field in your exit package, another exit routine may also need to use the same field. Use the following criteria to reduce conflicts if exit packages share the user fields:

- Use the second 4 bytes of any user field as a pointer to the data area your exit package wants to share with other exit packages.
- Use the following format for the first 16 bytes of any shared data areas:

Displacement	Length	Contents
0	4	Pointer to next shared exit data area
4	8	Identifies the purpose of the data area
12	4	Version number (if applicable) for the data area

This format allows other exit packages that may depend on your exit package to search this chain of data areas. The exit routines can use the identifier and version number to locate specific data areas.

For example, to do the sample exit packages previously discussed, you could create the UTILCRV macro (see [Figure 21 on page 32](#)).

```

MACRO
&NAME    UTILCRV &DSECT=YES
        LCLC  &LABEL
&LABEL   SETC  'UTILCRV'           Default name if none specified
        AIF   (T'&NAME EQ '0').NONAME
&LABEL   SETC  '&NAME'             Use the specified name
.NONAME   ANOP
&LABEL   DS    0D
        AGO   .GENCODE
.DSECT1   ANOP
&LABEL   DSECT
.GENCODE   ANOP
UTLNEXT   DC    A(0)               Pointer to next data area
UTLID     DC    CL8'EXUTILS '      Identifier for this data area
UTLLEVEL  DC    CL4'1.01'          Level of routines
UTLRAUTH  DC    V(UTLAUTBL)        Routine to build AUTH table
UTLRAUTH  DC    V(UTLAUTCK)        Routine to check authority
:
UTLCRVLN  EQU   *-&LABEL           Length of data area
        MEND

```

Figure 21. Example Macro to Map Exit Utility Routine Addresses

This macro can then be used in the Exit 0 routine in [Figure 22 on page 32](#).

```

UTIL00    RMOD  CRVBASE=TCRVTAB,CRVCALL=YES
*
        USING CVT,R9              Get CVT addressability
UTIL00EP  RENTRY RENT=YES,SAVAREA=PREALLOC,ARGS=(@CVT)
*
        L      R9,@CVT            Get CVT pointer from plist
        MVC    UTLNEXT,TUSER+4    Chain any existing ones on us
        LA     R1,UTILCRV         Point at our package CRV
        ST     R1,TUSER+4         Anchor our data area in CVT
        REXIT  RC=0,REGS=(0,12)   Return to caller
*
        UTILCRV DSECT=NO          Generate copy of UTILCRV
*
        CRV    DSECT=YES
        CVT    DSECT=YES
        END

```

Figure 22. Exit 0 Routine: Installing a Utility Routine Package

If an exit routine in another exit package needs to use a routine in the utility routines package, it can get the address of the routine from the CVT. For example, the exit routine can use the code in [Figure 23 on page 32](#).

```

:
LOOKLOOP  LA     R2,TUSER+4        Point at anchor for data areas
        EQU     *
        ICM     R2,B'1111',0(R2)  Get next data area
        BZ      PACKMISS          Not found ... pre-req package missing
        CLC     4(8,R2),=CL8'EXUTILS '
*
        BNE     LOOKLOOP          Is this the data area we want?
        CLC     12(4,R2),=CL4'1.01' Is it a late enough level?
        BL      BADLEVEL          No ... can't use this level
        USING  UTILCRV,R2         Map the UTILCRV
        LA     R1,BLPLIST         Point at build plist
        L      R15,UTLAUTBL       Get address of build routine
        RCALL  (R15)              Call routine
:
        UTILCRV DSECT=YES          Generate copy of UTILCRV

```

Figure 23. Finding and Calling a Utility Routine

## Link-Editing Considerations

After packaging exit routines, you must choose the characteristics of each load module. These characteristics are defined by the option specified on the %LEPARMS statement in the LKEDCTRL control file. There are three categories of load modules:

Category	Option	Description
Reentrant	RENT	Read-only load module; multiple LOAD and ATTACH macros issued for these entry points cause only one copy to be loaded into storage.
Reusable	REUS	Read-write load module; each LOAD macro processed for an entry point in the module results in the use of the entry point in the single copy of the load module that is loaded into storage.  The first ATTACH macro issued for an entry point will be successful; later ATTACH macros must wait until the first attached task terminates (only one active task can be attached at a time).
Nonreusable	Neither	Read-write load module; each LOAD and ATTACH macro processed against an entry point in the load module loads a new copy of the load module into storage.

To combine several load modules in one load library, you can use several %LEPARMS statements to specify the link-editing option for each load module. The load modules in a load library do not necessarily need to be in the same load module category.

Load modules that contain exit routines for use with exit points in the RSCS exit facility should be placed in a reusable load module. Even if the exit has been written to be reentrant, it should still be placed in a reusable module. Loadable link drivers, defined by the LINKTYPE statement, should be placed in reentrant load modules, if the link driver routine is written in a read-only manner. If the link driver has been written to be nonreentrant, it should be placed in a nonreusable load module. Similarly, ASCII, LPR, LPD, UFT, and UFTD exit routines and gateway programs should also be placed in reentrant or nonreusable load modules, depending on how the exit routine or gateway program is written. Transmission algorithms should be placed in a serially reusable load module.

**Note:** The characteristics of each load module are not related to the parameters you specify on the RENTRY macro. For example, if you specify RENT=YES on the RENTRY macro, it does not effect the link-edit process. For more information, see [“RENTRY – Defining a Module Entry Point” on page 290](#).

## Common Problems and Solutions

If you specify the wrong option for a load module, problems may occur. The following list describes some common problems and corrections:

- A protection exception occurs when your exit routine attempts to store data in a field inside the load module.

The load module has been loaded in a read-only manner; this implies that the RENT option was specified. If your exit routine is reusable or nonreusable, correct the option and rebuild your load library.

- Exit loading takes an unusually long amount of time.

You may have created a nonreusable module instead of a reusable module. Note the addresses at which RSCS loads the exit routines and compare the difference in the addresses to the size of your load module. If the location of the entry points in the load module are not in the same relation as they are in the source module, the entry points are in separate copies of the load module. GCS may be loading a new copy of the load module for each entry point; this can cause performance problems when RSCS initializes.

- Information set in the load module by an exit routine, such as Exit 0, is not initialized when referenced from another exit routine.

You may have created a nonreusable module instead of a reusable module. The exit routine may have set information in its copy of the load module. However, this information is unavailable to the other exit routines because they are running in separate copies of the load module.

Note the addresses at which RSCS loads the various exit routines and compare the difference to the size of your load module. The address of the entry points should relate to the location of the entry points in the source module. If not, the entry points are in separate copies of the load module.

- When you start a second instance of a link defined by the LINKTYPE statement, the link does not start until the first link driver has stopped or drained.

You may have defined the load module as reusable; it should have been defined as reentrant or nonreusable.

- Message DMT430E (exit routine is not loadable) is issued when you try to load an exit module.

The exit module may not be enabled for 31-bit addressing. You may also receive this message if the exit module does not exist in any member specified on the GLOBAL LOADLIB command.

Ensure the AMODE 31 RMODE ANY options have been added to the %LEPARMS statement in the LKEDCTRL file or on the :OPTIONS. statement of a VMSES/E build list. Also, ensure that the exit module name is specified correctly on the GLOBAL command.

- Incorrect usage of the TUSER field of the CVT could cause unpredictable results, including abends, when multiple exit packages utilize it.

## Distribution Considerations

When distributing exit packages that use IBM-defined exit points or define new types of link drivers, you should also include the necessary EXIT or LINKTYPE configuration statements in a separate file. The installation that receives the exit package can then use the RSCS IMBED facility to install the package, without adding each statement in its configuration file.

For example, the installation PITTSBGH uses exit packages from several sources. It could then use the RSCS configuration file in [Figure 24 on page 34](#) to install the exit routines.

```
LOCAL PITTSBGH
/*
/* Install Virtual Printer (Loadable Link Driver)      */
/*
IMBED VPRLINK CONFIG
/*
/* Get links, routes, authorizations, miscellaneous items */
/*
IMBED LINKS CONFIG
IMBED ROUTES CONFIG
IMBED AUTHS CONFIG
IMBED MISC CONFIG
/*
/* Install Security Package                             */
/*
IMBED SECURE CONFIG
/*
/* Install Accounting Package                           */
/*
IMBED ACCNTR CONFIG
```

*Figure 24. Sample RSCS CONFIG File*

The VPRLINK, SECURE, and ACCNTR CONFIG files should then contain the statements necessary to install the exit packages. The SECURE CONFIG file, for example, contains the statements in [Figure 25 on page 35](#).



```

/*
/* Statements required to install SECURE package */
/*
EXIT 00 ON      SECURE00
EXIT 01 ON      SECURE01
EXIT 14 ON FIRST SECURE14
EXIT 15 ON FIRST SECURE15
EXIT 19 ON FIRST SECURE19
EXIT 21 ON FIRST SECURE21
EXIT 32 ON FIRST SECURE32

```

Figure 25. SECURE CONFIG File for Security Exit Package

## Specifying the Order of the Exit Routines

Each exit point defined by an EXIT statement is associated with a chain of exit routines. The FIRST parameter of the EXIT statement identifies the exit routine that is placed at the head of this chain. RSCS places the last exit routine that was defined with the FIRST parameter at the head of this chain. For example in [Figure 25 on page 35](#), the exit routine SECURE32 is placed at the head of the exit routine chain.

For most exit packages, you should specify the FIRST parameter on the EXIT statement that defines Exit 1. This ensures that Exit 1 routines can perform or complete any processing before RSCS terminates. Most other exit routines do not have to appear in a certain order. Generally, the configuration file statements for an exit package should have the format shown in [Figure 26 on page 35](#).

```

EXIT 0 ON      SAMP00EP
EXIT 1 ON FIRST SAMP01EP
EXIT 2 ON      SAMP02EP
.
EXIT 40 ON     SAMP40EP

```

Figure 26. Sample Configuration File Statements

However, the order that you specify exit routines may be especially important for exit packages for security and accounting functions. When installing two or more exit packages, you must ensure that their function does not rely on being the first exit routine in the chain. You must determine the order that the entry points in the exit routines are installed. You must also determine if the exit packages have conflicting functions or requirements; RSCS does not perform these checks.

For example, assume the security and accounting exit packages in [Figure 24 on page 34](#) each have the requirement to be the first exit routine called for Exit 32. The order that the packages are installed can affect the information recorded by the accounting exit routines on the PITTSBGH node. For example, if the security exit routines are installed first, any files they reject cannot be processed by the accounting exit routines.

## Tracing Exit Routines

The ITRACE macro records calls to, and the return from, exit routines and transmission algorithms 0 and 1.

You can also use the RSCS ITRACE facility within exit routines to record information that can be used for debugging. IBM has reserved predefined event-type categories for customer use only. You can use these categories to record various data areas in the RSCS internal trace table. By specifying the GTRACE option on the ITRACE command and statement, trace information can also be recorded in the RSCS virtual machine's GCS trace table.

For more information about the ITRACE macro, see [“ITRACE – Tracing an Event” on page 274](#). For more information about the ITRACE statement, see [z/VM: RSCS Networking Planning and Configuration](#). For more information about the ITRACE command, see [z/VM: RSCS Networking Operation and Use](#).

## Using Sample Exit Packages

RSCS provides sample exit packages that demonstrate how you can use the RSCS exit facilities and control files at your installation. The sample packages contain the files required to install and use the sample routines. These files include: overview information, assembler and macro files, message repositories, and sample configuration file statements. The sample exit routine packages are built into the RSCSEXIT LOADLIB file.

These samples are provided for illustrative purposes and are supplied on an *as is* basis. You may be able to use these sample exit routines with little or no modifications, depending on the needs and configuration of your installation.

## Enabling Sample Exit Routines

The RSCS sample exit routines are installed and serviced using the VMSES/E component of z/VM. These samples are also 31-bit enabled and are designed to run in ESA mode. The RSCSEXIT LOADLIB and DMTMACEX MACLIB files, which are supplied with the RSCS installation tapes, contain the exit routines and macros, respectively, for the sample packages. The control file supplied with the sample exit routines is DMTVMEX CNTRL.

To enable these sample exit routines as they are supplied, perform the following steps:

Step	Action
1	Ensure the ON parameter is specified on the EXIT statement included in the configuration file supplied with the sample package. (To disable the exit routine, you can specify the OFF parameter.)
2	Include the sample configuration file in the RSCS CONFIG file; to do so, use one of the following methods: <ul style="list-style-type: none"> <li>Specify the sample configuration file on an IMBED statement in the RSCS CONFIG file.</li> <li>Specify the sample configuration file on a FILEDEF statement in the PROFILE GCS file for the RSCS virtual machine.</li> <li>Add the information from the sample configuration file directly into the RSCS CONFIG file.</li> </ul>
3	Add the RSCSEXIT LOADLIB to the GLOBAL statement in the PROFILE GCS file for the RSCS virtual machine.

For more information about using or modifying the sample exit packages, see the RSCS program directory.

## Summary of Sample Packages

Table 3 on page 37 describes the sample packages and the type of exit facility that is demonstrated. For more information about these samples, including file names and installation locations, see the RSCS program directory.

For information about other sample exit routines supplied with RSCS, see:

- [“Sample ASCII Printer and Plotter Exit Modules” on page 156](#)
- [“Sample LPR Exit Routines” on page 200](#)
- [“Sample LPD Exit Routine” on page 224](#)
- [“Sample UFT Exit Routine” on page 240](#)
- [“Sample UFTD Exit Routine” on page 257](#)

Table 3. Sample Exit Routine Packages

Package Name	Function	Exit Facility
Spool manager command echoing (SAC)	Views the CP commands issued by the RSCS spool manager task.	Exit 24, Exit 25, Exit 29
Back-to-back RSCS configuration (SBK)	Create multiple RSCS virtual machines on one node.	Exit 0, Exit 21, Exit 24
Secondary RSCS list processor (SBURST)	Creates two RSCS virtual machines to handle processing on LISTPROC-type links.	Exit 0, Exit 1, Exit 19, Exit 21, Exit 24
Selective file filter (SFF)	Purge undesired files from the network or route them to a security machine.	Exit 0, Exit 1, Exit 15, Exit 21, Exit 29
Shift-based file limiting (SFL)	Prevents large files from being transmitted during a specific time. This package, which is shown in <a href="#">“Example 1: Defining Printing Shifts”</a> on page 18, also uses the RSCS event scheduler.	Exit 0, Exit 1, Exit 31, Exit 33
Note selection and modification (SNM)	Modify the note produced by the NOTIFY link driver.	Exit 22, Exit 23
NOTIFY link driver purge (SNS)	Prevent class H files from being purged from a NOTIFY-type link.	Exit 36
Path Alias map processing and PAPATH command (SPA)	View the RSCS routing network as defined on your local node and the routing defined on each other RSCS node in the network.	Exit 0, Exit 29
File queue aging (SQA)	Determine the order by which files are sent on a link; the selection criteria can be determined by the size of the file or the time it has been waiting on the link queue.	Exit 0, Exit 1, Exit 3, Exit 26, Exit 31, Exit 33, Exit 34
REMOVE command (SRMVEX)	Creates a new command that enables the RSCS operator to transfer files away from the RSCS virtual machine.	Exit 21, Exit 29
SHOW and PATH command (SSH)	Creates new commands to display information about the RSCS network. SHOW displays files queued for a specific node; PATH displays the path to a node.	Exit 29
Sample TYPE command (STY)	Creates a CMS-like TYPE command for RSCS authorized operators; this routine is shown in <a href="#">“Example 3: Creating a New Command”</a> on page 22.	Exit 29
Separator pages (SSP)	Modify the printer separator pages produced by RSCS; functions let you create an 80-column output per line or highlight variable fields.	Exit 17, Exit 18
Sample GPI link driver (GPSAMP)	Enable two RSCS virtual machines on the same processor to communicate through IUCV. This sample also enables RSCS virtual machines on different processors to communicate using an IUCV path through a VM/Pass-Through Facility network.	Gateway programming interface
Gateway security modifications (GSM)	Enables installations to control the data traffic through the RSCS virtual machine.	Gateway programming interface, Exit 0, Exit 1, Exit 14, Exit 15, Exit 19, Exit 21, Exit 29, Exit 32

Table 3. Sample Exit Routine Packages (continued)

Package Name	Function	Exit Facility
Set Greenwich Mean Time offset (SSI)	Correct the Greenwich Mean Time (GMT) offset value for the RSCS virtual machine; this package is required on many systems to ensure that the RSCS event scheduler works properly.	Exit 0
Simple security package (SSS)	Limit file traffic on a specific link to a specific user IDs or limit the use of RSCS on the local node to specific users.	Exit 0, Exit 1, Exit 14, Exit 15, Exit 19, Exit 21, Exit 32
MESSAGER link driver (SMS)	Writes messages, accounting information, and statistics into a file and sends the file to a specified user ID or output device.	Loadable link driver, Exit 0, Exit 27
Console logging and screening (SMG)	Suppress or send RSCS console messages to a MESSAGER-type link.	Loadable link driver, Exit 0, Exit 27
Simple accounting package (SAS)	Create accounting records and send them on a MESSAGER-type link to be printed.	Loadable link driver, Exit 2, Exit 3, Exit 4, Exit 5
Statistics-gathering (SST)	Creates a 120-byte record for each file sent by a link and send the record to a MESSAGER-type link. Each record contains file statistics, such as: origin, destination, size, and transmission times.	Loadable link driver, Exit 0, Exit 3, Exit 21
Host transfer agent link driver (STR)	Transfer files to a specific server machine, while retaining the RSCS store-and-forward tag.	Loadable link driver
Virtual printer link driver (SVP)	Send files to a specific server machine to perform special processing.	Loadable link driver

---

## Chapter 3. IBM-Defined Exit Points

This section describes each of the IBM-defined exit points in the RSCS exit facility. For general information about writing exit routines, see [Chapter 2, “Customizing RSCS,” on page 9](#).

### Usage Conventions

---

This section describes the standard conventions used to pass control to and from the IBM-defined exit points. Your exit routines should follow these conventions.

#### Standard Entry Conditions

When RSCS passes control to the *first* exit routine associated with an exit point, registers 2 - 15 contain the following information. The parameters passed in R0 and R1 can vary for each exit point. Their contents are described for each individual exit point.

Register	Contents
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

If more than one exit routine is associated with an exit point, the register contents passed *to* those routines are determined by the register contents passed *back* by the *preceding* exit routine.

#### Standard Exit Conditions

When most exit routines return control to RSCS, the registers contain the following information.

Register	Contents
R0 - R1	Not applicable; however, if an exit routine issues return code 0, it should specify REGS=(0,12) on the REXIT macro to restore the contents of these registers.
R2 - R13	Restored to same values as on entry
R14	Not applicable
R15	Always contains a return code

#### Standard Return Codes

Each exit routine issues a return code (a multiple of 4) in R15 when it returns control to RSCS. This return code tells RSCS how it should continue to process the task from which the exit routine was called. All exit routines issue return codes 0 and 4, which tell RSCS to continue its processing as if the exit point was not enabled.

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing the task.
4	RSCS does not call any other exit routines associated with this exit point and continues usual task processing.

Exit routines may also issue other return codes (8 or greater) that tell RSCS to modify its processing of the task. These return codes, if applicable, are described for each exit point.

## Data Areas

All exit routines are passed the address of the CVT in Word 1 of the R1 parameter list. The CVT contains pointers to some RSCS data areas, counters, and flags that are available for exit routine use. The CVT also contains a pointer to the CRV, which contains the addresses of some RSCS routines that are available for exit routine use.

RSCS also passes the addresses of other data areas to each exit point. These data areas can contain information about the file, command, or message being processed when the exit routine is called.

For a list of RSCS data areas supported as programming interfaces, see [Appendix A, "DSECTs Generated by Mapping Macros,"](#) on page 367. RSCS does not pass the address of all of these data areas to each exit point. The "Entry Conditions" section of each exit point description identifies the applicable data areas. Also, see the programming considerations section of the description of each exit point for information about any restrictions.

## Accounting Records

If the z/VM user directory entry for the RSCS virtual machine includes an OPTION ACCT statement (see [z/VM: RSCS Networking Planning and Configuration](#)), RSCS can issue DIAGNOSE code X'4C' to create accounting records for each file received or transmitted and pass them to CP to include in the accounting log. For more information about the OPTION ACCT statement, see [z/VM: CP Planning and Administration](#). For more information about DIAGNOSE code X'4C', see [z/VM: CP Programming Services](#).

You can use the following exits to accumulate accounting information or to generate or process RSCS accounting records:

- Exit 2 – Spool File Accept Accounting
- Exit 3 – Spool File Send Accounting
- Exit 4 – Spool File Purge Accounting
- Exit 5 – Spool File Receive Accounting
- Exit 7 – Auto-Answer Sign-On Time Out
- Exit 8 – Auto-Answer Unrecognizable Data
- Exit 9 – Auto-Answer Sign-On Validation
- Exit 10 – Auto-Answer Sign-On Reject
- Exit 21 – Spool File Accept/Reject
- Exit 26 – Link State Change Accounting
- Exit 44 – Link Termination
- Exit 45 – Output Page Accounting
- Exit 46 – Verification of Page Accounting
- Exit 47 – Driver Initialization
- Exit 48 – Verification of Output Page Error

The ACNTBUFF macro maps the ACNTBUFF data area, which contains the format of the standard RSCS accounting record.

## Exit 0 – Initialization

Use Exit 0 to perform additional processing during RSCS initialization. For example, you can use Exit 0 to prepare calls to other exit packages, obtain working storage, read information from another file, or open virtual devices.

If an exit package fails to start and its function is vital to your installation, your Exit 0 routine should issue return code 8. This prevents the RSCS virtual machine from initializing; you can then correct any problems.

### Point of Processing

Process	Exit Attribute
RSCS initialization	Serially reusable

Exit 0 is called after RSCS processes the configuration file. It is the first exit point called as RSCS initializes; no other exit routines have been installed.

If RSCS finds an error as it processes the configuration file, Exit 0 is not called. If the configuration file is successfully opened and read, your Exit 0 routine can use the CVT to access information about the initial RSCS configuration. At this point, however, no system tasks (for example, spool manager or exec tasks) have been started and no files are queued on any links.

On return from Exit 0, if your exit routine has not specified return code 8, RSCS attaches all system tasks to complete its initialization; it then returns control to GCS. When START commands are issued, RSCS attaches the appropriate link driver tasks.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT

### Exit Conditions

On return, Exit 0 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues initialization processing.
4	RSCS does not call any other exit routines associated with this exit point and continues initialization processing.
8	An Exit 0 routine or exit package did not start correctly. Any other exit routines associated with Exit 0 will be called; however, on return, RSCS will not start.

## Programming Considerations

- Any storage obtained at Exit 0 is kept until RSCS terminates or until the storage is explicitly released by a FREEMAIN macro.
- Use Exit 1 to ensure that all processing done by exit routines associated with this exit has been completed.
- If necessary, your exit routine can use RSCS user fields to anchor any working storage that must be accessible to other exit packages. Your exit routine can use the TUSER field in the CVT. For more information, see [“Sharing Information”](#) on page 30.
- During initialization, RSCS calls the exit routines associated with Exit 0 and checks their return codes. If none of the Exit 0 routines issues return code 8, RSCS will install the exit routines for all other defined exit points. If an Exit 0 routine issues return code 8, RSCS will call any remaining exit routines associated with Exit 0. It will then call Exit 1 to perform termination processing. However, RSCS will not install or call any other exit routines. The RSCS virtual machine will not start.

For example, assume that three exit routines are associated with Exit 0. The first routine, EXIT0A, issues return code 0. RSCS then calls the EXIT0B and EXIT0C routines. If these Exit 0 routines do not issue return code 8, RSCS will then install the exit routines for all defined exit points. If, however, EXIT0B issues return code 8, RSCS calls EXIT0C and any Exit 1 routines. RSCS does not install any other exit routines and does not continue to start.

- Because RSCS has not yet installed other exit routines when it calls Exit 0, your Exit 0 routines should not rely on the functions of other exit points.

For example, if the EXIT0A routine issues a message, it cannot be logged by any Exit 27 routines. After RSCS installs all other exit routines, however, Exit 27 can process other messages.

- To define additional event types for the internal trace table, your Exit 0 routine can specify the ITFORMAT and INSTALIT macros. Other exit routines can then issue ITRACE macros to trace the specified event. For more information, see [“INSTALIT – Adding a Record Format Table”](#) on page 272 and [“ITRACE – Tracing an Event”](#) on page 274.



## Exit 1 – Termination

Use Exit 1 to complete any processing started at Exit 0 and process information before RSCS terminates. For example, you can use an Exit 1 routine to write data, which was obtained and stored by other exit routines, into a CMS file.

### Point of Processing

Process	Exit Attribute
RSCS termination	Serially reusable

Exit 1 is called when RSCS processes a SHUTDOWN command. It is the last exit point called before ending completes. All exit routines associated with other exit points are no longer installed; all RSCS tasks have ended. On return from Exit 1, RSCS ends and returns control to GCS.

If an error occurs as the configuration file is read and Exit 0 is not called, Exit 1 is not called when RSCS terminates. However, Exit 1 is called if an Exit 0 routine issues return code 8.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about the other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT

### Exit Conditions

On return, Exit 1 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues to end processing.
4	RSCS ignores any other exit routines associated with this exit point and continues to end processing.

### Programming Considerations

- Exit 1 routines do not need to release any storage acquired from nonpersistent subpools. However, Exit 1 routines must release any storage acquired from a persistent subpool, including storage associated with a user-defined QSABLOK that has specified PERSIST=YES.
- When Exit 1 is called, all exit routines for any other defined exit points are no longer available. The exit routines associated with Exit 1 are the last to be called before RSCS terminates. Your Exit 1 routines

## **Exit 1**

should not rely on any functions of other exit points. For example, if your Exit 1 routine issues messages, they will not appear in any message logs created at Exit 27.

## Exit 2 – Spool File Accept Accounting

Use Exit 2 to determine if RSCS should accept, reject, or create an accounting record for an input spool file. Your exit routine can define the criteria for accepting the file (for example, maximum record count or authorization of users).

Exit 2 is called each time RSCS finds a new file in its virtual reader, unless a REORDER command is being processed. Here, Exit 2 may not process some incoming files. To perform security functions, especially to force the acceptance of a file, you should use Exit 21 (see [“Exit 21 – Spool File Accept/Reject”](#) on page 83).

### Point of Processing

Process	Exit Attribute
Spool file reception	Serially reusable

Exit 2 is called each time RSCS processes a new input file. The file may be store-and-forward or may have originated at the local node. At this point, RSCS knows the file's origin and destination from the information in the file's CP TAG. RSCS has also determined if second-level addressing and rerouting are needed.

On return from Exit 2, if your exit routine issues return codes 0 or 4, RSCS accepts files that originated at the local node. RSCS also accepts store-and-forward files if they originated from the RSCS virtual machine and have not been transferred.

RSCS then sends the files to their destination. Files for local users are transferred to the specified user ID. Files for remote users are queued on all available links to the specified destination node.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions”](#) on page 39.

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the file's TAG element</li> <li><b>Word 3 (+8)</b> Address of a 70-byte accounting record work area</li> <li><b>Word 4 (+12)</b> Address of a halfword field containing the length of the file's CP TAG text</li> <li><b>Word 5 (+16)</b> Address of the CP TAG text</li> <li><b>Word 6 (+20)</b> Address of the file's CP SFBLOK</li> </ul>

Register	Contents
----------	----------

**Note:** If the file originated at the local node, word 5 of R1 contains the address of the CP TAG text, as entered by the file originator. If the file originated at a remote node, word 5 contains the address of the store-and-forward TAG text.

## Exit Conditions

On return, Exit 2 sets the standard register contents (see [“Standard Exit Conditions”](#) on page 39).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and accepts or rejects the file.
4	RSCS does not call any other exit routines associated with this exit point and accepts or rejects the file.
8	RSCS rejects the file. If the file originated at the local node, it is transferred to the originating user ID. If the file originated at a remote node, RSCS purges it and sends message DMT112E to the file originator.
12	RSCS accepts the file as if it has passed all standard RSCS authorization checks.
16	RSCS rejects the file. If the file originated at the local node, RSCS transfers it to the originating user ID. If it originated at a remote node, RSCS purges the file but does not issue a message to the originator.

## Programming Considerations

- Your exit routine should not change any fields in the TAG element for files that have NJE headers. Because RSCS has already determined the file's destination, changes to the TAG may not be reflected in the headers before the file is transmitted.

Store-and-forward files originating from workstation links may not have NJE headers; they will contain a blank TAGCNTRL field. Your exit routine can modify the TAG data for these files. Any changes are reflected in the NJE headers created to send the file to the next node.

- Use return code 12 to accept store-and-forward files created by user IDs other than RSCS and store-and-forward files whose transfer bit has been set.
- To create an accounting record at this exit point, your exit routine can issue return code 0 and use the ACNTBUFF for the basic format of the accounting record and then use CP DIAGNOSE code X'4C' or another facility to create the record.

**Note:** Because Exit 2 is not called when a REORDER command is processed, some files may not be included in the accounting record.

- For information about the CP SFBLOK and DIAGNOSE code X'4C', see [z/VM: CP Programming Services](#).
- If a file's destination is unknown, your exit routine can queue the file onto a NOTIFY-type link. To do so, the exit routine should place the name of the NOTIFY-type link in the TAGORLOC field of the file's TAG element.

## Exit 3 – Spool File Send Accounting

Use Exit 3 to modify or suppress an accounting record for each file RSCS sends. On networking links, Exit 3 is called once for each file on the link. For printer and workstation links, Exit 3 is called once for each copy of a multiple copy file.

The ACNTBUFF macro contains the format of the standard accounting record. If modifying the standard accounting record, your exit routine must replace the entire standard accounting record. It must then produce any record fields that are to be retained from the standard record format and any new or modified fields.

### Point of Processing

Process	Exit Attribute
Spool file accounting	Serially reusable

Exit 3 is called as RSCS starts to close an input spool file. On return, if the exit routine has not issued return code 8, RSCS issues CP DIAGNOSE code X'4C' to generate an accounting record. (The ACCT parameter must be specified on the OPTION statement in the RSCS virtual machine's z/VM directory entry to enable RSCS to create accounting records. For more information, see [z/VM: CP Planning and Administration](#).)

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the TAG element <b>Word 3 (+8)</b> Address of a 70-byte accounting record work area

### Exit Conditions

On return, Exit 3 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and creates a standard SEND accounting record.
4	RSCS does not call any other exit routines associated with this exit and creates a standard SEND accounting record.

<b>Return Code</b>	<b>Results</b>
8	RSCS does not create the standard accounting record nor call any other exit routine associated with this exit point.
12	RSCS does not create the standard accounting record and calls the next exit routine associated with Exit 3.

### **Programming Considerations**

- Your exit routine should not change any fields in the file's TAG element.
- To generate an accounting record, you can use the ACNTBUFF macro for the basic format and use DIAGNOSE code X'4C' to create a record. For more information about this facility, see [z/VM: CP Programming Services](#). You can also write the accounting information to another file.

## Exit 4 – Spool File Purge Accounting

Use Exit 4 to create an accounting record each time RSCS purges a spool file. As supplied, RSCS does not create an accounting record when files are purged from the network.

### Point of Processing

Process	Exit Attribute
Spool file processing	Serially reusable

Exit 4 is called each time a PURGE command is issued to delete a file. On return from the exit routine, RSCS purges the file.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the TAG <b>Word 3 (+8)</b> Address of a 70-byte accounting record work area

### Exit Conditions

On return, Exit 4 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and purges the file.
4	RSCS does not call any other exit routines associated with this exit point and purges the file.

### Programming Considerations

- Your exit routine should not alter any fields in the file's TAG element.
- Your exit routine can use the ACNTBUFF macro as the basis of the accounting record. Use CP DIAGNOSE code X'4C' to create the record. For more information, see [z/VM: CP Programming Services](#). Your exit routine can also write the accounting information to a CMS file.

## Exit 5 – Spool File Receive Accounting

Use Exit 5 to modify or suppress an accounting record for each spool file that RSCS receives on a link.

Exit 5 cannot directly modify the standard accounting record, which is described by the ACNTBUFF macro. When creating a new accounting record format, your exit routine must supply any new or changed fields; it also must retain any needed fields from the standard accounting record.

### Point of Processing

Process	Exit Attribute
Spool file processing	Serially reusable

Exit 5 is called each time RSCS receives a file on a networking or workstation link. At this point, RSCS knows the file's attributes, including its origin, destination, and size.

On return, RSCS issues CP DIAGNOSE code X'4C' to create an accounting record, which is mapped by the ACNTBUFF macro. RSCS then determines the validity of the file's final destination, including second-level addressing, and closes the file.

If the file's destination is valid, RSCS sends the file to its destination on the local node or queues it on another link to continue store-and-forward processing. If the destination is not valid, RSCS spools the file to the system printer or punch.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the TAG <b>Word 3 (+8)</b> Address of a 70-byte accounting record work area

### Exit Conditions

On return, Exit 5 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and creates a standard RECEIVE accounting record.
4	RSCS does not call the next exit routine associated with this exit point and creates a standard RECEIVE accounting record.



Return Code	Results
8	RSCS does not create the standard accounting record nor call the next exit routine associated with Exit 5.
12	RSCS calls the next exit routine associated with Exit 5 but does not create the standard accounting record.

### Programming Considerations

- If your exit routine modifies any fields in the file's TAG element (for example, class or distribution code), RSCS uses those new characteristics to receive the file.

Do not alter any fields in the TAG element if a store-and-forward file is being processed.

- To generate an accounting record, use DIAGNOSE code X'4C' For more information, see [z/VM: CP Programming Services](#).

## Exit 6 – TAG Priority Change

Use Exit 6 to change the priority of a file before RSCS queues it for transmission on a link (the CHANGE command can also modify the priority). Your exit routine can also ensure that the TAG priority option is not misused (for example, priority 1 for a very large file). The exit routine can also define the criteria for selecting file priority (for example, record count, origin user ID, or location ID).

### Point of Processing

Process	Exit Attribute
Spool file processing	Serially reusable

Exit 6 is called each time a file is about to be queued on one or more links. The file can be queued on a link when it is processed during a real or internal reorder. It can also be queued when a CHANGE or TRANSFER command is processed. The TAGPRIOR field contains the file's priority value. This value is set by the file's origin, incoming NJE headers, or a previous call to Exit 6.

On return from the exit, TASHADOW elements are queued on each link that can send the file. The TASHADOW elements are queued according to how each link queues files (priority, FIFO, or size). For priority queuing, TAGPRIOR field in the TAG element determines the file's priority on the link.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the TAG element

### Exit Conditions

On return, Exit 6 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point. Normal processing continues; the file is queued on appropriate links.

### Programming Considerations

- Your exit routine should not alter any fields in the file's TAG element other than TAGPRIOR.

- For more information about changing a file's priority on an individual link, see [“Exit 31 – Sort Priority Change” on page 103](#).

## Exit 7 – Auto-Answer Sign-On Time Out

Use Exit 7 to create an accounting record, or perform other functions, when the sign-on time out value for an auto-answer port expires.

### Point of Processing

Process	Exit Attribute
Auto-answer	Reentrant

Exit 7 is called when an auto-answer task receives a phone call, but has not received any data records during the time out period. The default time out period is 5 minutes.

On return from the exit, RSCS disables the port if the port has reached the maximum number of sign-on attempts. Otherwise, the call is terminated and must be dialed again.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the PORT entry

### Exit Conditions

On return, Exit 7 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues regular processing.

### Programming Considerations

- Your exit routine should not modify any accessible control blocks.
- To generate an accounting record at this exit, use CP DIAGNOSE code X'4C'. For more information, see [z/VM: CP Programming Services](#).

## Exit 8 – Auto-Answer Unrecognizable Data

Use Exit 8 to create an accounting record when an auto-answer port receives unrecognizable data.

### Point of Processing

Process	Exit Attribute
Auto-answer	Reentrant

Exit 8 is called when an auto-answer task receives a phone call that contains unrecognizable data in the first record. This may occur if the data does not match a sign-on card format supported by RSCS. It may also occur if the link-identifier of the sign-on card does not match any links defined to RSCS.

On return, if return code 8 is not issued, RSCS disables the port if it has reached the maximum number of sign-on attempts. If this number has not been reached, the call is terminated and will be re-enabled.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the PORT entry <b>Word 3 (+8)</b> Address of a halfword field containing the length of the unrecognizable data record <b>Word 4 (+12)</b> Address of the data record

### Exit Conditions

On return, Exit 8 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point.
4	RSCS does not call any other exit routines associated with this exit point.
8	RSCS returns control to GCS. RSCS does not increment the incorrect sign-on counter nor end the phone call.

### Programming Considerations

- Your exit routine should not modify any accessible control blocks.

## Exit 8

- To generate an accounting record at this exit, use CP DIAGNOSE code X'4C'. For more information, see [z/VM: CP Programming Services](#).

## Exit 9 – Auto-Answer Sign-On Validation

Use Exit 9 to create an accounting record or reject a sign-on attempt when an auto-answer port receives a valid sign-on card.

Your exit routine can reject sign-on cards using the criteria you define for your installation. For example, you can make some Binary Synchronous Communications (BSC) links ineligible for an auto-answer session or you can make links ineligible at defined times.

### Point of Processing

Process	Exit Attribute
Auto-answer	Reentrant

Exit 9 is called when an auto-answer task receives a phone call and has verified that the sign-on record is valid. However, RSCS has not completed the link validation process.

On return, if the exit routine does not issue return code 8, RSCS transforms the dial-up task into the link driver specified on the sign-on record.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the PORT entry <b>Word 3 (+8)</b> Address of the LINKTABL entry <b>Word 4 (+12)</b> Address of a halfword field containing the length of the sign-on record <b>Word 5 (+16)</b> Address of the sign-on record

### Exit Conditions

On return, Exit 9 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.

<b>Return Code</b>	<b>Results</b>
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS rejects this sign-on attempt.

### **Programming Considerations**

- If RSCS recognizes the sign-on identification and finds a valid link identifier within the sign-on card, it determines that the sign-on card is valid. However, each link driver must determine if the *entire* sign-on card is valid for that link. For example, a password may be incorrect or may not have been specified.
- Your exit routine should not modify any accessible control blocks.
- To generate an accounting record at this exit, use CP DIAGNOSE code X'4C'. For more information, see [\*z/VM: CP Programming Services\*](#).



## Exit 10 – Auto-Answer Sign-On Reject

Use Exit 10 to create an accounting record, or perform other functions, when RSCS receives a valid sign-on card from an auto-answer port that, after further validation, has been rejected by the associated link driver.

### Point of Processing

Process	Exit Attribute
Auto-answer	Reentrant

Exit 10 is called when an RSCS auto-answer task receives a call that contains a sign-on card with valid sign-on identification and a valid link identifier. However, the associated link driver has rejected this sign-on attempt because the specified link type is not valid, the link is already active, or the password is missing or incorrect. The sign-on attempt may also have been rejected by an Exit 9 routine.

On return, RSCS disables the port if the maximum number of sign-on attempts is reached. Otherwise, RSCS terminates the call and must re-enable the port.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the PORT table entry <b>Word 3 (+8)</b> Address of the LINKTABL entry <b>Word 4 (+12)</b> Address of a halfword field containing the length of the sign-on record <b>Word 5 (+16)</b> Address of the sign-on record

### Exit Conditions

On return, Exit 10 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.

## **Programming Considerations**

- Your exit routine should not modify any accessible control blocks.
- To generate an accounting record for sign-on attempts that are not valid, your exit routine can use CP DIAGNOSE code X'4C'. For more information, see [\*z/VM: CP Programming Services\*](#).

## Exit 11 – NJE Job Header Creation

Use Exit 11 to scan the job header, as created by RSCS, and, as needed, change fields or add user sections to the job header. For example, you can add sections to the header to associate other information with the files, including: accounting and security information, and other file characteristics.

The NHDTR macro contains the recommended format for the job header and user sections. For more information on the recommended format of a user section, 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](https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf)).

### Point of Processing

Process	Exit Attribute
NJE header creation	Reentrant

Exit 11 is called as RSCS creates an NJE job header for a file. Any job header options specified on the CP TAG are already reflected in the job header RSCS creates. For more information about TAG options, see *z/VM: RSCS Networking Operation and Use*.

Files that originate from the local node or from a workstation connected to the local node do not contain NJE headers. RSCS will create a job header for files that will be sent on a networking or list processor link. Exit 11 is *not* called for files that already have NJE headers.

On return, if your exit routine issues return code 0 or 4, RSCS uses the NJE header created before Exit 11 was called. RSCS then sends the first part of the file on the appropriate link and continues processing.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the TIB for the stream on which the file is sent</li> <li><b>Word 4 (+12)</b> Address of the job header, as RSCS has created it before Exit 11 was called</li> <li><b>Word 5 (+16)</b> Address of the file's TAG element</li> <li><b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none</li> </ul>

## Exit Conditions

On return, Exit 11 sets the following register contents. To ensure the data in R0 and R1 is restored for any repeated calls to Exit 11, your exit routine should specify REGS=(0,12) on the REXIT macro. For more information, see [“REXIT – Defining a Module Return Point” on page 295](#).

Register	Contents
R0	Length of the user section to add to the job header. RSCS uses this register only for return codes 8, 12, 16, or 20.
R1	Address of the user section to add to the job header. R1 is used only for return codes 8, 12, 16, or 20.
R2 - R13	Restored to the same values as on entry.
R14	Not applicable.
R15	Return code.

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the job header it created. This is the final section of the job header.
12	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the job header it created. RSCS then re-calls all exit routines associated with this exit point.
16	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the job header it created. RSCS then calls the next exit routine associated with the point. Use this return code to create more than one user section.
20	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the job header it created. RSCS then re-calls this exit routine. Use this return code to create more than one user section and to check the results of adding the new section.

## Programming Considerations

- If your exit routine issues return code 0 or 4, it must not pass back a user section. If one is passed back, RSCS ignores it.
- After returning from the exit, the following conditions end the link and cause the specified user abend:

Abend	Condition
7	A user section without identifier B'11XXXXXX' (NJHUTYPE X'C0').
7	User section has the same identifier and modifier as an existing user section.
8	The length of a user section caused the total job header length to exceed 32764 records.
8	A user section is 1 - 3 bytes long.

Abend	Condition
9	The exit routine issued return code 8 or 12 but did not set up R0 and R1 correctly (one or both contain 0).

## Exit 12 – NJE Data Set Header Creation

Use Exit 12 to scan the data set header as created by RSCS, change some fields, and create user sections for the data set header. You can add sections to the header to associate other information with the file, including: accounting and security information and other file characteristics.

The NHDTR macro contains the recommended format of the data set header and user sections. For more information, 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).

### Point of Processing

Process	Exit Attribute
NJE header creation	Reentrant

Exit 12 is called as RSCS creates a data set header for a file that will be sent on a networking link but does not have NJE headers. Files that originate from the local node or from a workstation connected to the local node do not have NJE headers. RSCS does not create data set headers for SYSIN files.

Before calling Exit 12, RSCS creates a data set header using information in the file's TAG element, XAB, and other data areas. If a LISTPROC-type link is processing a file that contains an unprocessed distribution list, several dataset headers may be created and included in the file.

On return if your exit routine issues return code 0 or 4, RSCS uses the data set header it created before calling Exit 12 to send the file.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the TIB for the stream on which the file is sent</li> <li><b>Word 4 (+12)</b> Address of the data set header, as RSCS has created it before calling Exit 12</li> <li><b>Word 5 (+16)</b> Address of the file's TAG element</li> <li><b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none</li> </ul>

## Exit Conditions

On return, Exit 12 sets the following register contents. To ensure the data in R0 and R1 is restored, your exit routine should specify REGS=(0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point”](#) on page 295).

Register	Contents
R0	Length of the user section, created by the exit routine, to add to the data set header. RSCS uses this register only for return codes 8, 12, 16, or 20.
R1	Address of the user section, created by the exit routine, to add to the data set header. R1 is used only for return codes 8, 12, 16, or 20.
R2 - R13	Restored to the same values as on entry.
R14	Not applicable.
R15	Return code.

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routine associated with this exit point and continues usual processing.
8	RSCS appends the user section created by the exit routine (pointed to, on return, by R0 and R1) to the data set header. This is the final section of the data set header.
12	RSCS appends the user section created by the exit routine (pointed to, on return, by R0 and R1) to the data set header. RSCS then re-calls Exit 12 so that the exit routine can create additional user sections.
16	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the data set header it created. RSCS then calls the next exit routine associated with the point. Use this return code to create more than one user section.
20	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the data set header it created. RSCS then re-calls this exit routine. Use this return code to create more than one user section and to check the results of adding the new section.

## Programming Considerations

- If an exit routine issues return code 0 or 4, it must not pass back a user section. If one is passed back, RSCS ignores it.
- After returning from the exit, the following conditions result in the link ending and the specified userabend:

Abend	Condition
7	A user section without identifier B'11XXXXXX' (NDHUTYPE X'C0').
7	User section has the same identifier and modifier as an existing user section.
8	The length of user section created by the exit routine caused the total data set header length to exceed 32764 records.
8	A user section is 1 - 3 bytes long.

## Exit 12

Abend	Condition
9	The exit routine issued return code 8 or 12 but did not set up R0 and R1 correctly (one or both contain 0).



## Exit 13 – NJE Job Trailer Creation

Use Exit 13 to scan the job trailer created by RSCS, change some fields, or add user sections to the job trailer. You can add sections to the trailer to associate other information with the files, including: accounting and security information and additional file characteristics.

The NHDTR macro contains the recommended format of the job trailer and user sections. For more information on the recommended formats, 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](https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf)).

### Point of Processing

Process	Exit Attribute
NJE header creation	Reentrant

Exit 13 is called as RSCS creates an NJE job trailer for a file that will be sent on a networking link but does not have a job trailer. Files that originate on the local node or from a workstation connected to the local node do not have NJE job trailers. RSCS creates the job trailer from information in the file's TAG element, CP SFBLOK, and other data areas. For more information about the SFBLOK, see [z/VM: CP Programming Services](#).

On return, if your exit routine issues return code 0 or 4, RSCS sends the file, using the NJE job trailer it created before calling Exit 13.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the TIB for the stream on which the file is sent <b>Word 4 (+12)</b> Address of the job trailer, as RSCS has created it before calling Exit 13 <b>Word 5 (+16)</b> Address of the file's TAG element <b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none

### Exit Conditions

On return, Exit 13 sets the following register contents. To ensure the data in R0 and R1 is restored on multiple calls to Exit 13, your exit routine should specify REGS=(0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point” on page 295](#)).

Register	Contents
R0	Length of the user section, created by the exit routine, to add to the job trailer. RSCS uses this register only for return codes 8, 12, 16, or 20.
R1	Address of the user section, created by the exit routine, to add to the job trailer. R1 is used only for return codes 8, 12, 16, or 20.
R2 - R13	Restored to the same values as on entry.
R14	Not applicable.
R15	Return code.

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routine associated with this exit point and continues usual processing.
8	RSCS appends a user section created by the exit routine (pointed to on return by R0 and R1) to the job trailer. This is the final section of the job trailer.
12	RSCS appends the user section created by the exit routine (pointed to on return by R0 and R1) to the job trailer. RSCS then re-calls all exit routines associated with this exit point.
16	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the job header it created. RSCS then calls the next exit routine associated with this exit point. Use this return code to create more than one user section.
20	RSCS appends the user section created by the exit routine (pointed to by R0 and R1 on return) to the job header it created. RSCS then re-calls this exit routine. Use this return code to create more than one user section and to check the results of adding the new section.

## Programming Considerations

- If your exit routine issues return code 0 or 4, it must not pass back a user section. If one is passed back, RSCS ignores it.
- After returning from the exit, the following conditions end the link and cause the specified user abend:

Abend	Condition
7	A user section without identifier B'11XXXXXX' (NJTUTYPE X'C0').
7	User section has the same identifier and modifier as an existing user section.
8	A length of a user section caused the total header length to be greater than 32764 records.
8	A user section has a length of 1 - 3 bytes.
9	A return code of 8 or 12 was issued without setting up R0 and R1 correctly (one or both contain 0).

## Exit 14 – NJE Job Header Reception

---

Use Exit 14 to scan the job header file received by RSCS before RSCS updates the TAG element. Your exit routine can examine information from the following sections:

- NJE header, which contains information you can use to determine if the file should be rejected or rerouted based on:
  - Destination
  - Origin
  - Attributes
  - Security information
- User sections, which contain information about:
  - Accounting
  - Security
  - Other file characteristics
- Other header sections, which provide information for:
  - Accounting routines
  - Security validation routines

The NHDTR macro contains the format for the job header RSCS creates. However, headers received on the link may have been created by a different release of RSCS or by another product. For more information the general format of all NJE job headers, 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](https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf)).

The NHDTR macro also contains the recommended format for a user section. However, you should ensure that your exit routine can process the DSECTs specified by the exit routine that created the user section. These areas may differ from the recommended format (see [“Exit 11 – NJE Job Header Creation”](#) on page 61 for more information).

If you want to override the information in the job header after RSCS updates the information in the TAG element, see [“Exit 41 – NJE Job Header Post-Processing”](#) on page 121 for more information.

### Point of Processing

Process	Exit Attribute
NJE header reception	Reentrant

Exit 14 is called each time RSCS receives a file over a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). Each of these files will have a job header. As a file is sent, RSCS first receives its job header, which contains information about the remaining file transmission. Exit 14 is called before RSCS places any information from the general section into a TAG element for the file. RSCS also calls Exit 14 when it finds a user section in the job header.

On return, if your exit routine issues return code 0 or 4, RSCS transfers the information in the general section of the header into various control blocks (for example, the TAG element). RSCS then receives the rest of the file. RSCS does not process the data in any user sections of the file's job header. Rather, it continues to process the remaining sections of the header.

## Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	<p>A code indicating why the exit is called:</p> <p><b>0</b> The general section has been processed.</p> <p><b>4</b> A user section has been found.</p>
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the RIB for the stream on which RSCS receives the file</p> <p><b>Word 4 (+12)</b> Address of the job header</p> <p><b>Word 5 (+16)</b> Address of the file's TAG element</p> <p><b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none</p> <p><b>Word 7 (+24)</b> Address of the user section being processed, or 0 if this exit point is called when RSCS processes the general section of the job header</p>

## Exit Conditions

On return, Exit 14 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing the job header.
4	RSCS does not call any other exit routine associated with this exit point and continues processing the job header.
8	RSCS rejects this file and does not process the job header.

## Programming Considerations

- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any fields in the job headers. If these fields are altered, RSCS may not correctly process NJE store-and-forward files.

- If your exit routine sets the TAGORLOC field in the TAG element, RSCS queues the file for transmission as if it were destined to the node in the TAGORLOC field. Because this field may be reset when RSCS receives the data set header, you may need to provide a similar Exit 15 routine to use this feature.

## Exit 15 – NJE Data Set Header Reception

---

Use Exit 15 to scan the data set header as received by RSCS before RSCS updates the TAG element. Your exit routine can examine information from the following sections:

- NJE header, which contains information you can use to determine if a file should be rejected based on:
  - Destination
  - Origin
  - Attributes
  - Security information
- User sections, which contain:
  - Accounting information
  - Security information
  - Other file characteristics
- Other header sections, which provide information for:
  - Accounting routines
  - Security validation routines

The NHDTR macro contains the recommended format for the job header and user section. However, your exit routine should be able to use the DSECTs specified by the exit routine that created the job header if they differ from the recommended format (see [“Exit 12 – NJE Data Set Header Creation”](#) on page 64 for more information).

If you want to override the information in the data set header after RSCS updates the information in the TAG element, see [“Exit 42 – NJE Data Set Header Post-Processing”](#) on page 123 for more information.

### Point of Processing

Process	Exit Attribute
NJE header reception	Reentrant

Exit 15 is called when RSCS receives a file on a networking (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type) link. The exit point is called before RSCS has placed any information from the general section of the data set header in a control block, such as a TAG element. Exit 15 is also called each time RSCS finds a user section in the data set header.

For SYSOUT files, the data set headers identify the various sections of the file and the file's destinations. SYSIN files generated by RSCS do not contain data set headers. SYSIN files generated by some z/OS® systems may contain a data set header. Here, the data set header will contain only a record characteristics change section (RCCS).

Your exit routine can also access the job header for the file; the RIB contains a pointer to the address of the job header.

On return, if your exit routine issues return code 0 or 4, RSCS transfers the information in the general section of the header into various control blocks (for example, the TAG element). RSCS then receives the rest of the file. RSCS does not process the data in any user sections of the file's job header. Rather, RSCS continues to process the remaining sections of the header.

## Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	A code to show why the exit has been called:
	<b>0</b> The general section has been processed.
	<b>4</b> A user section has been processed.
R1	Address of a parameter list that contains:
	<b>Word 1 (+0)</b> Address of the CVT
	<b>Word 2 (+4)</b> Address of the LINKTABL entry
	<b>Word 3 (+8)</b> Address of the RIB for the stream on which the file is received
	<b>Word 4 (+12)</b> Address of the data set header
	<b>Word 5 (+16)</b> Address of the file's TAG element
	<b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none
	<b>Word 7 (+24)</b> Address of the user section being processed, or 0 if the exit point is called for general section processing

## Exit Conditions

On return, Exit 15 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit and continues to process the data set header.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS rejects this file, issues a message, and stops processing the job header.

## Programming Considerations

- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any fields in the job headers. If these fields are altered, RSCS may not correctly process NJE store-and-forward files.
- If your exit routine sets the TAGORLOC field in the TAG element, RSCS queues the file for transmission as if it were destined to the node in the TAGORLOC field.

## Exit 16 – NJE Job Trailer Reception

---

Use Exit 16 to scan the job trailer for each file RSCS receives over a networking link before RSCS updates the TAG element for the file. Your exit routine can examine information from the following sections:

- NJE header, which contains information you can use to determine if a file should be rejected based on:
  - Destination
  - Origin
  - Attributes
  - Security information
- User sections, which contain:
  - Accounting information
  - Security information
  - Other file characteristics
- Other header sections, which provide information for:
  - Accounting routines
  - Security validation routines

The NHDTR macro contains the recommended format for the job header and user section. However, your exit routine should also be able to use the DSECTs specified by the exit routine that created the job header if they differ from the recommended format. For more information, see [“Exit 13 – NJE Job Trailer Creation”](#) on page 67.

If you want to override the information in the job trailer after RSCS updates the information in the TAG element, see [“Exit 43 – NJE Job Trailer Post-Processing”](#) on page 125 for more information.

### Point of Processing

Process	Exit Attribute
NJE header reception	Reentrant

Exit 16 is called each time RSCS receives a file over a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). The job trailer is the last part of the file transmission. Exit 16 is called before RSCS places any information from the general section into a TAG element for the file. RSCS also calls Exit 16 when it finds a user section in the job header.

On return, if your exit routine issues return code 0 or 4, RSCS transfers the information in the general section of the header into various control blocks (for example, the TAG element). RSCS then receives the rest of the file. RSCS does not process the data in any user sections of the file's job header. Rather, it continues to process the remaining sections of the header.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions”](#) on page 39.



Register	Contents
R0	<p>A code to show why the exit has been called:</p> <p><b>0</b> The general section is being processed.</p> <p><b>4</b> A user section is being processed.</p>
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the RIB for the stream on which the file is received</p> <p><b>Word 4 (+12)</b> Address of the job trailer</p> <p><b>Word 5 (+16)</b> Address of the file's TAG element</p> <p><b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none</p> <p><b>Word 7 (+24)</b> Address of the user section being processed, or 0 if called for general section processing</p>

## Exit Conditions

On return, Exit 16 sets the standard register contents (see [“Standard Exit Conditions”](#) on page 39).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit and continues to process the job trailer.
4	RSCS does not call any other exit routines associated with this exit point and continues to process the job trailer.
8	RSCS rejects the file, issues a message, and stops processing the job trailer.

## Programming Considerations

- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any fields in the job trailers. If these fields are altered, RSCS may not correctly process NJE store-and-forward files.
- The TAG address that is passed to your exit routine is the TAG that corresponds to a file associated with the job trailer. Because the file may have had multiple dataset headers and RSCS may have split the incoming file into several files, several TAG elements may be associated with this job trailer.
- If your exit routine sets the TAGORLOC field in the TAG element, RSCS queues the file for transmission as if it were destined to the node in the TAGORLOC field.

## Exit 17 – Separator Page Selection

Use Exit 17 to select or suppress the separator page for each print file. The separator page can use the RSCS or VM style or a user-generated style. For examples of separator page styles, see [z/VM: RSCS Networking Operation and Use](#).

### Point of Processing

Process	Exit Attribute
Separator page creation	Reentrant

Exit 17 is called as RSCS is about to generate a copy of a print file. RSCS has not determined the separator page format, if any, for the file.

On return, if your exit routine issues return code 0 or 4, RSCS generates the appropriate header and trailer separator pages:

- If the configuration file contains a matching FORM statement, RSCS uses the specified form name.
- If a SEP parameter is specified for the link, RSCS uses that format.
- If neither option is specified, RSCS-style separator pages are generated.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	A code to show why the exit has been called: <b>0</b> The file has been opened and a header page is processed. <b>4</b> The file has been printed and a trailer page is processed.
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the SEPBLOK

### Exit Conditions

On return, Exit 17 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues to process the separator pages.
4	RSCS does not call any other exit routine associated with this exit point and continues to process the separator pages.

Return Code	Results
8	RSCS does not print the header or trailer separator pages.
12	RSCS calls Exit 18 to generate an alternate header or trailer page (see <a href="#">“Exit 18 – Separator Page Generation”</a> on page 78). Do not use return code 12 if you have not loaded or enabled Exit 18.

## Exit 18 – Separator Page Generation

---

Use Exit 18 to create an alternative style for separator pages. Your exit routine returns only one line of a separator page at a time. Use return code 8 to have RSCS call your exit routine for each line in the separator page.

### Point of Processing

Process	Exit Attribute
Separator page generation	Reentrant

Exit 18 is called only when an Exit 17 routine issues return code 12 to generate an alternate style separator page.

On return, if your exit routine issues return code 0 or 4, RSCS stops formatting the separator page and continues processing the file. If a header page is being processed, RSCS starts to print the file. If a trailer page is processed, the file has been printed. If your exit routine issues return code 8, one line is written to the separator page in progress.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	<p>A code to show why the exit has been called:</p> <p><b>0</b> A header page is being processed.</p> <p><b>4</b> A trailer page is being processed.</p>
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the SEPBLOK</p> <p><b>Word 3 (+8)</b> Address of the output data area</p> <p><b>Word 4 (+12)</b> Address of a halfword data area to contain the length of the output data returned</p>

### Exit Conditions

On return, Exit 18 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing.

Return Code	Results
4	RSCS does not call any other exit routines associated with this exit point and continues processing.
8	RSCS writes a line of text to the separator page and calls this exit point again to process the next line.

## Programming Considerations

- To write each line of the separator page, your Exit 18 routine should issue return code 8. To identify the line being processed, your exit routines can maintain a counter in the first fullword of SEPUSER field in the SEPBLOK. This counter can be initialized at Exit 17 and incremented each time an Exit 18 routine generates a line.
- Your Exit 18 routine can use the SEPUSER field to communicate between multiple calls to Exit 17. The Exit 18 routine that generates a separator page should correspond to the Exit 17 routine that called it to make the request.

In exit packages that generate separator pages, Exit 17 routines should store a unique identifier in the second fullword of SEPUSER before issuing return code 8.

An Exit 18 routine can then check this field to identify the exit routine that called it. If it recognizes the Exit 17 routine, the exit routine can generate the separator line and issue return code 8. If the Exit 18 routine does not recognize the identifier, it can issue return code 0. RSCS will then call the next exit routine associated with Exit 18.

- Use the output area (word 3) and length (word 4) fields to store the line of separator page text. The first byte in the output area is a channel-command opcode that can be used to create various effects. The rest of the output area is the separator page text. The following list describes some of the opcodes:

### **X'01'**

Writes data, but does not perform a line feed (use for overprinting)

### **X'09'**

Writes data, then a linefeed

### **X'0B'**

Does a linefeed, but does not print data

### **X'11'**

Writes data, then moves two lines

### **X'13'**

Does two linefeeds, but does not print data

### **X'19'**

Writes data, and spaces three lines

### **X'1B'**

Does three linefeeds, but does not print data

### **X'89'**

Writes data, then a page eject

### **X'8B'**

Does a page eject, but does not write data

**Note:** If you have no separator page text, you *must* pass back an opcode and a blank in the output area field (word 3) and a value of 1 in the length field (word 4). If you do not pass these values back, the link driver may cause RSCS to abend.

- Your exit routine should check the SEPFLAG field in the SEPBLOK data area to determine when to issue a page eject. Some printers require a page eject before the header page is started; others require a page eject after the header page is finished. The printer link drivers set the SEPFLAG flag in SEPBLOK to show when the page eject is required.

- SNARJE-type, MRJE-type, and RJE-type links need to have different headers generated for them if the file being printed is a punch file. This is determined by checking the TAGINDEV flag in the file's TAG element and the SEPFLAG field in the SEPBLOK control block. If the TYPPUN flag is set in TAGINDEV, the file is a punch file. If the SETPUNCH flag is set in SEPFLAG, your exit routine must generate a short, one-line header.

## Exit 19 – Command Screening

Use Exit 19 to determine if RSCS should process a command. You can modify the command element, bypass RSCS authorization checking, or cause RSCS to ignore the command.

### Point of Processing

Process	Exit Attribute
Command processing	Serially reusable

Exit 19 is called when RSCS prepares to run a command. At this point, RSCS has identified the origin of the command, but has not yet processed the command.

On return, if your exit routine issues return code 0 or 4, RSCS processes and runs the command.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	<p>A code indicating where the command originated:</p> <p><b>0</b> From the RSCS console.</p> <p><b>-1</b> From a user through the SMSG command.</p> <p><b>+n</b> From a remote node. R0 contains the address of the LINKTABL entry for the link on which RSCS received the command.</p>
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the command execution request buffer, which is mapped by the CMNDAREA macro</p>

### Exit Conditions

On return, Exit 19 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and processes the command.
4	RSCS does not call any other exit routines associated with this exit and processes the command.

Return Code	Results
8	RSCS does not process this command as if the originator was not authorized to issue the command. RSCS issues message DMT209E to the command originator.
12	RSCS processes the command as if it had been issued by the RSCS console operator.
16	RSCS ignores this command and does not issue a message to the command originator.

### Programming Considerations

- The Type L3 format of the CMNDAREA data area maps the command element. Your exit routine can change any field in the element including the command originator and text.
- To alter the authorization level usually required for a command, your exit routine should issue return code 8 or 12.
- To process a command, your exit routine should issue return code 16; RSCS will then ignore the command.

**Note:** If you want to implement new RSCS commands, you should use Exit 29 (see [“Exit 29 – Unknown Command”](#) on page 99).



## Exit 21 – Spool File Accept/Reject

Use Exit 21 to define the criteria for accepting or rejecting an input spool file. Your exit routine can use any information from the file's TAG element, CP TAG text, and SFBLOK to determine if RSCS should accept the file.

### Point of Processing

Process	Exit Attribute
Spool file processing	Serially reusable

Exit 21 is called when RSCS begins to process an input file. The file may be store-and-forward or may have originated at the local node. At this point, RSCS knows the file's origin and destination from the information in the file's CP TAG. RSCS has also determined if second-level addressing and rerouting are needed.

Exit 21 is called each time RSCS finds a new file in its virtual reader, including when a REORDER command is being processed. Here, Exit 21 may process some incoming files twice.

On return from Exit 21, if your exit routine issues return codes 0 or 4, RSCS accepts files that originated at the local node. RSCS also accepts store-and-forward files if they originated from the RSCS virtual machine and have not been transferred.

RSCS then sends the file to its destinations. Files for local users are transferred to the specified user ID. Files for remote users are queued on all available links to the specified node.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	<p>Address of parameter list, which contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the file's TAG element</p> <p><b>Word 3 (+8)</b> Address of a 70-byte accounting record work area</p> <p><b>Word 4 (+12)</b> Address of the halfword field that contains the length of the file's CP TAG text</p> <p><b>Word 5 (+16)</b> Address of the CP TAG text</p> <p><b>Word 6 (+20)</b> Address of the file's CP SFBLOK</p> <p><b>Word 7 (+24)</b> Address of the REORDER command element if a reorder is in progress, or 0 if a reorder is not in progress</p>

Register	Contents
----------	----------

**Note:** If RSCS has previously processed a file, including one that originated on the local system, it may have a modified origin user tag. RSCS places any information it needs at the beginning of the origin user tag. If the file originated from a remote node, word 5 contains the address of the RSCS store-and-forward tag.

## Exit Conditions

On return, Exit 21 sets the standard register contents (see [“Standard Exit Conditions”](#) on page 39).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS rejects the file. If the file originated at the local node, RSCS transfers it to the originating user ID. If the file originated at a remote node, RSCS purges it and issues message DMT112E.
12	RSCS accepts the file as if it had passed all standard authorization checks.
16	RSCS rejects the file. If the file originated at the local node, RSCS transfers it to the originating user ID. If the file originated at a remote node, RSCS purges it but does not issue a message.

## Programming Considerations

- Your exit routine should not alter any TAG fields for files with NJE headers; changes to the TAG element may not be reflected in the headers before the file is transmitted to the next node.

Store-and-forward files originating from workstation links may not contain NJE headers; they will have a blank TAGCNTRL field. Your exit routine can modify the TAG fields for these files. Any changes are reflected in the NJE headers created to send the file to the next node.

- To accept store-and-forward files from a trusted RSCS virtual machine or another virtual machine on the same processor, your exit routine should issue return code 12.
- If a file is destined to the local node and does not contain a valid destination user ID, RSCS will queue the file on links that are determined by the routing of the \*USER\* node. RSCS rejects the file if no routes to the \*USER\* node have been defined.
- Your exit routine can set the TAGORLOC field in the file's TAG element to force the file to be queued onto one or more links, without regard to the destination node ID of the file. Use this feature to queue files on NOTIFY-type links or to use information, other than that specified on the ROUTE statements and commands, to route the file.
- To generate an accounting record at this exit point, your exit routine should issue return code 0. It can then use CP DIAGNOSE code X'4C' or another facility to create the record.

**Note:** Because Exit 21 is called when a REORDER command is processed, some incoming files may appear more than once in the accounting record.

- For more information about the SFBLOK and DIAGNOSE code X'4C', see [z/VM: CP Programming Services](#).

## Exit 22 – NOTIFY Driver Note Selection

Use Exit 22 to determine if a NOTIFY link driver should create and issue a note when a file is queued on the link.

### Point of Processing

Process	Exit Attribute
NOTIFY link driver	Reentrant

Exit 22 is called when a file is queued on a NOTIFY-type link. RSCS has not yet issued a note.

On return, if your exit routine issues return code 0 or 4, RSCS generates a note. The contents and destination of the note are specified by the TEMPLATE file with which the NOTIFY-type link was initialized.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the NOTEBLOK

### Exit Conditions

On return, Exit 22 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS purges the misdirected file and sends a message to the file originator.
12	RSCS calls Exit 23 to modify the note sent to the file originator.
16	RSCS holds the file on the NOTIFY-type link but does not send a note to the file originator.

### Programming Considerations

- To communicate between Exit 22 and Exit 23, your exit routine should use the NOTEUSER field in the NOTEBLOK.

## Exit 23 – NOTIFY Driver Note Editing

Use Exit 23 to modify the note that the NOTIFY link driver creates in response to a file queued on the link. You can use Exit 23 to perform special substitutions on &-symbols in the note text. You can also use Exit 23 to edit the note's distribution list, which includes all lines before the first blank line in the note.

### Point of Processing

Process	Exit Attribute
NOTIFY link driver	Reentrant

Exit 23 is called only when Exit 22 issues return code 12 to modify the note or its distribution list. Flags in the NOTEFLAG field of the NOTEBLOCK identify the types of records to be processed from the TEMPLATE file. All substitution fields in the file (&-symbols) have been converted to their correct values.

On return, RSCS adds the record to the distribution list or text of the note.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	One of the following codes: <b>0</b> A record is being edited. <b>4</b> There are no more records in the note.
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT. <b>Word 2 (+4)</b> Address of the NOTEBLOCK. <b>Word 3 (+8)</b> Address of an area containing a 1-byte field, which contains the length of the text, and the text of record. This word contains 0 if R0 contains 4.

### Exit Conditions

On return, Exit 23 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing.

Return Code	Results
4	RSCS does not call any other exit routines associated with this exit point and continues processing.
8	RSCS adds the current record to the note but has not yet read the next record from the TEMPLATE file. Rather, it creates a blank line and adds it to the note.

## Programming Considerations

- To add sections to the note or entries to the distribution list, your exit routine should issue return code 8. The list processor sends notes to multiple destinations. If your exit routine specifies additional destinations and your node does not define a LISTPROC-type link, the NOTIFY-type link will end.
- The NOTEUSER field contains a pointer to a record that contains the text to be placed in the note. All IBM-defined variables have already been substituted in the text.

Your exit routine can change this text, alter the length, and issue any valid return code. The length of the text cannot, however, exceed 251 bytes.

- To ensure communication between Exit 22 and Exit 23, your exit routine should use the NOTEUSER field. NOTEUSER can be used to identify the Exit 22 routine that requested the user-format note file and ensure that the appropriate Exit 23 routine is called.

It is recommended to use the second fullword in NOTEUSER to identify the exit routine that issued return code 12 at Exit 22. Exit 22 and Exit 23 should use the first fullword of the NOTEUSER field in the same way (for example, as a counter or a pointer to a working storage area).

## Exit 24 – Spooling CP Command Screening

---

Use Exit 24, with Exit 25, to examine, modify, or extend the CP commands run by the RSCS spool manager task.

### Point of Processing

Process	Exit Attribute
Spool command processing	Serially reusable

Exit 24 is called before RSCS processes a CP command. This exit point may be called by the spool manager or link driver tasks. However, RSCS ensures that its spool resources are serialized and are used by only one task at a time.

On return, if your exit routine issues return code 0 or 4, RSCS runs the CP command as it was entered.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions”](#) on page 39.

Register	Contents
R0	A code identifying the CP command to be processed:
<b>0</b>	CHANGE
<b>4</b>	CLOSE
<b>8</b>	DEFINE
<b>12</b>	DETACH
<b>16</b>	PURGE
<b>20</b>	SPOOL
<b>24</b>	TAG
<b>28</b>	TAG and SPOOL
<b>32</b>	TRANSFER

Register	Contents
R1	<p>Address of parameter list which contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the CP command to be run</p> <p><b>Word 3 (+8)</b> Length of the CP command to be run</p> <p><b>Word 4 (+12)</b> Contents vary, depending on command issued:</p> <p><b>CHANGE</b> Address of TAG element</p> <p><b>CLOSE</b> Address of TAG element, or 0 if none</p> <p><b>DEFINE</b> 0</p> <p><b>DETACH</b> 0</p> <p><b>PURGE</b> Address of TAG element, or 0 if none</p> <p><b>SPOOL</b> Address of TAG element, or 0 if none</p> <p><b>TAG</b> Address of TAG element, or 0 if none</p> <p><b>TAG/SPOOL</b> 0</p> <p><b>TRANSFER</b> Address of TAG element, or 0 if none</p>

**Note:** For code 28, the TAG and SPOOL commands are in the same buffer, separated by a X'15' character. When this buffer is processed by CP DIAGNOSE code X'08', both commands are run.

## Exit Conditions

On return, Exit 24 sets the following register contents. To ensure the data in R0 and R1 is restored, your exit routine should specify REGS=(0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point”](#) on page 295).

Register	Contents
R0	If the return code in R15 is 8, R0 contains the length of the command string passed back by the exit routine. Otherwise, R0 is ignored.
R1	If the return code in R15 is 8, R1 contains a pointer to the substitute command that is to be run. Otherwise, R1 is ignored.
R2 - R13	Restored to the same values as on entry.
R14	Not applicable
R15	Return code

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing.
4	RSCS does not call any other exit routines associated with this exit point and continues processing.
8	RSCS runs a different command supplied by the exit routine; on return, R1 points to the substituted command and R0 contains its length.

## Programming Considerations

### Attention

Use extreme caution when issuing return code 8 from your exit routine. If a substituted command does not perform the same type of function that was originally intended by RSCS, severe damage may occur to RSCS spool file processing.

- To process multiple-line commands, your exit routine should separate the lines of the command by a X'15' byte. The commands can be processed by DIAGNOSE code X'08'.



## Exit 25 – Post-CP Command Screening

---

Use Exit 25, with Exit 24, to examine the CP commands run by the RSCS spool manager task and the return codes from those commands.

### Point of Processing

Process	Exit Attribute
Spool command processing	Serially reusable

Exit 25 is called after RSCS has issued a CP command or a substitute command from Exit 24. It may be run under the spool manager or link driver tasks. However, RSCS ensures that only one task can use its spool resources at a time.

On return, if your exit routine issues return code 0 or 4, RSCS passes the return code from the CP command or substitute command to the calling task.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	A code identifying the CP command that was run:
<b>0</b>	CHANGE
<b>4</b>	CLOSE
<b>8</b>	DEFINE
<b>12</b>	DETACH
<b>16</b>	PURGE
<b>20</b>	SPOOL
<b>24</b>	TAG
<b>28</b>	TAG and SPOOL
<b>32</b>	TRANSFER

Register	Contents
R1	<p>Address of parameter list which contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the CP command that was run</p> <p><b>Word 3 (+8)</b> Length of the CP command that was run</p> <p><b>Word 4 (+12)</b> Contents varies, depending on command issued:</p> <p><b>CHANGE</b> Address of TAG element</p> <p><b>CLOSE</b> Address of TAG element, or 0 if none</p> <p><b>DEFINE</b> 0</p> <p><b>DETACH</b> 0</p> <p><b>PURGE</b> Address of TAG element, or 0 if none</p> <p><b>SPOOL</b> Address of TAG element, or 0 if none</p> <p><b>TAG</b> Address of TAG element, or 0 if none</p> <p><b>TAG/SPOOL</b> 0</p> <p><b>TRANSFER</b> Address of TAG element, or 0 if none</p> <p><b>Word 5 (+16)</b> Return code from the processed command</p>

**Note:** For code 28, the TAG and SPOOL commands are in the same buffer, separated by a X'15' character. When this buffer is processed by CP DIAGNOSE code X'08', both commands are run.

## Exit Conditions

On return, Exit 25 sets the following register contents. To ensure the data in R0 and R1 is restored, your exit routine should specify REGS=(0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point”](#) on page 295).

Register	Contents
R0	If the return code in R15 is 8, R0 contains the return code from the issued command.
R1	Not applicable.
R2 - R13	Restored to the same values as on entry.
R14	Not applicable.
R15	Return code.

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS uses the value passed back in R0 as the return code from the previous CP command.

## Programming Considerations

### Attention

Use extreme caution when using this exit point to process a return code from a substituted command. Severe damage may occur to RSCS spool file processing if the substituted command does not perform a function similar to the original CP command.

- To replace or force usual processing from a nonzero CP return code, your exit routine may supply a different return code in register 0 by issuing return code 8.

## Exit 26 – Link State Change Accounting

Use Exit 26 to create accounting records or to start recovery procedures when a link changes state.

### Point of Processing

Process	Exit Attribute
Link state accounting	Reentrant

Exit 26 is called when any RSCS task causes a link to change state. On return, RSCS returns control to the calling task and the link changes to the specified state.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	<p>Address of parameter list which contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> A fullword value indicating the previous state of the link</p> <p><b>Word 4 (+12)</b> A fullword value indicating the next state of the link:</p> <p><b>0</b> Inactive</p> <p><b>4</b> Retry-wait</p> <p><b>8</b> Dial-queue</p> <p><b>12</b> Starting</p> <p><b>16</b> Active</p> <p><b>20</b> Intervention required</p> <p><b>24</b> Released</p> <p><b>28</b> Connected</p> <p><b>32</b> RPL-wait</p>

## Exit Conditions

On return, Exit 26 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.

## Programming Considerations

- To generate an accounting record at this exit, use CP DIAGNOSE code X'4C'. For more information, see [z/VM: CP Programming Services](#).

## Exit 27 – Message Request Screening

Use Exit 27 to inspect, log, modify, or suppress RSCS messages. The exit routine may inspect or modify any fields in the MSGBLOK passed to it, except the message number. For user-supplied messages, the exit routine may not modify the conversion or translation repository.

**Note:** Exit 27 and Exit 28 replace the function of Exit 20, which was defined in RSCS V2.3.

### Point of Processing

Process	Exit Attribute
Message processing	Reentrant

Exit 27 is called when an RSCS task or an exit routine issues a message. The calling task provides a MSGBLOK, which contains information about the message, such as the message number and destination code. At this point, RSCS has acquired any necessary message work areas. It has also determined that the conversion repository contains information about the specified message.

On return, if your exit routine issues return code 0 or 4, RSCS proceeds to format the message and issue it to all relevant destinations.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the MSGBLOK

### Exit Conditions

On return, Exit 27 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with Exit 27 and continues to process the message.
4	RSCS does not call any other exit routines associated with this exit point and continues to process the message.
8	RSCS does not issue the message.

## Programming Considerations

- Your exit routine can inspect or modify any fields in the MSGBLOK, except for the message number and, for user-supplied messages, the conversion or translation repository fields.
- To suppress a routing code for a message (for example, prevent a message from being routed to the console), your exit routine can use the IRRELMSG routing code.

To do so, your exit routine must ensure that the MSGBRCOD field contains a nonzero value. The exit routine can then turn off the unwanted routing code and turn on the IRRELMSG code. The exit routine must ensure, however, that RSCS does not use the default routing code for the message. (The message conversion repository contains the default routing code.) Your exit routine should then issue return code 0 to allow usual processing of the message.

- To issue a message from an exit routine for Exit 27 or Exit 28, you should specify the RF=REX parameter on the RMSG macro (see [“RMSG – Issuing a Message”](#) on page 299).

## Exit 28 – Message Language Selection

Use Exit 28 to change the language which RSCS uses to issue a message. You can also use Exit 28 to suppress a message or issue it from a different message repository.

**Note:** Exit 27 and Exit 28 replace the function of Exit 20, which was defined in RSCS V2.3.

### Point of Processing

Process	Exit Attribute
Message processing	Reentrant

Exit 28 is called when a message (with routing code O or V only) is about to be formatted and issued in the specified local and network languages. Exit 28 also processes messages that are part of a subscription. However, it is not called if a message is issued with CRI specifications.

On return, if your exit routine issues return code 0 or 4, RSCS uses the specified local language to issue the message to the local user. RSCS issues messages to remote users in the specified network language.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of parameter list which contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the MSGBLOK

### Exit Conditions

On return, Exit 28 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and issues the message.
4	RSCS does not call any other exit routines associated with this exit point and issues the message.
8	RSCS does not issue the message.
12	RSCS issues the message in the local language.
16	RSCS issues the message in the network language.
20	RSCS issues the language-independent form of the message.



## Exit 29 – Unknown Command

---

Use Exit 29 to examine any command that RSCS cannot identify. Your exit routine can process, suppress, or reject the command.

### Point of Processing

Process	Exit Attribute
Command processing	Serially reusable

Exit 29 is called when RSCS does not recognize the first token of a command text. At this point, RSCS knows the origin of the command and the authorization level of the originator. RSCS has also parsed any CRI prefixes that may have been specified on the command.

On return from the exit routine, RSCS issues message DMT201E to show that it did not recognize the command.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	A code indicating the command origin: <b>0</b> From the RSCS console. <b>-1</b> From a user through the SMSG command. <b>+n</b> From a link; R0 contains the address of the LINKTABL for the link on which the command was received.

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT.</p> <p><b>Word 2 (+4)</b> Address of the command execution request buffer, which is mapped by the CMNDAREA macro.</p> <p><b>Word 3 (+8)</b> Address of the MSGBLOK that RSCS set up to issue command responses.</p> <p><b>Word 4 (+12)</b> Address of the AUTHBLOK that describes the command originator (if the originator is an authorized alternate or link operator), or 0 if no AUTHBLOK exists for the command originator. If there are multiple AUTHBLOKs for the command originator, only the first AUTHBLOK found is supplied.</p> <p><b>Word 5 (+16)</b> Address of a 16-byte field containing the name of the unknown command; it is padded on the right with blanks.</p> <p><b>Word 6 (+20)</b> Address of the first character after the command verb in the command execution request buffer.</p>

## Exit Conditions

On return, Exit 29 sets the standard register contents (see [“Standard Exit Conditions”](#) on page 39).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and issues message DMT201E.
4	RSCS does not call any other exit routines associated with this exit point and issues message DMT201E.
8	RSCS ignores this command and does not issue a message.

## Programming Considerations

- To implement your own RSCS commands, your exit routine can use the command verb passed in R1 to identify the command. Your exit routine must parse this command text.  
If the exit routine can process the command, it should issue return code 8. If your exit routine does not recognize the command, it should issue return code 0.
- The Type L3 format of the CMNDAREA data area contains information about the command, such as its origin and text.
- Destination and CRI-related fields are preset in the MSGBLOK passed in R1 to the exit routine. The MSGBLOK also contains space for 16 message substitution variables.

## Exit 30 – Reroute Interception

Use Exit 30 to determine if RSCS should reroute a file, command, or message. If the options on the REROUTE command or statement are insufficient for your installation, you can also use Exit 30 to define additional criteria for rerouting the data traffic.

### Point of Processing

Process	Exit Attribute
Reroute processing	Reentrant

Exit 30 is called when RSCS attempts to reroute a file, message, or command.

On return, if your exit routine issues return code 0 or 4, RSCS uses information provided by the REROUTE command and statement to determine if the message or file can be rerouted.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of FOR node</p> <p><b>Word 3 (+8)</b> Address of FOR user ID</p> <p><b>Word 4 (+12)</b> Address of 1-byte field that contains a symbolic reference to the data being rerouted; the symbols are defined in the REROUTE macro:</p> <p><b>RERCMDS</b> Command</p> <p><b>RERFILES</b> File</p> <p><b>RERMSGs</b> Message</p> <p><b>RERNTRCV</b> Not-received message</p> <p><b>Word 5 (+16)</b> Address of 8-character field containing the new (TO) node ID</p> <p><b>Word 6 (+20)</b> Address of 8-character field containing the new (TO) user ID</p>

### Exit Conditions

On return, Exit 30 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

**Return Codes**

<b>Return Code</b>	<b>Results</b>
0	RSCS calls the next exit routine associated with this exit point and continues processing.
4	RSCS does not call any other exit routines associated with this exit point and continues processing.
8	RSCS does not scan the REROUTE table; the exit routine has supplied the reroute destination node and user ID.
12	RSCS does not scan the REROUTE table; the exit routine has supplied the reroute destination node and user ID. RSCS does not issue a message.
16	RSCS does not reroute the message or file; the REROUTE table is not scanned.

## Exit 31 – Sort Priority Change

Use Exit 31 to change the sort priority of a file's TASHADOW elements before they are queued on links. You can ensure that the TAG priority option is not misused (for example, priority 1 for a very large file). Your exit routine can extend the RSCS file queuing algorithms (for example, time spent at a node can be used to modify the queue order). You can also determine the criteria for file selection (for example, record count, originating user ID, or location ID).

### Point of Processing

Process	Exit Attribute
Spool file processing	Serially reusable

Exit 31 is called when RSCS has determined a TASHADOW element's sort priority according to the link's queuing algorithm (priority, size, or FIFO). RSCS is about to place the TASHADOW element on a link's queue.

On return, RSCS places the TASHADOW element on the queue using the value in the TASSORT field. If your exit routine issues return code 8, RSCS holds the file on the link. If you issue the QUERY command to find the status of the file, the indicator *exit-held* will be displayed in the status column.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the file's TAG element <b>Word 3 (+8)</b> Address of the TASHADOW element <b>Word 4 (+12)</b> Address of the LINKTABL for the link on which the file will be queued <b>Word 5 (+16)</b> Address of the REORDER command element if a reorder is in progress, or 0 if a reorder is not in progress

### Exit Conditions

On return, Exit 31 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Meaning to RSCS
0	RSCS calls the next exit routine associated with this exit point and continues processing.

<b>Return Code</b>	<b>Meaning to RSCS</b>
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS queues the file on the link but marks the file as ineligible for transmission.

### **Programming Considerations**

- Do not alter any fields in the file's TAG or TASHADOW element except TASSORT, which contains the sort value that RSCS uses to queue the file.
- To limit file transmission during certain times of the day or on certain links, check the TSHIFT field in the CVT. This field is set by the RSCS SHIFT command. If the file should not be sent during the specified shift, your exit routine should issue return code 8.

## Exit 32 – NMR Reception

Use Exit 32 to scan the Nodal Message Records (NMRs) RSCS receives on a link. The exit routine can use the NMR to determine if the message or command should be rejected based on its destination, origin, attributes, or security information.

### Point of Processing

Process	Exit Attribute
NMR processing	Reentrant

Exit 32 is called each time RSCS receives an NMR on a GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type link. At this point, CMD or MSG rerouting has completed.

On return, if the exit routine issues return code 0 or 4, RSCS sends or runs the command or delivers the message. If a message that was rerouted by a NOTRCVG request is not delivered, Exit 32 processes the message again.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry for the link that received the element <b>Word 3 (+8)</b> Address of command execution request buffer, which is mapped by the CMNDAREA macro

### Exit Conditions

On return, Exit 32 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing the record.
4	RSCS does not call any other exit routines associated with this exit point and continues to process the record.
8	RSCS rejects the record and issues message DMT944E to the NMR originator.
12	RSCS ignores this record but does not issue a message.

**Programming Considerations**

- The Type L3 format of the CMNDAREA data area maps the command or message element. Your exit routine can modify any fields in the element. Any changes are reflected if the command or message is rerouted.



## Exit 33 – User Parm Processing

Use Exit 33 to process the values specified on the UPARM operand of the RSCS DEFINE command.

### Point of Processing

Process	Exit Attribute
Command processing	Serially reusable

Exit 33 is called when the UPARM operand is specified on a DEFINE command. At this point, RSCS has validated all other operands specified on the command.

On return, if your exit routine issues return code 0 or 4, RSCS completes its processing of the DEFINE command. It then queues a REORDER request to the spool manager task.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL for the link whose UPARM has been changed <b>Word 3 (+8)</b> Address of an initialized MSGBLOK

### Exit Conditions

On return, Exit 33 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Meaning to RSCS
0	RSCS calls the next exit routine associated with Exit 33 and continues processing the DEFINE command.
4	RSCS does not call any other exit routine associated with this exit point and continues processing the DEFINE command.
8	The exit routine has rejected the DEFINE command. RSCS issues message DMT559E and stops processing the command.
12	The exit routine has rejected the DEFINE command. RSCS does not issue a message; however, the exit routine may use the supplied MSGBLOK to issue a message.

**Programming Considerations**

- LUSRPARM field of the LINKTABL points to the user parameter string. This string consists of a 2-byte field, which contains the length of the text, and the text of the parameter.
- Use Exit 0 to scan UPARM values specified in the configuration file. You can then use Exit 33 to monitor any changes that may occur to the UPARM values as RSCS operates.

## Exit 34 – Spool Manager Command

Use Exit 34 to determine if RSCS should process a spool manager command.

### Point of Processing

Process	Exit Attribute
Spool manager command	Serially reusable

Exit 34 is called when a task has passed a command element to the spool manager for synchronous processing. At this point, the calling task has also gained exclusive use of all RSCS spool file resources. These resources include TAG and TASHADOW elements.

On return, if your exit routine does not issue return code 8, RSCS processes the command request. It then returns control to the calling task.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the command element

### Exit Conditions

On return, Exit 34 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual command processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual command processing.
8	RSCS does not process the command element nor issue a message.

### Programming Considerations

- Your exit routine can modify the command element before RSCS processes it by issuing return code 0 or 4. To process the command element only in your exit routine, issue return code 8.
- The RSCS communications task queues the encoded forms of the following command elements to the spool manager task: CHANGE, FLUSH, PURGE, REORDER, and TRANSFER. The end-of-task routine can also queue a CLOSE command request to the spool manager task.

## Exit 35 – Dump Processing

Use Exit 35 to determine if RSCS should generate a dump, as part of ESTAE exit processing, when a task abends.

### Point of Processing

Process	Exit Attribute
Dump processing	Serially reusable

Exit 35 is called when a task abends and ESTAE processing begins. At this point, interrupts have been disabled.

On return, RSCS issues a CP DUMP or VMDUMP command, as specified on the DUMP configuration file statement, to process the dump. For more information about the DUMP statement, see [z/VM: RSCS Networking Planning and Configuration](#).

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	<p>A code indicating the task that abended:</p> <p><b>0</b> System task</p> <p><b>4</b> Link driver task</p> <p><b>8</b> Auto-answer task</p>
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Abend code X'00sssuu' (where <i>sss</i> is the system code and <i>uuu</i> is the user abend code)</p> <p><b>Word 3 (+8)</b> Address of the SYSIDENT, LINKTABL, or PORT entry for the task identified in R0</p>

### Exit Conditions

On return, Exit 35 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.

Return Code	Results
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS issues a symptom summary message to the console but does not generate a dump for this abend.

### Programming Considerations

- RSCS does not request a dump or call Exit 35 if an S804, S80A, or S878 abend occurs in a link driver or auto-answer task.
- Return code 8 prevents RSCS from issuing a DUMP or VMDUMP command. However, if GCS does not provide a system diagnostic work area (SDWA), it may perform a dump for RSCS. For example, GCS processes a dump for RSCS if an S13E abend occurs when the FORCE command is issued. Here, the ESTAE exit and Exit 35 are not called.

## Exit 36 – NOTIFY Driver Purge

Use Exit 36 to determine if a NOTIFY link driver should purge a file.

### Point of Processing

Process	Exit Attribute
NOTIFY link driver	Reentrant

Exit 36 is called when the specified PURGE period has expired and a file queued on a NOTIFY-type link is about to be purged.

On return, if your exit routine issues return code 0 or 4, RSCS purges the file and sends a message to the file originator.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the NOTEBLOK

### Exit Conditions

On return, Exit 36 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point, purges the file, and issues message DMT834I.
4	RSCS does not call any other exit routines associated with this exit point, purges the file, and issues message DMT834I.
8	RSCS does not purge the file.

### Programming Considerations

- If your exit routine issues return code 8, the file remains on the NOTIFY-type link for the specified purge period. If this time expires and the user has not taken action on the file, Exit 36 will process the file again.

## Exit 37 – NJE Job Header Transmission

Use Exit 37 to scan information from user sections in the NJE job header before a file is transmitted. All files transmitted over networking links have job headers. The job header, which describes the file's characteristics, is always the first part of the file transmission.

The NHDTR macro contains the format of the job header RSCS creates. However, headers for files being transmitted on the link may have been created by a different release of RSCS or by another product. For more information on the general format of all NJE job headers, 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).

The NHDTR macro also contains the recommended format for a user section. However, you should ensure that your exit routine can process the DSECTs specified by the exit routine that created the user section; these areas may differ from the recommended format.

### Point of Processing

Process	Exit Attribute
NJE header transmission	Reentrant

Exit 37 is called as RSCS prepares to send an NJE store-and-forward file on a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). The exit routine is called before RSCS segments the NJE header into buffers for the transmission.

On return from your exit routine, RSCS sends the file on the networking link.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the TIB for the stream on which the file is sent</li> <li><b>Word 4 (+12)</b> Address of the job header</li> <li><b>Word 5 (+16)</b> Address of the file's TAG element</li> </ul>

### Exit Conditions

On return, Exit 37 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

**Return Codes**

<b>Return Code</b>	<b>Results</b>
0	RSCS calls the next exit routine associated with this exit point and continues processing the file.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.

**Programming Considerations**

- Your exit routine should not alter any fields in the job headers. When RSCS processes NJE store-and-forward files, it assumes that all NJE headers are preserved.
- Exit 37 is not called for job headers that are initially built at the local node. Use Exit 11 to process job headers created at the local node (see [“Exit 11 – NJE Job Header Creation”](#) on page 61).



## Exit 38 – NJE Data Set Header Transmission

Use Exit 38 to scan information from user sections in the NJE data set header before a file is transmitted. Generally, only SYSOUT files transmitted over networking links contain data set headers. A SYSIN file from some z/OS systems may contain a data set header. If present, the data set header will contain only a Record Characteristics Change Section (RCCS) (identifier X'00', modifier X'40'). RSCS does not generate an RCCS section.

The NHDTR macro contains the format of the data set header RSCS creates. However, headers for files being transmitted on the link may have been created by a different release of RSCS or by another product. For more information on the general format of all NJE data set headers, 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)).

The NHDTR macro also contains the recommended format for a user section. However, you should ensure that your exit routine can process the DSECTs specified by the exit routine that created the user section; these areas may differ from the recommended format.

### Point of Processing

Process	Exit Attribute
NJE header transmission	Reentrant

Exit 38 is called as RSCS prepares to send an NJE store-and-forward file over a networking link. The exit routine is called before RSCS places the NJE header into transmission buffers. LISTPROC-type, NJE-type, SNANJE-type, and TCPNJE-type links may segment the headers before placing them in buffers.

On return from your exit routine, RSCS sends the file on the networking link.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the TIB for the stream on which the file is sent</li> <li><b>Word 4 (+12)</b> Address of the job header</li> <li><b>Word 5 (+16)</b> Address of the file's TAG element</li> </ul>

### Exit Conditions

On return, Exit 38 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

**Return Codes**

<b>Return Code</b>	<b>Results</b>
0	RSCS calls the next exit routine associated with this exit point and continues processing the file.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.

**Programming Considerations**

- Your exit routine should not alter any fields in the job headers. When RSCS processes NJE store-and-forward files, it assumes that all NJE headers are preserved.
- Exit 38 is not called for data set headers that are initially built at the origin node. Use Exit 12 to process data set headers created at the local node (see [“Exit 12 – NJE Data Set Header Creation”](#) on page 64).

## Exit 39 – NJE Job Trailer Transmission

Use Exit 39 to scan information from user sections in the NJE job trailer before a file is transmitted. All files sent over networking links contain job trailers. The job trailer is always the last part of the file transmission.

The NHDTR macro contains the format of the job trailer that RSCS creates. However, trailers for files being transmitted on the link may have been created by a different release of RSCS or by another product. For more information on the general format of all NJE job trailers, 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)).

The NHDTR macro also contains the recommended format for a user section. However, you should ensure that your exit routine can process the DSECTs specified by the exit routine that created the user section; these areas may differ from the recommended format.

### Point of Processing

Process	Exit Attribute
NJE header transmission	Reentrant

Exit 39 is called as RSCS prepares to send a file over a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). The exit routine is called before RSCS places the NJE header into transmission buffers. LISTPROC-type, NJE-type, SNANJE-type, TCPNJE-type links may segment the headers before placing them in the buffers.

On return from your exit routine, RSCS sends the file on the networking link.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see “Standard Entry Conditions” on page 39.

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the TIB for the stream on which the file is sent</li> <li><b>Word 4 (+12)</b> Address of the job header</li> <li><b>Word 5 (+16)</b> Address of the file's TAG element</li> </ul>

### Exit Conditions

On return, Exit 39 sets the standard register contents (see “Standard Exit Conditions” on page 39).

**Return Codes**

<b>Return Code</b>	<b>Results</b>
0	RSCS calls the next exit routine associated with this exit point and continues processing the file.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.

**Programming Considerations**

- Your exit routine should not alter any fields in the job headers. When RSCS processes NJE store-and-forward files, it assumes that all NJE headers and trailers are preserved.
- Exit 39 is not called for job trailers that are initially built at the local node. Use Exit 13 to process job trailers created at the local node (see [“Exit 13 – NJE Job Trailer Creation”](#) on page 67).

## Exit 40 – NJE Record Reception

Use Exit 40 to scan all records and record segments, except NJE headers, for each file RSCS receives on a networking link.

### Point of Processing

Process	Exit Attribute
NJE record reception	Reentrant

Exit 40 is called when RSCS receives a file on a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). At this point, RSCS has previously received the job header and has just decompressed an NJE record or segment. However, RSCS does not expect to receive segmented records. The exit routine is called before RSCS writes the record to an output device associated with the transmission stream.

On return, if your exit routine does not reject the record, RSCS writes the record to the appropriate output device.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the RIB for the stream on which the file is sent</li> <li><b>Word 4 (+12)</b> Address of a halfword field that contains the length of the data</li> <li><b>Word 5 (+16)</b> Address of the SRCB</li> <li><b>Word 6 (+20)</b> Pointer to the data</li> </ul>

### Exit Conditions

On return, Exit 40 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing the file.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS rejects the file.

## Programming Considerations

- The data pointed to by Word 6 of R1 begins with a carriage control byte, if applicable, or a first data byte. It does not include length fields found in the NJE records.
- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any part of the records or NJE headers associated with the file.
- Word 5 of the R1 parameter list contains the sub record control bytes (SRCBs) for the record, which are defined in the NJEEQU macro.

## Exit 41 – NJE Job Header Post-Processing

Use Exit 41 to scan the job header file received by RSCS after RSCS updates the TAG element. Your exit routine can override the information RSCS puts in the TAG element from the job header.

The NHDTR macro contains the format for the job header RSCS creates. However, headers received on the link may have been created by a different release of RSCS or by another product. For more information the general format of all NJE job headers, 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](https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/hasa600_v2r5.pdf)).

The NHDTR macro also contains the recommended format for a user section. However, you should ensure that your exit routine can process the DSECTs specified by the exit routine that created the user section. These areas may differ from the recommended format. See “Exit 11 – NJE Job Header Creation” on page 61 for more information. Also, see “Exit 14 – NJE Job Header Reception” on page 69 for more information about scanning the job header before RSCS updates the TAG element.

### Point of Processing

Process	Exit Attribute
NJE header reception	Reentrant

Exit 41 is called each time RSCS receives a file over a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). Each of these files will have a job header. As a file is sent, RSCS first receives its job header, which contains information about the remaining file transmission. Exit 41 is called after RSCS places any information from the general section into a TAG element for the file.

On return, if your exit routine issues return code 0 or 4, RSCS receives the rest of the file.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see “Standard Entry Conditions” on page 39.

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the RIB for the stream on which RSCS receives the file</li> <li><b>Word 4 (+12)</b> Address of the job header</li> <li><b>Word 5 (+16)</b> Address of the file's TAG element</li> <li><b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none</li> </ul>

## Exit Conditions

On return, Exit 41 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues usual processing.
4	RSCS does not call any other exit routine associated with this exit point and continues usual processing.
8	RSCS rejects this file.

## Programming Considerations

- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any fields in the job headers. If these fields are altered, RSCS may not correctly process NJE store-and-forward files.
- If your exit routine sets the TAGORLOC field in the TAG element, RSCS queues the file for transmission as if it were destined to the node in the TAGORLOC field. Because this field may be reset when RSCS receives the data set header, you may need to provide a similar Exit 42 routine to use this feature.



## Exit 42 – NJE Data Set Header Post-Processing

Use Exit 42 to scan the data set header as received by RSCS after RSCS updates the TAG element. Your exit routine can override the information RSCS puts in the TAG element from the data set header.

The NHDTR macro contains the recommended format for the job header and user section. However, your exit routine should be able to use the DSECTs specified by the exit routine that created the job header if they differ from the recommended format. See [“Exit 12 – NJE Data Set Header Creation” on page 64](#) for more information. Also, see [“Exit 15 – NJE Data Set Header Reception” on page 72](#) for more information about scanning the data set header before RSCS updates the TAG element.

### Point of Processing

Process	Exit Attribute
NJE header reception	Reentrant

Exit 42 is called when RSCS receives a file on a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). The exit point is called after RSCS places any information from the general, VM, 3800, and output processing sections of the data set header into a control block, such as a TAG element and after Exit 15 has been called to process any user sections.

For SYSOUT files, the data set headers identify the various sections of the file and the file's destinations. SYSIN files generated by RSCS do not contain data set headers. SYSIN files generated by some z/OS systems may contain a data set header. Here, the data set header will contain only a record characteristics change section (RCCS).

Your exit routine can also access the job header for the file; the RIB contains a pointer to the address of the job header.

On return, if your exit routine issues return code 0 or 4, RSCS receives the rest of the file.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the RIB for the stream on which the file is received <b>Word 4 (+12)</b> Address of the data set header <b>Word 5 (+16)</b> Address of the file's TAG element <b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none

## Exit Conditions

On return, Exit 42 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS rejects this file.

## Programming Considerations

- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any fields in the job headers. If these fields are altered, RSCS may not correctly process NJE store-and-forward files.
- If your exit routine sets the TAGORLOC field in the TAG element, RSCS queues the file for transmission as if it were destined to the node in the TAGORLOC field.

## Exit 43 – NJE Job Trailer Post-Processing

Use Exit 43 to scan the job trailer for each file RSCS receives over a networking link after RSCS updates the TAG element. Your exit routine can override the information RSCS puts in the TAG element from the job trailer.

The NHDTR macro contains the recommended format for the job header and user section. However, your exit routine should also be able to use the DSECTs specified by the exit routine that created the job header if they differ from the recommended format. See [“Exit 13 – NJE Job Trailer Creation” on page 67](#) for more information. Also, see [“Exit 16 – NJE Job Trailer Reception” on page 74](#) for more information about scanning the job trailer before RSCS updates the TAG element.

### Point of Processing

Process	Exit Attribute
NJE header reception	Reentrant

Exit 43 is called each time RSCS receives a file over a networking link (GATEWAY-type, LISTPROC-type, NJE-type, SNANJE-type, or TCPNJE-type). The job trailer is the last part of the file transmission. Exit 43 is called after RSCS places any information from the general section into a TAG element for the file and after Exit 16 has been called to process all user sections.

On return, if your exit routine issues return code 0 or 4, RSCS receives the rest of the file.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the *first* exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the RIB for the stream on which the file is received <b>Word 4 (+12)</b> Address of the job trailer <b>Word 5 (+16)</b> Address of the file's TAG element <b>Word 6 (+20)</b> Address of the file's XAB, or 0 if there is none

### Exit Conditions

On return, Exit 43 sets the standard register contents (see [“Standard Exit Conditions” on page 39](#)).

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit and continues usual processing.
4	RSCS does not call any other exit routines associated with this exit point and continues usual processing.
8	RSCS rejects the file.

## Programming Considerations

- If your exit routine issues return code 8 to reject a file, RSCS issues a Receiver Cancel with reason code X'2000' to the node that sent the file.
- Your exit routine should not alter any fields in the job trailers. If these fields are altered, RSCS may not correctly process NJE store-and-forward files.
- The TAG address that is passed to your exit routine is the TAG that corresponds to a file associated with the job trailer. Because the file may have had multiple dataset headers and RSCS may have split the incoming file into several files, several TAG elements may be associated with this job trailer.
- If your exit routine sets the TAGORLOC field in the TAG element, RSCS queues the file for transmission as if it were destined to the node in the TAGORLOC field.

## Exit 44 – Link Termination

Use Exit 44 to perform any special termination processing that might be needed for a print output link. You can use this exit for accumulating accounting information or to perform any general clean up of a link. As supplied, RSCS does not provide for any special processing when a printer link is terminated.

### Point of Processing

Process	Exit Attribute
File Account Processing	Reentrant

Exit 44 is called when a printer link is in the process of terminating. On return from the exit routine, RSCS continues termination processing for the link.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the DWA for the link

### Exit Conditions

On return, Exit 44 restores the registers to the same values as upon entry.

### Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues to terminate the link.
4	RSCS does not call any other exit routines associated with this exit point and continues to terminate the link.

### Programming Considerations

- Your exit routine should not alter any fields in the file's LINKTABL, DWA, or CVT.
- Your exit routine can use the ACNTBUFF macro as the basis for any accounting record. Use CP DIAGNOSE code X'4C' to create the record. For more information, see [z/VM: CP Programming Services](#). Your exit routine can also write the accounting information to a CMS file.

## Exit 45 – Output Page Accounting

Use Exit 45 to perform any output page accounting process that may be needed for a printer link. Your exit routine can verify or adjust the actual data buffers being processed on the link. You can use this information for billing purposes and to help you determine the total number of pages being sent to a printer. As supplied, RSCS does not provide for any special processing when a buffer is printed.

### Point of Processing

Process	Exit Attribute
File Account Processing	Reentrant

Exit 45 is called each time a data buffer has been prepared for output to a printer. Your Exit 45 routine can view or alter this data buffer, as required by your installation.

On return, RSCS continues normal buffer output processing for the printer link.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of DWA for the link</li> <li><b>Word 4 (+12)</b> Address of the buffer to be examined</li> <li><b>Word 5 (+16)</b> Address of the length of the buffer</li> </ul>

### Exit Conditions

On return, Exit 45 sets the following register contents. To ensure the data in R1 is restored for any repeated calls to Exit 45, your exit routine should specify REGS= (0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point” on page 295](#)).

Register	Contents
R0	Not applicable.
R1	Actual length of new buffer data contents, if altered; the return code must be set to 4.
R2 - R13	Restored to the same values as on entry.
R14	Not applicable.
R15	Return code.

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing until all routines are called. The last exit will cause the buffer to be sent as is; no adjustments are needed. The buffer must be identical to what was initially passed to the exit routine.
4	RSCS does not call any other exit routines associated with this exit point and continues processing the buffer. No adjustments are needed to send the buffer. The buffer must be identical to what was initially passed to the exit routine.
8	The link driver must readjust the buffer size; register 1 contains the new length of the buffer.

## Programming Considerations

- Your exit routine should not alter any fields in the file's LINKTABL or any other RSCS control blocks.
- Your exit can use the ACNTBUFF macro as the basis of the accounting record. Use CP DIAGNOSE code X'4C' to create the record. For more information, see [z/VM: CP Programming Services](#). Your exit routine can also write the accounting information to a CMS file.

## Exit 46 – Verification of Page Accounting

Use Exit 46 to adjust any output page accounting process that might be needed for billing by pages printed. Exit 46 is valid only for 3270P-type and TN3270E-type links.

This exit is called if an I/O error has occurred while processing the output. As supplied, RSCS does not provide for any special processing when an I/O error has occurred while processing a data buffer. The information about the I/O error condition is available in the IOTABLE control block. Your Exit 46 routine can review this information and perform any needed adjustments. For example, it could request to send the buffer again or adjust the buffer size and issue the I/O again.

### Point of Processing

Process	Exit Attribute
Verification of Output Page Processing	Reentrant

Exit 46 is called each time an I/O error has completed unsuccessfully. Your exit routine can adjust or verify the buffer data and can reissue the I/O request on the 3270P-type link.

On return from the exit, the I/O process will be issued again on the link after the indicated return code actions have completed.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the DWA for the link</li> <li><b>Word 4 (+12)</b> Address of buffer</li> <li><b>Word 5 (+16)</b> Address of the length of the buffer (halfword)</li> </ul>

### Exit Conditions

On return, Exit 46 sets the following register contents. To ensure the data in R1 is restored for any repeated calls to Exit 46, your exit routine should specify REGS= (0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point” on page 295](#)).

Register	Contents
R0	Not applicable.
R1	Actual length of new buffer data contents, if altered; the return code is set to 4.



Register	Contents
R2 - R13	Restored to the same values as on entry.
R14	Not applicable.
R15	Return code.

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues processing until all routines are called. The last exit routine will cause the I/O operation to be retried; the buffer data and length will be the same as it was for the last I/O operation.
4	RSCS does not call any other exit routines associated with this exit point. Processing continues, with the buffer data and length as it was for the last I/O operation.
8	RSCS must adjust the buffer size; register 1 contains the new length. The I/O operation is issued again after this new length has been used. The buffer data contents are left as returned by the exit.

## Programming Considerations

- Your exit routine should not alter any fields in the file's LINKTABL element or any other RSCS control blocks.
- Your exit can use the ACNTBUFF macro as the basis of the accounting record. Use CP DIAGNOSE code X'4C' to create the record. For more information, see [z/VM: CP Programming Services](#). Your exit routine can also write the accounting information to a CMS file.

## Exit 47 – Driver Initialization

Use Exit 47 to perform any initialization required to perform page accounting for an SNA3270P-type link. Your Exit 47 routines can also perform any processing that may be unique to the SNA printer environment at your installation. As supplied, RSCS does not provide for any special processing when an SNA3270P link driver is initialized.

### Point of Processing

Process	Exit Attribute
Verification of Output Page Processing	Reentrant

Exit 47 is called when the SNA Control Task attaches a SNA3270P link driver and passes control to it.

On return from the exit, initialization of the SNA3270P link driver continues. The SNA3270P-type link can start to process file traffic.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the DWA for the link

### Exit Conditions

On return, Exit 47 sets the following register contents. To ensure the data in R1 is restored for any repeated calls to Exit 47, your exit routine should specify REGS=(0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point” on page 295](#)).

Register	Contents
R0	Not applicable
R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues to initialize the link.
4	RSCS does not call any other exit routines associated with this exit point and continues to initialize the link.

## Programming Considerations

- Your exit routine should not alter any fields in the file's LINKTABL or in any other RSCS control blocks.
- Your exit can use the ACNTBUFF macro as the basis of the accounting record. Use CP DIAGNOSE code X'4C' to create the record. For more information, see [z/VM: CP Programming Services](#). Your exit routine can also write the accounting information to a CMS file.

## Exit 48 – Verification of Output Page Error

Use Exit 48 to handle any special processing needs if an error occurs while an SNA3270P-type link is processing a print file. Your Exit 48 routine can accumulate any special accounting that may be needed when a file can no longer be processed. For example, Exit 48 routines can adjust the accounting information that has been gathered up to, and including, the file currently being processed on the link.

### Point of Processing

Process	Exit Attribute
Verification of Output Page Error	Reentrant

Exit 48 is called when an error occurs while an SNA3270P link driver is processing a file.

On return from the exit, the file termination procedure continues. The SNA3270P link driver file termination and requeue processes also continue.

### Entry Conditions

The following table shows the parameters passed in R0 and R1 to the exit routine associated with this exit point. For more information about other register contents, see [“Standard Entry Conditions” on page 39](#).

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <ul style="list-style-type: none"> <li><b>Word 1 (+0)</b> Address of the CVT</li> <li><b>Word 2 (+4)</b> Address of the LINKTABL entry</li> <li><b>Word 3 (+8)</b> Address of the DWA for the link</li> </ul>

### Exit Conditions

On return, Exit 48 sets the following register contents. To ensure the data in R1 is restored for any repeated calls to Exit 48, your exit routine should specify REGS=(0,12) on the REXIT macro (see [“REXIT – Defining a Module Return Point” on page 295](#)).

Register	Contents
R0	Not applicable
R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## Return Codes

Return Code	Results
0	RSCS calls the next exit routine associated with this exit point and continues to terminate the link.
4	RSCS does not call any other exit routines associated with this exit point and continues to terminate the link.

## Programming Considerations

- Your exit routine should not alter any fields in the file's LINKTABL or in any other RSCS control blocks.
- Your exit can use the ACNTBUFF macro as the basis of the accounting record. Use CP DIAGNOSE code X'4C' to create the record. For more information, see [z/VM: CP Programming Services](#). Your exit routine can also write the accounting information to a CMS file.



## Chapter 4. Transmission Algorithm Processing

Networking link drivers (GATEWAY, LISTPROC, NJE, SNANJE, and TCPNJE) support a feature called multistreaming. This feature lets the links send many files at the same time. LISTPROC-type, NJE-type, SNANJE-type, and TCPNJE-type links can support up to 7 transmission streams. GATEWAY-type links can support up to 32 streams; the gateway program you supply determines the number of streams that are used. You specify the number of transmission streams for a link on the STREAMS parameter in a link's PARM text.

Transmission algorithms determine how files are selected for transmission on a networking link. A transmission algorithm is called in the following situations:

- When a link is started (an *open* transmission algorithm request)
- When each file is queued on a link (an *accept* request)
- When the link tries to get a file for transmission (a *select* request)

A transmission algorithm, however, does not affect how a link *receives* files on multiple streams. The number of transmission streams you specify to send files does not have to equal the number of streams on which the link receives files. The RSCS networking link drivers support any number of incoming streams. When communicating with some other systems, however, you may need to inform them of the number of incoming streams.

### Specifying a Transmission Algorithm

The transmission algorithm to be used for a networking link is identified on the TA operand on the PARM configuration statement for the link or in the link operational parameters on the RSCS DEFINE or START command. By specifying the TA=*epname* link parameter, you can have RSCS access transmission algorithms that reside in load modules outside the RSCS load library. See [“External Transmission Algorithms”](#) on page 138.

RSCS can also access transmission algorithms that reside within the RSCS load library. RSCS supplies transmission algorithms 0 and 1 (accessible by specifying TA=0 or TA=1). For compatibility with previous versions of RSCS, you can also create transmission algorithms 2 - F. However, there are some restrictions to using this method to create a transmission algorithm. See [“Internal Transmission Algorithms”](#) on page 142.

You can also specify a transmission algorithm parameter string on the TAPARM parameter. This string can be used to configure a transmission algorithm in several different ways. The string is passed to the transmission algorithm as part of the initial open request when a networking link driver initializes.

### Transmission Algorithm Programming Considerations

Transmission algorithms need only to be serially reusable. You can supply *external* or *internal* transmission algorithms.

External transmission algorithms are similar to other exit routines because they do not reside in the RSCS load library. You can create any number of external transmission algorithms. External transmission algorithms can also support up to 32 transmission streams.

RSCS also provides internal transmission algorithms in DMTAXA. This module contains transmission algorithms 0 and 1, which are supplied by IBM. You do not need to modify these transmission algorithms for usual RSCS processing. DMTAXA also contains the initial structure for transmission algorithms 2 - F. Unlike external exit routines, however, if you modify or create an internal transmission algorithm, you must reassemble DMTAXA and rebuild the RSCS load library to implement your routines. This can reduce the portability of your transmission algorithms and can increase the chance of errors. Also, internal transmission algorithms can support only up to seven transmission streams.

If you modify or create a transmission algorithm, consider the following:

- Links do not start if you specify an undefined transmission algorithm on its START command.
- Any transmission algorithms you supply must provide the processing functions needed to support the transmission algorithm requests.

## External Transmission Algorithms

The first word of the parameter list RSCS passes to an external transmission algorithm points to the CVT, which contains several fields available for use by transmission algorithms.

The second word of the parameter list contains a pointer to a *function byte*. This byte, which identifies the function of the transmission algorithm, has the following values:

**X'00'**

Open request

**X'80'**

Accept request

**X'40'**

Select request

## Open Request Processing

An open transmission algorithm request occurs when a networking link is started. The transmission algorithm validates the STREAMS and TAPARM values specified on the PARM statement or on the DEFINE or START commands for the link.

The request spool device block (RDEVBLK) contains information that the transmission algorithms use to process these requests. The RDEVSTR field contains the number of streams defined for the link. The RDEVTAP field contains the address of the 80 byte transmission algorithm parameter (TAPARM).

During an open request, the transmission algorithm must validate the TAPARM input string. If it is accepted, the transmission algorithm then converts information in the input string to a data structure that is processed in accept and select requests.

While processing the TAPARM string, the transmission algorithm should build a data structure that describes the contents of the string. This data structure cannot be larger than 80 bytes. To do so, it should use a work area that has been defined in the module containing the transmission algorithm. Before returning control to RSCS, the transmission algorithm should copy this data structure into the storage that was used by the TAPARM string. RSCS then passes this string to the transmission algorithm during accept and select requests.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the function byte X'00' <b>Word 3 (+8)</b> Address of the RDEVBLK
R2 - R12	Not applicable
R13	Save area address
R14	Return address



Register	Contents
R15	Entry address

## Exit Conditions

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## Return Codes

Return Code	Results
0	The transmission algorithm has completed its processing. RSCS copies the values in RDEVSTR and RDEVTAP fields into the LMSSMAX and LMSTAP fields, respectively, of the LINKTABL.
4	The transmission algorithm does not accept the specified TAPARM value; the link driver issues message DMT814E.
8	The value specified for TAPARM is not valid; the link driver issues message DMT815E.
12	The value specified on the STREAMS operand was not valid; the link driver issues message DMT816E.
16	An undefined transmission algorithm was specified for a link; the link driver issues message DMT817E. Transmission algorithms 2 - F are initially configured to issue this return code.

## Accept Request Processing

An accept request occurs when a file is queued on a link; it identifies the stream or streams over which a file may be sent. A transmission algorithm processes accept requests:

- When a networking link is started; all files on the link's queue are processed.
- When individual files are queued on an active networking link.
- For any file that is affected by a file queue reorder on a networking link.

If the REORDER command is issued, all files are processed. If the reorder results from other commands, the minimum number of files necessary to correctly reorder the queues are affected.

For the SHIFT command, all files are processed; transmission algorithms that work with a defined shift manager can change a file's eligibility for various streams when a shift changes. (To prevent files from being sent during certain hours, you can use Exit 31; see [“Exit 31 – Sort Priority Change”](#) on page 103 and [“Example 1: Defining Printing Shifts”](#) on page 18.)

To determine the stream on which the file is processed, the transmission algorithm can use information in the file's TAG element and the CVT. The transmission algorithm receives pointers to these areas when it is called. The TAG element contains information about the number of records in the file, its spool class, priority, origin, and destination. The CVT contains information about the shift setting of the RSCS virtual machine.

The transmission algorithm then determines the streams on which the file can be sent. It returns an identification mask for the selected streams to RSCS in register 1. RSCS uses this mask to determine when it should pass a file to the transmission algorithm during a select request. The following bit values in the mask represent the streams:

Bit Value	Stream
X'80000000'	Stream 1
X'40000000'	Stream 2
X'20000000'	Stream 3
X'10000000'	Stream 4
:	:
X'00000008'	Stream 29
X'00000004'	Stream 30
X'00000002'	Stream 31
X'00000001'	Stream 32

If the file is eligible for two or more streams, the transmission algorithm must return a composite value for *all* applicable stream identifications to RSCS. For example, if streams 1, 3, and 7 match the selection criteria, the transmission algorithm returns the identifier X'A2000000'. If the file is not eligible for any streams, register 1 contains zeros; if you query the status of the file, "no stream" is displayed.

### Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT. <b>Word 2 (+4)</b> Address of the function byte X'80'. <b>Word 3 (+8)</b> Address of the TAG element for the file being queued on the link. The transmission algorithm must not change any fields in the TAG element. <b>Word 4 (+12)</b> Address of the LINKTABL entry, which contains the STREAMS value in LMSSMAX and a pointer to the 80 byte data structure created in the open request in the LMSTAP field. The transmission algorithm must not alter any fields in the LINKTABL.
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

### Exit Conditions

Register	Contents
R0	Not applicable
R1	A hexadecimal identifier for all streams that match the transmission algorithm's selection criteria
R2 - R13	Restored to the same values as on entry

Register	Contents
R14	Not applicable
R15	Return code

## Return Code

Return Code	Results
0	The transmission algorithm has completed its processing; RSCS continues to process the file.

## Select Request Processing

A select request is made when a link requests to select a file for transmission. All files that are queued on the link and are eligible to be sent on idle streams are passed to the transmission algorithm (eligible streams are determined during accept processing). These files are passed in the order they appear on the queue until a file is selected or all files have been checked.

After examining file characteristics (class, number of records, origin address, and destination address), the transmission algorithm may assign the file for transmission on a stream. If this stream is currently active, RSCS ignores the selection and prompts the transmission algorithm for the next suitable file.

After processing a select request, the transmission algorithm must return a single stream identification in register 1. Individual streams are identified by the values used in accept requests for external transmission algorithms (see [“Accept Request Processing”](#) on page 139). The transmission algorithm should *not* return a composite stream identification.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT. <b>Word 2 (+4)</b> Address of the function byte X'40'. <b>Word 3 (+8)</b> Address of the TAG element for the file presented to the transmission algorithm for select processing. The transmission algorithm must not change any TAG fields. <b>Word 4 (+12)</b> Address of the LINKTABL entry, which contains the STREAMS value in LMSSMAX and a pointer to the 80 byte data structure created in the open request in the LMSTAP field. The LMSSACT field contains a mask that describes which streams are currently active on the link. The transmission algorithm must not alter any fields in the LINKTABL.
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

## Exit Conditions

Register	Contents
R0	Not applicable
R1	A hexadecimal value identifying the selected stream
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## Return Code

Return Code	Results
0	The transmission algorithm has completed its processing; RSCS continues to process the file.

## Internal Transmission Algorithms

RSCS provides transmission algorithms 0 and 1 in DMTAXA. You do not need to modify these transmission algorithms or supply additional ones for usual RSCS processing. You can request one of these transmission algorithms by specifying TA=0 or TA=1 on a networking link's DEFINE or START commands or PARM statement.

For compatibility with previous versions of RSCS, DMTAXA also contains 14 reserved entry points (DMTAXAG2 - DMTAXAGF) for additional transmission algorithms. These can be accessed by specifying TA=2 - TA=F.

If you modify transmission algorithm 0 or 1 or create additional internal transmission algorithms, you must reassemble DMTAXA and rebuild the RSCS load library. IBM recommends that you supply any additional transmission algorithms in a load module separate from the RSCS load library. External transmission algorithms can improve the maintenance and portability of your code and reduce the chance of errors.

## Programming Considerations

If you modify or create an internal transmission algorithm in DMTAXA, you should consider these differences:

- The parameter lists for the open, accept and select requests do not include a pointer to the CVT. The parameter lists begin at word 2 of the parameter lists described in [“Open Request Processing” on page 138](#), [“Accept Request Processing” on page 139](#), and [“Select Request Processing” on page 141](#).
- Internal transmission algorithms cannot support 32 streams for GATEWAY-type links.
- The stream masks returned by the accept and select requests in register 1 differ from those specified by external transmission algorithms. The following stream identifiers are applicable only to internal transmission algorithms:

Bit Value	Stream
X'00000080'	Stream 1
X'00000040'	Stream 2
X'00000020'	Stream 3
X'00000010'	Stream 4
X'00000008'	Stream 5

Bit Value	Stream
X'00000004'	Stream 6
X'00000002'	Stream 7

## Transmission Algorithm 0

Transmission algorithm 0 does not place restrictions on the use of the streams for a networking link driver. If a link is connected to a system that does not support multistreaming, you should specify one stream for use with transmission algorithm 0. If the link is started with more than one stream, transmission algorithm 0 makes all files queued on the link eligible for all streams. The order which the files are queued on the link determines the order they are selected for transmission.

### Open Request

For open requests, transmission algorithm 0 validates the STREAMS value to ensure that at most 7 streams are being used on the link. Links that want to use more than 7 streams must use an external transmission algorithm. Transmission algorithm 0 also checks for a TAPARM value and issues the following return codes:

Return Code	Results
0	No TAPARM value was specified.
4	A TAPARM value was specified but not needed.
12	The specified STREAMS value is not valid.

### Accept Request

Transmission algorithm 0 builds a composite mask for the number of streams requested. It then returns this string to RSCS in register 1.

### Select Request

Transmission algorithm 0 scans the LMSSACT field in the LINKTABL to determine the active transmission streams on the link. When it finds an inactive stream, the transmission algorithm sets a value in a register that corresponds to the selected stream.

## Transmission Algorithm 1

Transmission algorithm 1 supports 1 - 7 transmission streams. It selects files by the number of records or the number of spool file blocks in the file. The TAPARM specified with transmission algorithm 1 can specify the upper and lower size limit for the defined streams.

### Open Request

On an open request, transmission algorithm 1 verifies the value specified on the STREAMS parameter. If more than 7 streams are specified, transmission algorithm 1 issues return code 12 in register 15. If the STREAMS value is valid, the transmission algorithm validates the TAPARM values.

If the TAPARM value is not valid, the transmission algorithm issues return code 8. If valid, transmission algorithm 1 places the specified limits for each stream (or the default values) in the TAPARM character string and issues return code 0. Transmission algorithm 1 then uses this TAPARM value as it processes accept and select requests.

Return Code	Results
0	The specified TAPARM value is valid.
8	The specified TAPARM value is not valid.

Return Code	Results
12	The specified STREAMS value is not valid.

The default number of streams is 2 and the default TAPARM is TH=( , 100) , (101 , ). The operational parameters for each networking link describes the format of the TAPARM values for transmission algorithm 1. For more information, see [z/VM: RSCS Networking Operation and Use](#).

### Accept Request

For an accept request, transmission algorithm 1 checks the upper and lower limits for each stream to determine if the file is eligible for transmission on the stream. It then returns a composite mask to RSCS that describes the streams that are eligible to send the file. If the file is ineligible for all defined streams on the link, the transmission algorithm returns zeros in register 1.

### Select Request

For select requests, transmission algorithm 1 checks the size of the file against the upper and lower limits for each defined stream. It then returns the stream identifier of the first idle stream that can send the file in register 1.

## Transmission Algorithms 2 - F

The entry points DMTAXAG2 - DMTAXAGF are provided for compatibility with previous releases of RSCS. These transmission algorithms contain standard linkage code only; no other functions are provided. If you specify TA=2 - TA=F without installing the transmission algorithm at the appropriate entry point, the entry point issues return code 16. The link then issues message DMT817E and deactivates.

## Packaging Transmission Algorithms

Figure 27 on page 144 shows part of an external transmission algorithm, TRANALGX. Like all transmission algorithms, TRANALGX is serially reusable.

```

TRANAL  RMOD
*
      USING CVT,R6          Get CVT addressability
      USING TAG,R7          Get TAG addressability
      USING LINKTABL,R9     Get LINKTABL addressability
TRANALGX RENTRY RENT=NO,SAVEAREA=NO,ARGS=(@CVT,@FUNC)
      L      R6,@CVT        Get CVT address
      L      R2,@FUNC        Get address of function byte
      CLI    0(R2),X'00'     'Open' request?
      BE     TRANOP          Yes ... process it
      CLI    0(R2),X'80'     'Accept' request?
      BE     TRANACC         Yes ... process it

*
*      We've received a select request
*
*
*      REXIT RC=(R15),REGS=(0,12) Return to caller
*
      CVT      DSECT=YES      Communications Vector Table
      LINKTABL DSECT=YES      Link table entry for this link
      TAG      DSECT=YES      Tag element entry
      END

```

Figure 27. Sample External Transmission Algorithm

If your transmission algorithm calls other routines, you should specify SAVAREA=PREALLOC on the RENTRY macro. If the transmission algorithm calls any RSCS routines that are accessible through the CRV, you should add the following to the RMOD macro:

```
CRVCALL=YES,CRVBASE=TCRVTAB
```

You should also place the following statement at the end of the module:

```
CRV DSECT=YES
```

See Chapter 12, “Supported Routines in the CRV,” on page 321 and Chapter 11, “RSCS Macros,” on page 265 for more information.

Next, you create a link-edit control file (see Figure 28 on page 145) for the transmission algorithm.

```
* TRANAL Exit load library load list
%CONTROL RSCSV3
%MAXRC 8
%LIBRARY TRANAL
%ERASE
%LEPARMS NCAL LIST XREF LET NOTERM REUS AMODE 31 RMODE ANY
*
INCLUDE TRANAL
ALIAS TRANALGX
NAME TRANAL
```

Figure 28. Sample LKEDCTRL File: External Transmission Algorithm

## Installing External Transmission Algorithms

To install an external transmission algorithm, like the one shown in Figure 27 on page 144, take the following steps:

1. Issue the following command to assemble the TRANAL ASSEMBLE file:

```
vmfhiasm tranal rscsv3
```

2. Issue the following command to build the PLIMIT LOADLIB:

```
vmflked tranal
```

3. Place the TRANAL LOADLIB on a disk that is accessible to RSCS.
4. Add TRANAL LOADLIB to the GCS GLOBAL command in the RSCS machine's PROFILE GCS; do *not* add another GLOBAL command.
5. Add TA=TRANALGX to the PARM statement for a networking link and add the appropriate TAPARM specification for this transmission algorithm.
6. Restart RSCS.

You can place up to 16 transmission algorithms in a load module and an unlimited number of transmission algorithms in a load library. To do so, add the name of each transmission algorithm's entry point to the ALIAS statement in the link-edit control file.

## Installing Internal Transmission Algorithms

To install a new or modified internal transmission algorithm in DMTAXA, perform the following steps *after* you install RSCS and *before* you start your RSCS virtual machine:

1. Edit the DMTAXA source code at the appropriate entry point to modify or create the required transmission algorithm.
2. Place the modifications on the service disk.
3. Issue the VMFHLASM command to assemble the DMTAXA module and create an updated TEXT file.
4. Specify the transmission algorithm name (TA0 - TAF) on the PARM statement or the START or DEFINE commands for the networking link.
5. Regenerate and link-edit the RSCS LOADLIB.
6. Replace the existing RSCS LOADLIB with the new one.





## Chapter 5. ASCII Printer and Plotter Exit Processing

The ASCII and TCPASCII link drivers use the ASCII printer and plotter exits to build device-specific data streams for transmission to a printer or plotter. With ASCII-type links, the ASCII printers or plotters are connected to RSCS by an IBM 3171 controller. TCPASCII-type links communicate with ASCII printers or plotters that are attached to a terminal server in a TCP/IP network.

ASCII printer and plotter exit routines are called at seven points in the ASCII and TCPASCII link drivers' operational cycle. Each exit routine contains entry points, which provide the following additional processing for each of the seven calls.

Entry Point	Function
Initialization	Called when the link driver is being initialized.
TAG Processing	Called when the link driver opens a new spool file.
Record Processing	Called when a record is read from the input spool file.
Device Reset	Called when a file ends or is flushed, backspaced, or forward-spaced.
Message Processing	Called when a message is received for output to the printer or plotter.
Attention Interrupt Processing	Called when the controller device generates an attention interrupt.
Termination	Called when the link driver is terminating.

The message, attention interrupt, and termination processing routines, are not necessary to support an ASCII or TCPASCII link driver; you do not have to supply these routines.

The termination routine is not necessary to support an ASCII or TCPASCII link driver; you do not have to supply this routine. An exit may require the termination routine for potential clean up processing such as returning any storage obtained in any of the other six exit routines.

The order in which the exit routines are listed above are not necessarily the order in which the ASCII or TCPASCII link driver will call them when processing a print stream.

The exit routine module that contains the exit routines to be used for a specific ASCII-type or TCPASCII-type link must be identified on the PARM configuration statement for the link or in the link operational parameters on the RSCS DEFINE or START command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

### ASCII Exit Programming Considerations

The programming requirements for the ASCII exit routines are described in the following sections.

#### Required Values

The first seven fullwords of each ASCII exit routine module must contain the following values.

**Word 1**

Address of the initialization routine

**Word 2**

Address of the TAG processing routine

**Word 3**

Address of the record processing routine

**Word 4**

Address of the device reset routine

### Word 5

Address of the message processing routine, or 0 if the routine is not provided

### Word 6

Address of the attention interrupt processing routine, or 0 if the routine is not provided

### Word 7

Address of the termination routine, or 0 if the routine is not provided

**Note:** For compatibility with exits written prior to these enhancements, the ASCII or TCPASCII link driver will accept six or seven fullwords of addresses at the beginning of an exit routine module.

## Entry Conditions

When each exit routine receives control, it is passed the address of the CVT and the LINKTABL entry for the link. It may also receive the address of a file's TAG element. The TAG element contains information about a file's characteristics.

The exit routine also receives the address of the print record vector (see "Print Record Vector" on page 148) and the EPARM value. The EPARM value, specified on the PARM configuration statement or on the RSCS DEFINE or START command, contains a parameter string that is associated with the ASCII exit routine.

### Print Record Vector

The ASCII printer and plotter exits use the logical print record vector to pass information between the ASCII or TCPASCII link driver and other ASCII exit routines. However, the attention interrupt processing routine does not receive the print vector record. The "Entry Conditions" section of each exit routine describes the contents of the print record vector.

## Exit Conditions

When an ASCII exit routine returns control to RSCS, the registers contain the following values.

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable

Some exit routines also issue a return code in register 15. If applicable, the return codes are described for the exit routines.

## ASCII Exit Routines

The following sections describe each of the exits supported for ASCII-type and TCPASCII-type links.

### Initialization Routine

This routine initializes the device, including logically switching to the hardcopy device, setting and resetting default states, and setting the *stand by* mode. The exit routine inserts into the print record the characters that RSCS passes to the device to set it into its initial (default) state.

If you specify RENT when link-editing this routine, any storage that will be used by the remaining entry points must be obtained by issuing a GCS GETMAIN macro. The address of this storage must be placed in word 7 of the parameter list so that the other routines can access the work area. In this case it is **required** that a **termination** exit routine issue the GCS FREEMAIN macro to return the storage obtained in the **initialization** exit routine.

The exit routine must specify the length of the data that is passed back to the ASCII or TCPASCII link driver in the data count field of the print record vector. If this routine does not generate data, it should set

the data count field to zero. If the data count is negative or greater than 1280 bytes, the ASCII-type or TCPASCII-type link terminates with user **ABEND 011**.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical print record vector that contains: <b>Bytes 0 - 1</b> Number of bytes in the print record <b>Byte 2</b> Not applicable <b>Bytes 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280) <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 <b>Word 7 (+24)</b> 0
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

## Exit Conditions

On return, the Initialization routine sets the register conditions described in [“Exit Conditions” on page 148](#).

## Return Codes

Return Code	Results
0	Tells RSCS that initialization processing is complete.
4	Tells RSCS that an EPARM value was specified, but the exit routine does not need it; the link terminates.
8	Tells RSCS that the specified EPARM value is not valid; the link terminates.
12	Tells RSCS that the specified EPARM value is valid; this exit routine is called again.
16	Tells RSCS to terminate the ASCII or TCPASCII link driver.

## TAG Processing Routine

This routine 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 device to print the separator into the print record portion of the print record vector.

The exit routine must specify the length of the data that is passed back to the ASCII or TCPASCII link drivers in the data count field of the print record vector. If the exit routine does not generate data, it should set the data count field to zero. If the data count is negative or greater than 1280 bytes, the link terminates with user ABEND 011.

### Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical print record vector that contains: <b>Bytes 0 - 1</b> Number of bytes in the print record <b>Byte 2</b> Not applicable <b>Bytes 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280) <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 <b>Word 7 (+24)</b> Address of the work area established by the initialization routine
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

### Exit Conditions

On return, the TAG processing routine sets the register contents described in [“Exit Conditions” on page 148](#).

### Return Codes

Return Code	Results
0	Tells RSCS that the TAG element processing is complete.

Return Code	Results
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the ASCII or TCPASCII link driver.
12	Tells RSCS to terminate the ASCII or TCPASCII link driver.
16	Tells RSCS to terminate the ASCII or TCPASCII link driver.

## Record Processing Routine

The record processing routine carries out any appropriate translation of the print data. If the exit routine changes the length of the data, the data count field (pointed to by Bytes 0 - 1 in Word 3 of the parameter list) must reflect this change before returning to the link driver. 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 link driver sends the buffer to the ASCII device.

Your exit routine must set the data count field in the print record vector to reflect the length of the data passed to the link driver. If your routine does not send a particular print record, it should set the data count field to zero. A data count that is negative or exceeds 1280 bytes terminates the link with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print record vector that contains:</p> <p><b>Bytes 0 - 1</b> Number of bytes in the print record (does not include the 1-byte CCW in Byte 2)</p> <p><b>Byte 2</b> CCW opcode associated with the print record</p> <p><b>Bytes 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> 0</p> <p><b>Word 7 (+24)</b> Address of the work area established by the initialization routine</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

## Exit Conditions

On return, the Record Processing routine sets the register contents described in [“Exit Conditions” on page 148](#).

## Device Reset Routine

This routine tells RSCS to reset the ASCII device for the next file (for example, feed paper to the top of a new page). The exit routine inserts into the print record portion of the print record vector the characters that RSCS should pass to reset the device.

Your routine must set the data count field in the print record vector to reflect the length of the data that is passed to the link driver. If your routine does not generate any data, it should set the data count field to zero. A data count that is negative or exceeds 1280 bytes terminates the link with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical print record vector that contains: <b>Bytes 0 - 1</b> Number of bytes in the print record <b>Byte 2</b> Not applicable <b>Bytes 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280) <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 <b>Word 7 (+24)</b> Address of the work area established by the initialization routine
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

## Exit Conditions

On return, the Device Reset routine sets the register contents described in [“Exit Conditions” on page 148](#).

## Return Codes

Return Code	Results
0	Tells RSCS that the RESET processing is complete.
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the ASCII or TCPASCII link driver.
12	Tells RSCS to terminate the ASCII or TCPASCII link driver.
16	Tells RSCS to terminate the ASCII or TCPASCII link driver.

## Message Processing Routine

The message processing routine translates EBCDIC code to ASCII code; it is an optional routine. If the exit routine changes the length of the message, the data count field must reflect this change before returning to the link driver. Bytes 0 - 1 in Word 3 of the parameter list point to the data count field. 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 the buffer is full, the link driver sends it to the ASCII device.

Your exit routine must set the data count field in the print record vector to show the length of the data that is being passed back to the link driver. If your routine does not send a message, it should set the data count field to zero. A data count that is negative or exceeds 1280 bytes terminates the link with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical print record vector that contains: <b>Bytes 0 - 1</b> Number of bytes in the print record <b>Byte 2</b> Not applicable <b>Bytes 3 - <i>n</i></b> Message text (where <i>n</i> is a maximum of 1280) <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 <b>Word 7 (+24)</b> Address of the work area established by the initialization routine
R2 - R12	Not applicable
R13	Save area address

Register	Contents
R14	Return address
R15	Entry address

## Exit Conditions

On return, the Message Processing routine sets the register contents described in [“Exit Conditions”](#) on page 148.

## Attention Interrupt Processing Routine

The entry conditions for the Attention Interrupt Processing exit routine vary for ASCII-type links or TCPASCII-type links.

For ASCII-type links, an attention interrupt is generated each time RSCS sends a data buffer. RSCS performs a Read Modified operation each time it receives an attention interrupt. This is required by the 7171 ASCII Device Attachment Control Unit to obtain the Attention Identifier (AID) byte. A null AID (X'E8') is usually returned. If the ASCII device has a keyboard, the attention generating keys (ENTER, PF key, or PA) can change the value of the AID. The exit routine can examine the AID byte and optionally pass a RSCS command back to the link driver for execution.

A TCPASCII-type link does not receive a single AID character. Rather, the exit is passed all data received from the terminal server as it arrives on the link.

If your routine passes a command to RSCS for execution, it must use the *workstation operator* form of the command. The length and address of the command are returned in registers 0 and 1. Setting register 0 to zero means that no command is to be processed and the default PF key commands are to be ignored. The only way to have RSCS use the default PF key commands is *not* to code an attention interrupt processing exit routine. A command length that is negative or exceeds 80 bytes terminates the link with user ABEND 012.

## Entry Conditions

Register	Contents
R0	Not applicable



Register	Contents
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Content varies, depending on the link type: <b>ASCII-type:</b> Address of the Attention Identifier (AID). <b>TCPASCII-type:</b> Address of a 1-byte field containing the length of the received data; the actual data follows. <b>Word 7 (+24)</b> Address of the work area established by the initialization routine
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

## Exit Conditions

On return, this routine sets the register conditions described in [“Exit Conditions” on page 148](#).

## Termination Routine

This exit routine is called just before the ASCII-type or TCPASCII-type link terminates to perform any special termination processing that might be needed. As supplied, RSCS does not provide for any special processing when an ASCII-type or TCPASCII-link is terminated. This exit routine is optional.

### Attention

This exit might be required if any of the other ASCII exit routines (such as initialization) obtain storage from GCS which has not yet been returned to GCS.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 <b>Word 7 (+24)</b> Address of work area established by the initialization routine
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, this routine sets the register conditions described in [“Exit Conditions” on page 148](#).

## Sample ASCII Printer and Plotter Exit Modules

IBM provides samples of the ASCII printer and plotter exits to demonstrate use of typical printer and plotter processing modules. The samples do not provide any processing for the message and attention interrupt routines.

The following sections describe the sample ASCII exit routine modules that are supplied by IBM. These exit routines are provided in the sample load library, RSCSEXIT LOADLIB.

Generally, these sample exit modules respond to options supplied on the CP TAG, SPOOL, or CLOSE commands to perform any needed translations of the print data. These commands are usually specified in the following forms (where *cuu* is the virtual device number or a symbolic name and *option* is the command operand and values):

```
tag dev cuu option
spool cuu option
close cuu option
```

See [z/VM: CP Commands and Utilities Reference](#) for more information.

If you choose to support one of these devices differently or to support a new device, you can modify a sample module or use the samples as a guide for creating your own exit routines.

### Attention

The sample ASCII printer and plotter exit routines are provided for illustrative purposes and on an *as is* basis only. However, you may be able use the samples with little or no modifications, depending on your installation's needs and configuration.

## Printer Exit Modules

The printer modules print separator pages containing the file originator's user ID in block characters and an identifier line. The printer exit modules are:

### **ASCXDSOE**

DS180 Matrix Printer from Datasouth

### **ASCXDWRE**

LA120 DECwriter Printer from DEC

### **ASCXONE**

Generic ASCII printer

### **ASCXPROP**

IBM Proprinter

### **ASCXPSE**

PostScript printer

### **ASCXSPWE**

NEC 3515 Spinwriter Printer

The TAG and SPOOL commands are valid for ASCII printers and produce the following effects on the processing of the printer exit modules. In addition, if a file is sent from an NJE network, the external writer name can be used to specify the *userid* value.

CP Command	Exit Module Processing
<code>tag dev cuu nodeid standard</code>	If STANDARD, options other than those listed below, or no options are specified on a CP TAG command, the spool file is in EBCDIC and contains printable characters. The exit routines translate each print record from EBCDIC to ASCII.
<code>tag dev cuu nodeid aplf</code>	The spool file is in EBCDIC and contains 3270 APL characters. The APL characters are printed, if an APL character set is available.
<code>tag dev cuu nodeid ascii</code>	The spool file is in ASCII. Control characters, such as linefeed and carriage return, are interpreted. No EBCDIC to ASCII translation occurs.
<code>tag dev cuu nodeid asciic</code>	The spool file is in ASCII; no EBCDIC to ASCII translation occurs. Control characters, such as linefeed and carriage return, are <b>not</b> interpreted. The control characters are printed as a character corresponding to a X'2A' (a # in most cases).
<code>tag dev cuu nodeid asciinoc</code>	The spool file is in ASCII; no EBCDIC to ASCII translation occurs. RSCS does not add any control characters.

### **Notes:**

1. The IBM Proprinter and the NEC 3515 Spinwriter Printer do not support the APLF option of the TAG command.
2. The TAG command ASCIINOC option is supported only by the ASCXPROP sample routine.
3. The ASCXONE sample routine supports only the ASCII and ASCIIC options of the TAG command.

The IBM Proprinter requires a carriage return line-feed. However, if you use the ASCXPROP routine with a device that does not require control characters, you should specify the ASCIINOC option on the TAG command. Also, the IBM Proprinter uses a compressed font to print class C files and files with the file type LISTING.

## ASCXPSE Routine

The ASCXPSE exit routine handles one printer queue to a PostScript-only printer. This sample exit routine is similar to the LPRXPSE sample provided for the LPR exits (see [“LPRXPSE Routine” on page 204](#)). This exit routine assumes that the remote printer is PostScript only, or that it will switch into PostScript mode when it receives a %!PS string following an EOT (X'04') character.

### Available EPARM Parameters

When using the ASCXPSE routine, you can specify parameters in the EPARM value on the PARM configuration statement or on the RSCS DEFINE or START command.

#### Notes:

1. Because the EPARM parameter is limited to 239 bytes, these options may be useful only for small amounts of data.
2. Any command received through the CP SMSG facility or the RSCS console will be truncated at 132 bytes.

The following parameters are supported:

#### Sep=

specifies whether a separator page is printed for each file.

##### Yes

Prints a separator page. This is the default. The origin user ID, node ID, and distribution information are printed in large characters; other file information is printed in small characters in Times-Bold font.

##### No

Does not print a separator page.

##### 2p

Produces a two-page separator page. This is useful with duplexing so that the print data starts on a fresh page.

#### Config=*ddname*

specifies the *ddname* which has been defined as an exit configuration file. If the *ddname* does not exist, the ASCXPSE exit will pass back a return code to cause the TCPASCII-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the exit, causing existing defaults to be used for values which can be defined by the configuration file. For more information see [“ASCXPSE Configuration File” on page 160](#).

#### Trailer=

specifies whether a trailer page will be printed.

##### Yes

Prints a trailer page after the file. It is identical to the header page with the addition of a count of the bytes in the file.

##### No

Does not print a trailer page. This is the default.

#### EOT=

specifies whether EOT characters will be inserted.

##### Yes

EOT characters will be inserted after the separator page, data file, and trailer page. This is the default.

##### No

EOT characters will not be inserted.

**Ehandler=**

specifies whether a PostScript error handler will be downloaded to the printer the first time a file is sent to the printer after the link is started. This error handler enables any errors to be printed, so the information will not be lost.

**Yes**

The error handler is downloaded; this is the default.

**No**

The error handler is not downloaded.

**Prefix=hex\_string**

optionally specifies a hexadecimal string to be sent in front of each file; this string is not translated. By default, a prefix string is not sent with each file. Up to 200 bytes of data can be specified.

The prefix string can be split with part sent before the separator page and part sent after. The string will be split if the X'FF04' divider characters are detected within the prefix string. The part before the divider characters will be sent prior to the separator page with the remaining sent after.

**SUFFIX=hex\_string**

optionally specifies a hexadecimal string to be sent after each file; the string is not translated. By default, a suffix string is not sent with each file. Up to 200 bytes of data can be specified.

**Initial=hex\_string**

optionally specifies a hexadecimal string that is sent when the link initializes; the string is not translated. By default, an initial string is not sent. Up to 200 bytes of data can be specified.

**Form=OrFnFsLs**

specifies the default orientation, font name, font size, and additional leading size to use when printing plain text files, overriding the defaults used by the exit. The spool file form name can be used to further override the values specified here.

**Note:** The actual fonts selected must be installed and used by the printer.

The following values can be specified for *OrFnFsLs* (or allowed to default as indicated):

**Or**

is the file orientation:

**PO**

Portrait (default)

**LA**

Landscape

**Fn**

is the font name code:

**CB**

Courier-Bold

**CI**

Courier-Oblique

**CP**

Courier (default)

**CX**

Courier-BoldOblique

**HB**

Helvetica-Bold

**HI**

Helvetica-Oblique

**HP**

Helvetica

<b>HX</b>	Helvetica-BoldOblique
<b>SP</b>	Symbol
<b>TB</b>	Times-Bold
<b>TI</b>	Times-Italic
<b>TP</b>	Times-Roman
<b>TX</b>	Times-BoldItalic

**Fs**  
is the font size, 04 - 99. The default is 11 for portrait and 10 for landscape orientation.

**Ls**  
is the additional leading size, 0.0 - 9.9. This value is added to the font size to give leading, and is specified as 00 - 99. The default is 09 for portrait and 12 for landscape.

**Note:** Any entry not one of the above will cause the default to be used for *Or*, *Fn*, *Fs*, or *Ls*. The value supplied here will be substituted for the form if the form on the spool file does not start with P+, P-, LA, or PO. For more information, see note “2” on page 163.

## ASCPSE Configuration File

The ASCPSE sample routine can read a configuration file. This configuration file supplies these:

- Translation table to override the one used by the exit
- Postscript program to override the one sent to the printer when printing plain text files
- Additional font names used when printing plain text files

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be supplied on the Config= link exit parameter statement when defining the TCPASCII-type link in the RSCS CONFIG file.

An example of a PROFILE GCS DDNAME entry is:

```
'FILEDEF ASCPSE DISK ASCPSE CONFIG *'
```

In this example, ASCPSE is the defined DDNAME and ASCPSE CONFIG is the name of the configuration file.

The following is an example of the RSCS CONFIG parameter (PARM) for a TCPASCII-type link with linkname TCPAP using the DDNAME defined on the FILEDEF statement in the PROFILE GCS:

```
PARM TCPAP EXIT=ASCPSE EPARM='C=ASCPSE'
```

## Layout of the ASCPSE Configuration File

An asterisk (\*) in column one denotes a comment line. Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry. All entries must be capitalized.

The following configuration records are supported.

### **FONT=xxname**

provides a 2-character font name abbreviation (xx) followed by a 32-character font name. There should be no blanks between the abbreviation and full font name. Multiple records can be provided for supplying as many additional fonts as required. The abbreviation should be unique on each FONT= record. In addition, the fonts must be loaded and available at the printer.

The available fonts are:

<b>XX</b>	<i>name</i>
<b>CB</b>	Courier-Bold
<b>CI</b>	Courier-Oblique
<b>CP</b>	Courier (exit default)
<b>CX</b>	Courier-BoldOblique
<b>HB</b>	Helvetica-Bold
<b>HI</b>	Helvetica-Oblique
<b>HP</b>	Helvetica
<b>HX</b>	Helvetica-BoldOblique
<b>SP</b>	Symbol
<b>TB</b>	Times-Bold
<b>TI</b>	Times-Italic
<b>TP</b>	Times-Roman
<b>TX</b>	Times-BoldItalic

#### **PSCRIPT='string'**

provides a replacement postscript program to be used when printing a plain text file. The postscript program must be enclosed within quotes. Anything after the ending quote will be ignored allowing for comments.

For example:

```
PSCRIPT='this is line one'  comment for line one
PSCRIPT='this is line two'
```

Multiple PSCRIPT= records can be provided in order to supply the entire program. ASCXPSE will add a carriage return (X'0A') after each record, and will translate the record from EBCDIC to ASCII.

**Note:** When replacing the postscript program, the ability to tailor the file orientation, font name, font size, and additional leading size through a FORM is lost. The supplied postscript program must define all of these.

#### **TOASCII=string**

provides a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCII= records to replace the 256-byte translation table.

## **Using the FORM Operand of the CP SPOOL Command**

When using the ASCXPSE exit routine, you can also specify the following values on the FORM operand of the SPOOL command.

### **FORM=value**

specifies how the file will be printed:

#### **P-SCRIPT**

PostScript programs.

#### **P+SCRIPT**

PostScript programs (streaming); see note [“2” on page 163](#).

#### **P-ASCII**

PostScript programs in ASCII; see note [“3” on page 163](#).

#### **P+ASCII**

PostScript ASCII (streaming); see note [“4” on page 163](#).

### **OrFnFsLs**

Text file information; if all defaults are used, the value is P0CP1109; see note [“1” on page 163](#).

#### **Or**

is the file orientation:

##### **PO**

Portrait (default)

##### **LA**

Landscape

#### **Fn**

is the font name code:

##### **CB**

Courier-Bold

##### **CI**

Courier-Oblique

##### **CP**

Courier (default)

##### **CX**

Courier-BoldOblique

##### **HB**

Helvetica-Bold

##### **HP**

Helvetica

##### **HI**

Helvetica-Oblique

##### **HX**

Helvetica-BoldOblique

##### **SP**

Symbol

##### **TB**

Times-Bold

##### **TI**

Times-Italic

##### **TP**

Times-Roman

##### **TX**

Times-BoldItalic

#### **Fs**

is the font size, 04 - 99. The default is 11 for portrait and 10 for landscape orientation.



**Ls**

is the additional leading size, 0.0 - 9.9. This value is added to the font size to give leading, and is specified as 00 - 99. The default is 09 for portrait and 12 for landscape.

**Notes:**

1. If the FORM *value* does not start with the characters P0 or LA, the exit routine checks the first record for the string %!PS in EBCDIC and ASCII. If found, it will treat the file as a PostScript file.
2. Streaming means that the spool file is treated as a stream of bytes without regard to record boundaries. Because CP spooling removes trailing blanks, records are padded with blanks up to 80 bytes. Wide PostScript files can be printed by packing them into blocks of 80 bytes, separating the records with linefeeds and punching them to the driver.
3. ASCII means that the data is ASCII and need not be translated before being sent.
4. ASCII streaming is useful for files that were received using the BINARY subcommand of the FTP command. These files should be punched 80 bytes per record to the driver.
5. If an installation requires different strings to be sent to the printer based on the data to print, multiple ASCII links can be defined for the same printer (same host address) with the different desired strings. For example, if the printer can accept five different prefix strings, define five separate links to the same printer, each one with a different prefix string.

## ASCXONE Routine

The ASCXONE sample exit routine performs function for a generic ASCII printer. It also performs simple translation of data to ASCII.

Attention
<p>ASCII translation is controlled by the value specified on the <i>userid</i> operand of the TAG command. The following values for <i>userid</i> have special meaning:</p> <p><b>ASCII</b> The file will not be translated into ASCII. No carriage control will be added by the exit. Any imbedded control characters within the file will be left as is.</p> <p><b>ASCIIIC</b> The file will not be translated into ASCII. Carriage controls will be added after each data record by the exit. Any imbedded control characters within the file will be translated to an ASCII # character.</p> <p>If <i>userid</i> is set to any other value, the file will be translated and carriage control will be performed.</p> <p>If a file is sent from an NJE network, the NJE external writer name can be used to specify the <i>userid</i> value.</p>

## Available EPARM Parameters

When using the ASCXONE routine, you can specify parameters in the EPARM value on the PARM configuration statement or on the RSCS DEFINE or START command.

**Notes:**

1. Because the EPARM parameter is limited to 239 bytes, these options may be useful only for small amounts of data.
2. Any command received through the CP SMSG facility or the RSCS console will be truncated at 132 bytes.

The following parameters are supported:

**Sep=**

specifies whether a separator page will be printed for each file.

**Yes**

Prints a separator page. This is the default. The origin user ID and node ID and distribution information are printed in large characters; other file information is printed in small characters in the Times-Bold font.

**No**

Does not print a separator page.

**2p**

Produces a two-page separator page. This is useful with duplexing so that the print data starts on a fresh page.

**FF=**

specifies how printer form-feeds are performed.

**TOP**

Form-feed is sent in the front of the file.

**Bottom**

Form-feed is sent after the file; this is the default.

**None**

Form-feed is not sent.

**Prefix=hex\_string**

optionally specifies a hexadecimal string to be sent in front of each file; this string is not translated. Up to 200 bytes of data can be specified.

The prefix string can be split with part sent before the separator page and part sent after. The string will be split if the X'FF04' divider characters are detected within the prefix string. The part before the divider characters will be sent prior to the separator page with the remaining sent after.

**SUFFIX=hex\_string**

optionally specifies a hexadecimal string to be sent after each file; the string is not translated. Up to 200 bytes of data can be specified.

**Initial=hex\_string**

optionally specifies a hexadecimal string that is sent when the link initializes; the string is not translated. By default, an initial string is not sent. Up to 200 bytes of data can be specified.

**Config=ddname**

specifies the *ddname* which has been defined as an exit configuration file. If the *ddname* does not exist, the ASCXONE exit will pass back a return code to cause the TCPASCII-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the exit, causing existing defaults to be used for values which can be defined by the configuration file. For more information, see [“ASCXONE Configuration file” on page 165](#).

**CONVERT=No****CONVERT=Yes**

specifies whether to allow protocol conversion. The default is CONVERT=NO. This function has been added for printers moved from coax connected protocol converters to LAN attached, connected to multiprotocol cards for TCP/IP. Some applications use the functions built in to protocol converters.

The ASCXONE exit will look for strings starting with a header defined by the PCL exit parameter and ending with a trailing X'4A' character, which surround pairs of bytes that spell out PCL commands. These pairs of bytes will be packed into single bytes that are the equivalent PCL bytes, which are already in ASCII. Multiple PCL strings can be contained in each record of the print file as long as each string contains the X'4A' trailing character.

In addition, the ASCXONE exit will look for the SCS (SNA Character String) transparency strings contained anywhere within each record. The SCS transparency string is defined with a X'35' followed by a one byte length, followed by data for the defined length that is sent unaltered (not translated). Multiple SCS transparency strings can be contained within a single record.

**PCL=hex\_string**

specifies a 2- to 8-character hexadecimal string which defines the header for PCL strings. When the CONVERT=YES exit parameter is specified, the ASCXONE exit will search for this string in each record of the file to be printed. An even number of characters must be specified; the default is 6A79.

## ASCXONE Configuration file

The ASCXONE sample exit routine can read a configuration file. This configuration file supplies translation tables to override the ones used by the exit.

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be supplied on the Config= link exit parameter statement when defining the TCPASCII-type link in the RSCS CONFIG file.

A example of a PROFILE GCS DDNAME entry is:

```
'FILEDEF ASCXONE DISK ASCXONE CONFIG *'
```

In this example, ASCXONE is the defined DDNAME and ASCXONE CONFIG is the name of the configuration file.

The following is an example of the RSCS CONFIG parameter (PARM) for a TCPASCII-type link with linkname TCPA using the DDNAME defined on the FILEDEF statement in the PROFILE GCS:

```
PARM TCPA EXIT=ASCXONE EPARM='C=ASCXONE'
```

## Layout of the ASCXONE Configuration File

An asterisk (\*) in column one denotes a comment line. Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry. All entries must be capitalized.

These configuration records are supported:

**ASCII=string**

provides a table for translating ASCII control characters, overriding the default used by the exit. ASCXONE uses this translation table when files are already in ASCII and the user ID field of the TAG is set to ASCIIIC. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple ASCII= records to replace the 256-byte translation table.

**TOASCII=string**

provides a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCII= records to replace the 256-byte translation table.

## IBM XY/749 Plotter Exit Module

ASCX749E contains the sample plotter exit module for the IBM Instruments XY/749 Multipen Digital Plotter. The following table shows the valid CP commands for the IBM XY/749 plotter and their effect on plotter exit module processing.

CP Command	Exit Module Processing
tag dev cuu nodeid standard	If STANDARD, options other than those listed below, or no options are specified on a CP TAG command, the spool file is in EBCDIC. The exit routines translate each print record from EBCDIC to ASCII.
tag dev cuu nodeid ascii	The spool file is in ASCII; no EBCDIC to ASCII translation occurs.

CP Command	Exit Module Processing
<code>tag dev cuu nodeid gddmpl</code>	The file is a GDDM® plot file. The first 2 bytes of each record specify its length and are discarded; no EBCDIC to ASCII translation occurs.
<code>spool cuu class a-y, 0-9</code>	An identification line is plotted vertically with pen 1 on the left side of the output medium.
<code>spool cuu class z</code>	An identification line is not plotted.

## Nicolet Zeta 8 Plotter Exit Module

ASCXZETE is the sample exit module for the Nicolet Zeta 8 Plotter. This exit module examines the first ten characters in the first data record of the spool file. If each of these characters is the letter Z, the exit routine translates the first 64 bytes of each print record to ASCII. If not, the file is not translated into ASCII. This exit module also plots an identification line on the output.

## Sending Files with Sample Exit Routines

Use the CMS PRINT and PUNCH commands to send a file to a printer or plotter. The PRINT command accepts up to 204 characters, depending on the virtual printer type. The PUNCH command limits the logical record length of a file to 80 characters. The exit module for the Nicolet Zeta 8 Plotter (ASCXZETE), however, accepts only files sent by the PUNCH command and uses only the first 64 characters of each record. All other exit modules accept both PRINT and PUNCH files.

If no EBCDIC to ASCII translation is performed, each print record is checked to see if the CMS PRINT command produced the header line. If the line is a CMS PRINT header line, FILE: are the first 6 characters of the line; PAGE are the 5 characters starting at position 99. If the previously processed line was the end of a page, the current line is assumed to be a page header. The printer exit modules then translate the line into ASCII. The CMS PRINT command can produce only EBCDIC headers, even if the actual file contains ASCII characters. The IBM XY/749 plotter exit module discards the header line. The ASCXPSE and ASCXONE exit routines do not search for the CMS print header line.

# Chapter 6. Gateway Programming Interface

The gateway programming interface (GPI) lets you create routines that enable RSCS to exchange data between systems using NJE or other networking protocols, such as TCP/IP and Systems Network Architecture Distribution Services (SNA/DS).

In RSCS, GPI support consists of the GATEWAY link driver and *gateway service macros* that provide networking functions for the *gateway program*, which you supply. The system to which the gateway program is communicating must also provide a corresponding routine to communicate with RSCS.

The GATEWAY link driver task initializes the information (data areas, work areas) needed for your gateway program. It then loads and passes control to the gateway program.

The gateway program must start the communication medium it uses for the link. It also issues gateway service macros to communicate with RSCS. When its processing completes or a STOP or DRAIN command is issued for the GATEWAY-type link, the gateway program returns control to RSCS. The GATEWAY link driver task carries out the stopping process, deletes the gateway program, and ends.

The exit routine module that contains the gateway program to be used for a specific GATEWAY-type link must be identified on the PARM configuration statement for the link or in the link operational parameters on the RSCS DEFINE or START command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

## Gateway Program

The gateway program can use the gateway service macros provided by RSCS (see “[Gateway Service Macros](#)” on page 174) to process each record of the NJE data. The gateway program must then convert data between the format required for spool storage and the communication medium used for the link. It must also manage that communication medium. For more information about writing a gateway program, see “[Programming Considerations](#)” on page 169. The GPSAMP sample package contains a sample gateway program called GPI.

### Attention

The GPI sample program is provided for illustrative purposes and on an *as is* basis only. You can modify the sample program or use it as a guide for writing your own gateway program.

## Entry Conditions

After processing the link parameters and necessary RSCS control blocks, the GATEWAY link driver task calls the gateway program with the following register conditions.

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the gateway work area, which contains a vector of routines that enable the gateway service macros to link to RSCS. It also contains three fullwords that are used to store parameter lists for those routines. Each gateway service macro needs this address to function. It is passed to them when the WORK parameter is specified.</p> <p><b>Word 2 (+4)</b> Address of the 80-byte parameter string specified in the EPARM parameter.</p> <p><b>Word 3 (+8)</b> Address of the file-arrival ECB. RSCS posts this ECB when a file is queued for transmission on the GATEWAY-type link.</p> <p><b>Word 4 (+12)</b> Address of the command ECB. RSCS posts this ECB when a command or message NMR element is queued for transmission or processing on the GATEWAY-type link.</p> <p><b>Word 5 (+16)</b> Address of the receiver-online ECB. RSCS posts this ECB when a FREE command is issued and the link is in an <i>input-held</i> state.</p> <p><b>Word 6 (+20)</b> Address of the terminate ECB. When a STOP or SHUTDOWN QUICK command is issued, RSCS posts this ECB to tell the GATEWAY-type link to end as quickly as possible.</p> <p><b>Word 7 (+24)</b> Address of an 8-byte field containing the name of the local RSCS node.</p> <p><b>Word 8 (+28)</b> Address of an 8-byte field containing the name of the GATEWAY-type link.</p> <p><b>Word 9 (+32)</b> Address of the CVT.</p> <p><b>Word 10 (+36)</b> Address of the LINKTABL entry.</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address
R15	Entry address

## Exit Conditions

When the gateway program returns control to RSCS, it should set the following register values.

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## Return Codes

When the gateway program returns to RSCS, it should restore registers in the conventional manner and issue a return code in R15.

Return Code	Meaning
0	Normal ending. Open input files are closed and requeued; open output files are closed and purged. Queued messages and commands are purged.
4	The gateway program did not accept the EPARM parameters passed to it. RSCS issues message DMT819E and ends the link.
8	The gateway program cannot function because of insufficient resources. RSCS issues message DMT708I and ends the link.
12	The gateway program detected a programming error. RSCS issues message DMT824E and ends the link.

## Programming Considerations

This section describes some considerations for writing and implementing gateway programs. [Chapter 2, “Customizing RSCS,”](#) on page 9 contains more information about writing exit routines.

## Work Area Considerations

When a gateway program receives control, it should generate additional save areas. To do so, specify the NUMSAVE parameter on the RENTRY macro for the gateway program (see [“RENTY – Defining a Module Entry Point”](#) on page 290). RSCS automatically acquires any save areas it needs when the gateway service macros are called.

The first time you specify a gateway service macro, you should specify the WORK parameter. This parameter points to the gateway work area that is passed to the gateway program. If you do not move this pointer in the gateway program, you do not have to specify the WORK parameter on any following invocations of a gateway service macro. For example, in [Figure 29 on page 169](#) the work area address in register 10 is used when each gateway service macro is called.

```
X99CAT   RMOD
*
X99CATEP RENTRY RENT=YES,NUMSAVE=10,
              ARGS=(@GWA,@EPARM,@FILEARR,@CMDARR,@TERM,
                  @LOCALID,@LINKID,@CVT,@LINKTAB)
              L   R10,@GWA           Gateway work area pointer
:
:   NJECONCT WORK=(R10)             Indicate we are connected
:
:   NJEOPEN file                    No need for WORK=
:
:   NJECLOSE file                   No need for WORK=
:
:   NJEDSCON                        No need for WORK=
:
END
```

Figure 29. Specifying the WORK Parameter

## Link-Editing Considerations

In most cases, your gateway program should be reentrant. You should specify RENT=YES on the RENTRY macro and the RENT parameter in your VMFLKED link-edit control file options. If your gateway program is nonreentrant, you should also ensure it is nonreusable. Here, do not specify the RENT and REUS link-edit control options. You can specify RENT=YES or RENT=NO on the RENTRY macro. For more information, see [“Link-Editing Considerations”](#) on page 33.

### Program Structure

Figure 30 on page 170 shows the basic structure of a gateway program. During its processing, your gateway program should consider any specified EPARM values and the type of communications used on the link. The gateway program should start its initialization by issuing the NJECONCT macro. Normal ending should be started by issuing the NJEDSCON macro. When terminating, the gateway routine should also consider the requirements of the communications medium.

```
read EPARM text
initialize communications medium
do other initialization
issue NJECONCT to put into connect state

do while not ready to terminate
    wait for work to do
    do the work
end

issue NJEDSCON to return to active state
tidy up communications medium
do other termination
return 0
```

*Figure 30. Basic Structure of Gateway Program*

Your gateway program may need to perform additional functions if errors force the GATEWAY-type link to end. You should, however, ensure that any processing that is started during initialization is completed before the link ends. RSCS system tasks may also perform termination processing.

### Types of Work

As Figure 30 on page 170 shows, the gateway program can contain a loop in which it waits for and carries out various types of work. The type of work the gateway program may perform is described by the following categories.

#### Transmitting Files

Files to be transmitted are called *input* files. Input files must be represented by a file control block that has the NJEINP bit set in the NJESMODE field.

To open these files, the gateway program should issue the NJEOPEN macro. Each record of the file can be obtained by issuing the NJEGET macro. To close the file, the gateway program should issue the NJECLOSE macro. If an error occurs, the gateway program can stop the transmission by issuing the NJERJECT macro.

NJEGET can return records that are up to 32 KB long. These records may contain NJE protocol elements identified by sub record control bytes (SRCBs). See “Supported NJE Sub Record Control Byte Values” on page 172. For SYSOUT files, the records may contain a job header, one or more data set headers, data, and a job trailer. For SYSIN files, the records may contain a job header, data, and a job trailer. Some SYSIN files created by non-VM systems may have Record Characteristics Change dataset headers to show that data other than fixed 80 byte records will follow. The gateway program should pack these record elements into transmission buffers. It should then ensure that the remote system can unpack the records as they are received.

#### Receiving Files

Files to be received are called *output* files because their records are written to spool. Output files must be represented by an output file control block (NJESMODE field should be set to NJEOUTP).

To open these files, the gateway program should issue the NJEOPEN macro. It can obtain each record of the file by issuing the NJEPUT macro. To close the file, the gateway program should issue the NJECLOSE macro. If an error occurs, the gateway program can stop the reception by issuing the NJEABORT macro.



Before NJEOPEN is processed for an output file, the correct SYSIN or SYSOUT flag must be set in the file control block. The gateway program should inspect the first record it receives to ensure that it is a job header from the file.

When the gateway program receives buffers, it must unpack their contents job headers, dataset headers, and data records. It should also identify them using their SRCB. Depending on the type of data received, the gateway program should then issue an NJEOPEN, NJEPUT, or NJECLOSE macro. NJEPUT carries out processing, such as writing data to spool and splitting multiple destination files.

## **Sending Messages and Commands**

Messages and commands are represented by NMR elements, which are sent as records with the SRCB X'00'. To send an NMR element, the gateway program must set up an input file control block for the NMRs (NJESTAT field should be set to NJENMR).

After it initializes, the gateway program should issue the NJEOPEN macro to open the file control block; it should not close this file. When NMR records are available, the gateway program should issue the NJEGET macro and pack each record into a transmission buffer.

## **Receiving Messages and Commands**

The gateway program processes the messages and commands it receives similarly to those it is transmitting. The gateway program should establish a file control block for NMR output and open it after initialization completes. When NMRs are identified in a received buffer, SRCB X'00', the gateway program should issue the NJEPUT macro to write the NMR records to the output file.

RSCS then routes the message or command, which can be processed by Exit 32, to its destination. Messages are delivered to a local user or sent to another node. Commands are run at the local node or sent to another node.

## **Terminating**

The gateway program must respond to error conditions and requests to end the GATEWAY-type link. The gateway program must stop the processing and end or perform appropriate recovery procedures.

## **Scheduling Work**

To schedule work, the gateway program should monitor a list of event control blocks (ECBs). Each ECB represents a specific type of work that a gateway routine must perform. These ECBs are described in the following sections.

### **Terminate ECB**

RSCS posts this ECB in the gateway program when the GATEWAY-type link must end. When posted, the gateway program should end its processing when possible.

### **File Arrival ECB**

RSCS passes this ECB to the gateway program. When posted, the gateway program should issue the NJEOPEN macro to start sending the input file. It should get each record in the file by issuing the NJEGET macro. After it reads the file, sends it to its destination, and receives an acknowledgement from the receiving system, the gateway program should issue the NJECLOSE macro.

After transmitting a file, the gateway program should clear the file arrival ECB and call the NJEOPEN macro again without waiting for this ECB to be posted. If the NJEOPEN macro fails, the gateway program should wait on the ECB and issue NJEOPEN again when it is posted. This ensures that the GATEWAY-type link transmits files when they are queued.

However, you should consider the following factors when creating your gateway program:

- Files may take a long time to send. The entire file is not sent at one time because it may prevent the gateway program from processing messages, commands, files on other streams, incoming traffic, and termination requests.
- On multistreaming links, many files may be transmitting at the same time.
- Many records from different sources may be packed into one transmission buffer, or one record may need several buffers.

### Command ECB

RSCS passes this ECB to the gateway program. It is posted when a message or command must be sent to another node on the link. The gateway routine should clear the ECB and call the NJEGET macro to obtain each record. Your gateway routine should then pack the records into transmission buffers and wait on the command ECB for any new commands or messages. This ensures that commands and messages are transmitted when they are queued.

### Buffer Received ECB

The gateway program chooses the teleprocessing medium it uses to communicate with a remote node. When data arrives in a buffer, GCS drives an interrupt handler in the gateway program. The gateway program must provide and identify this interrupt handler to GCS. The gateway program must then ensure that its buffer received ECB is posted. RSCS does not provide this interrupt handler nor post this ECB for the gateway program.

When it receives a buffer, the gateway program should unpack its contents and process it according to the SRCBs. The gateway program may call the NJEPUT macro to write these records to an appropriate stream. The gateway program may also have to send an acknowledgment, close input files when an acknowledgment is received, or drain the link.

### Receiver-Online ECB

RSCS passes this ECB to the gateway program. This ECB is posted when a FREE command is issued for the link while it is in input-held state. The link may be placed in this state when an authorized operator issues a HOLD *linkid* INPUT command for the link. The RSCS SLOWDOWN facilities may also place the link in this state.

The gateway program must be able to determine when a FREE command has been issued for the link. It must also identify when the link is in input-held state. To do so, the gateway program can check the LHHOLDINP flag in the LINKTABL. It can also determine if the link is in this state if it receives return code 12 after calling the NJEOPEN macro. The NJERJCOD field will contain the reason code X'1404'.

Most gateway programs should also carry out a control flow to tell the remote node when the input flow is held and when it can continue.

### Additional ECBs and Subtasks

Your gateway program may require other ECBs to monitor timeouts on the link or other exception conditions. These ECBs depend on the function of the gateway program.

The gateway program may use subtasks to control various parts of its operation. For example, a subtask in the gateway program can process each input and output transmission stream for the link. Different subtasks within a gateway program can issue gateway service macros at the same time. However, each subtask must specify its own ESTAE and ETXR routines. They should not use the RSCS-defined ESTAE exits.

## Supported NJE Sub Record Control Byte Values

A sub record control byte (SRCB) identifies each record sent or received over an NJE link. The following list identifies the SRCB values that RSCS uses. For more information, 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).

**X'00'**

Used for NMRs.

**X'80'**

Record without carriage control; the data begins immediately after the SRCB.

**X'90'**

Record with machine carriage control. The carriage control opcode byte immediately follows the SRCB; the data follows the carriage control byte.

**X'A0'**

Record with ASA carriage control. The carriage control opcode byte follows the SRCB; the data begins immediately after the carriage control byte.

**X'B0'**

Advanced Function Printing (AFP) record; the SRCB is followed by an X'5A' carriage control byte and the data.

**X'C0'**

Job header, starting with the general section, immediately follows the SRCB.

**X'D0'**

Job trailer, starting with the general section, immediately follows the SRCB.

**X'E0'**

Dataset header, starting with the general section or (rarely) the record characteristics change section, immediately follows the SRCB.

## Reason Code Responses

When the gateway program rejects an NJE stream or file, it should supply a hexadecimal reason code. This code determines the RSCS responses. The following table lists the reason codes and RSCS responses, if applicable (x means any value can appear in that position).

Code	Condition	RSCS Response
04xx	Abort request	Flush and hold current file. The next NJEOPEN request opens a different file.
08xx	Receiving system shutting down	RSCS indicates that the link is draining. The next NJEOPEN request receives return code 4.
0C04	RIF received on unsupported stream	None
0C08	RIF received on drained stream	Flush and requeue current file. The next NJEOPEN call obtains the same file, unless a new file with a higher queuing order arrives.
0C0C	RIF received on an unknown stream	None
0C10	FCS conflict	Flush and requeue current file. The next NJEOPEN call obtains the same file, unless a new file has arrived with a higher queuing order.
1004	Lack of real storage	None
1008	Lack of virtual storage	Flush and requeue current file. The next NJEOPEN call obtains the same file, unless a new file, with a higher queuing order, arrives.
100C	Lack of spool space	Flush and requeue current file. The next NJEOPEN call obtains the same file, unless a new file, with a higher queuing order, arrives.
1010	Lack of processor resources	Flush and requeue current file. The next call to NJEOPEN obtains the same file, unless a new file, with a higher queuing order arrives.

<b>Code</b>	<b>Condition</b>	<b>RSCS Response</b>
1404	HOLD command issued for link	RSCS flushes and requeues the current file. The next NJEOPEN call obtains a different file, unless a new file with a higher queuing order arrives.
1408	STOP command issued for link	RSCS indicates that the link is draining. The next call to NJEOPEN receives return code 4.
140C	FLUSH command issued for link	Flush and hold current file. The next NJEOPEN request opens a different file.
1804	Last transmission ended incorrectly	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.
1808	Compression/compaction error	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.
180C	Records sent are out of sequence	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.
1810	Mixed RCBs in buffer when mixed RCB support not in effect	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.
1814	Undefined RCB/SRCB combination	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.
1Cxx	Data stream error	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.
20xx	File rejected by installation security function or exit routine	RSCS flushes and holds current file. The next NJEOPEN call obtains a different file.

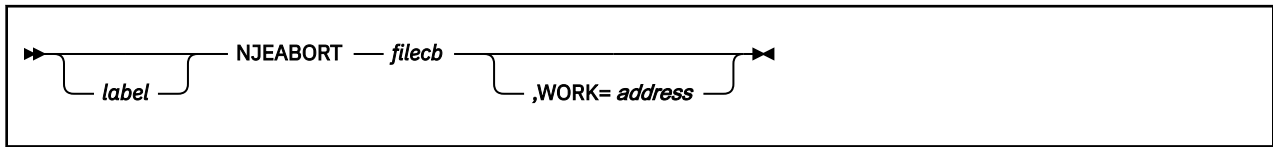
## Gateway Service Macros

Gateway programs use the gateway service macro supplied by IBM to run specific gateway functions. The WORK parameter, which points to the gateway work area, must be specified on at least the first invocation of a gateway service macro within a gateway program. If the WORK parameter is not specified, later invocations of the macros use the value defined on the latest specification of the parameter.

The macros issue a return code in R15 to show the success of the operation or a reason for its error. Sometimes, R0 and R1 return additional information. Generally, however, you should rely only on registers 2 - 13 being preserved when a gateway service macro is called.

The following sections describe each gateway service macro. For information about the macro notational conventions, see [“Specifying Parameters”](#) on page 265. For information about the conventions used in the syntax diagrams, see [“Syntax, Message, and Response Conventions”](#) on page xv.

## NJEABORT



### Purpose

The NJEABORT macro halts operations on an output (receiving) file. The file is closed and purged from spool.

### Parameters

#### *label*

is an optional assembler label.

#### *filecb*

is the address of the NJE file control block for the output NJE file. This value may be specified as an RX-type address or as register (2) - (12).

#### **,WORK=address**

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

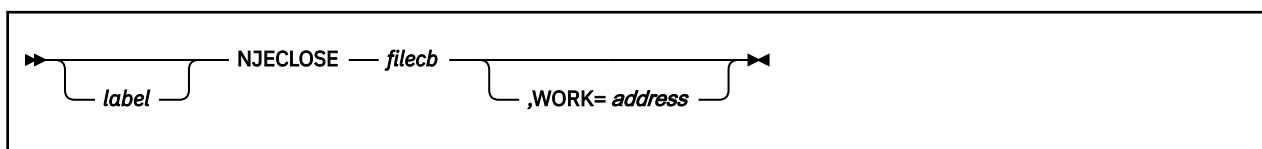
### Usage Notes

The gateway program does not have to issue an NJECLOSE macro.

### Return Codes

Return Code	Meaning
0	File transmission halted.
8	File not open or open for input only. The file control block had not been successfully opened by an NJEOPEN macro, or it had been opened but for output; the request is ignored.

## NJECLOSE



### Purpose

The NJECLOSE macro closes a file previously opened with the NJEOPEN macro. Input files are considered to have been sent successfully on the link and are purged from spool. All output files are considered to have been successfully received. They are closed and spooled to their destination or to RSCS for store-and-forward processing.

### Parameters

#### *label*

is an optional assembler label.

#### *filecb*

is the address of the file's NJE file control block. This value may be specified as an RX-type address or as register (2) - (12).

#### *,WORK=address*

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

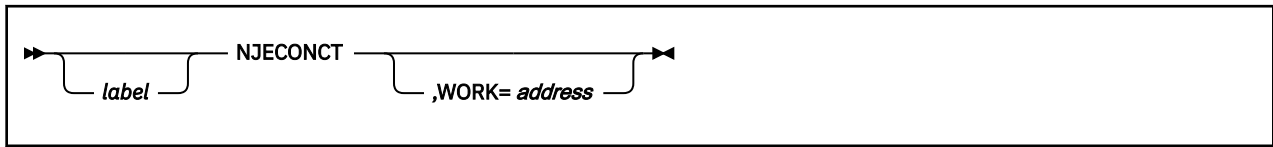
### Usage Notes

1. Your gateway program should not issue NJECLOSE for NMR streams.
2. Your gateway program should not issue NJECLOSE until it receives an acknowledgment that the remote system has received the file. Otherwise, the file may be lost.
3. To stop transmission of an input file, use the NJERJECT macro rather than NJECLOSE. The file will be held in the RSCS virtual reader. RSCS may purge the file or re-enqueue it for transmission. To stop reception of an output file, use the NJEABORT macro to stop reception; the file will then be purged from spool.

### Return Codes

Return Code	Meaning
0	The file has been successfully closed.
8	The file was not open.
12	An internal error was detected. The file control block is marked as closed, but RSCS may not have successfully closed the spool file or some other error may have occurred.

## NJECONCT



### Purpose

The NJECONCT macro shows that the gateway program has initialized. After this macro is run, RSCS marks the GATEWAY-type link as connect.

### Parameters

#### *label*

is an optional assembler label.

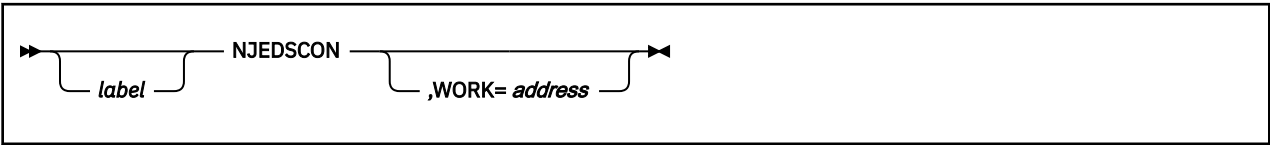
#### **,WORK=address**

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

### Results

Return Code	Meaning
0	The gateway program has initialized and the GATEWAY-type link is connected.

# NJEDSCON



## Purpose

The NJEDSCON macro shows that the GATEWAY-type link is no longer connected and cannot process any files, messages, or commands. RSCS closes any files that are active on the link and changes the link's status from connect to active.

## Parameters

***label***

is an optional assembler label.

***,WORK=address***

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

## Usage Notes

The gateway program should issue this macro before it returns control to RSCS to deactivate the link. If the gateway program returns control without issuing NJEDSCON, however, RSCS automatically issues the macro.

## Return Codes

Return Code	Meaning
0	The GATEWAY-type link is disconnected.



## NJEGET



### Purpose

The NJEGET macro gets a record to be transmitted in a file or gets an NMR element to run or send.

### Parameters

#### *label*

is an optional assembler label.

#### *filecb*

is the address of the file's NJE file control block. This value may be specified as an RX-type address or as register (2) - (12).

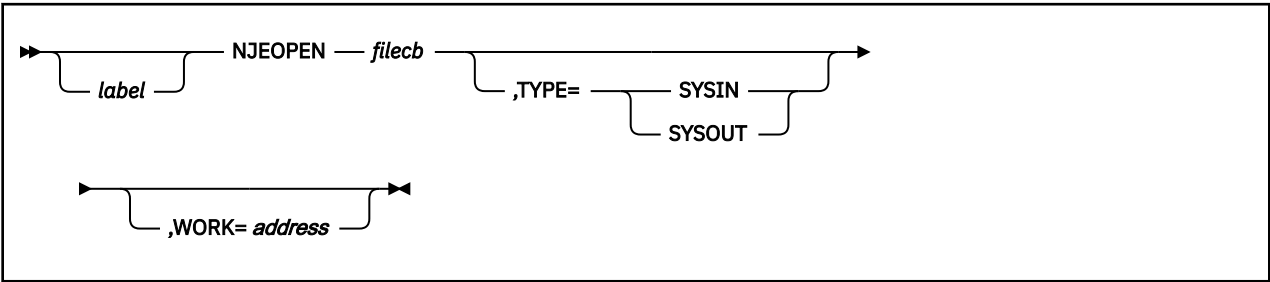
#### *,WORK=address*

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

### Return Codes

Return Code	Meaning
0	Record successfully read. R1 points to the record; R0 contains its length.
4	End of file. No record has been read. For files, the gateway program should finish transmitting the file and issue an NJECLOSE macro. For the input NMR stream, the gateway program should stop trying to get NMR records and perform other work or WAIT on the NMR ECB.
8	File not open. The specified NJE file control block had not been successfully opened; the request is ignored.
12	An internal error occurred; RSCS cannot supply another record. This occurs only on file streams; the file is closed and NJERJCOD has been set to show the error. NJERJECT should not be issued by the gateway program in response to this return code; the NJEFILE has already been closed internally. It should also notify the receiving system that the file transmission is ending prematurely.

# NJEOPEN



## Purpose

The NJEOPEN macro opens a file for input or output (SYSIN or SYSOUT).

## Parameters

### *label*

is an optional assembler label.

### *filecb*

is the address of the NJE file control block for the NJE file to be opened. This value may be specified as an RX-type address or as register (2) - (12).

### *,TYPE=*

specifies the type of file (SYSIN or SYSOUT) created by the gateway program when the *filecb* is MODE=OUTPUT on the NJEFILE macro. If not specified, the value already in the NJESTYPE field of the *filecb* is used. This field may be set by:

- Specifying the TYPE parameter on the *filecb* (if not reset)
- The TYPE value specified on a previously-processed NJEOPEN macro
- An NJESTYPE value set by the gateway program

### *,WORK=address*

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

## Usage Notes

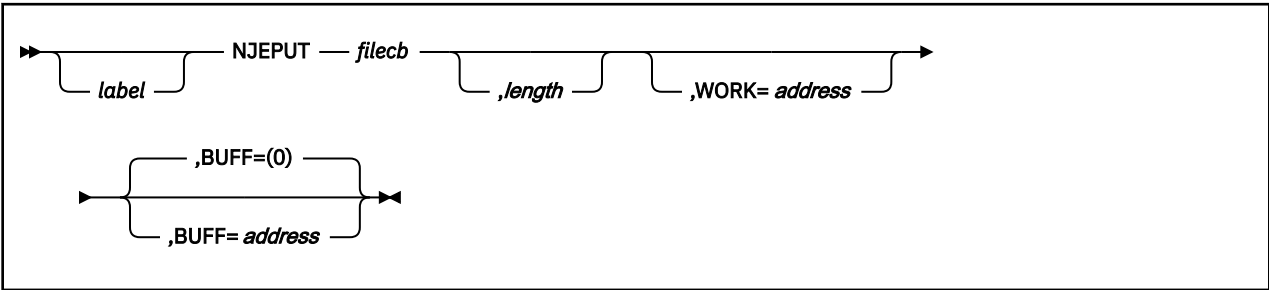
1. After a file is opened for input (transmission), the NJESYIN or NJESYOUT flag in the NJE file control block specifies the type of file that has been opened. For output files, however, the gateway program must set the appropriate bit in the NJE file control block to specify the type of file to be opened. The gateway program should issue the NJEOPEN macro to begin every file to be transmitted or received. It should issue the NJECLOSE macro to end files (unless they are stopped or rejected, whereas the NJEABORT or NJERJECT macro should be used).
2. Commands and messages are treated as records in an *NMR stream*. NMR streams are treated like files that are open all the time. After it initializes, your gateway program should issue NJEOPEN to open an NJE file control block to receive NMRs and an NJE file control block to send NMRs. It does not need to issue NJECLOSE to close the streams.

## Return Codes

Return Code	Meaning
0	The file was opened successfully.

Return Code	Meaning
4	No input files are available to be opened. This can occur when there are no files in the queue. It can also occur when the transmission algorithm or a query of the GATEWAY-type link show that no queued files are eligible for transmission. The gateway program should perform other work or wait on the file-arrival ECB.
8	<p>Too many open files. For input and output NMR streams, the maximum number of files is one. Your gateway program should ensure that this return code is not issued for NMR streams.</p> <p>Up to 32 streams can be used for input streams, depending on the value of the STREAMS parameter. If the gateway program receives this return code when trying to open an input file, the number of currently active streams specifies the stream limit. The gateway program should not try to open files beyond its stream limit again.</p> <p>There is no limit on the number of output files that can be open; this return code should not be issued now.</p>
12	Internal error occurred; the file was not opened. The NJERJCOD field in the file control block contains the reason code.

# NJEPUT



## Purpose

The NJEPUT macro writes a record that has been received to spool. You can also use NJEPUT to pass a received NMR element to RSCS for sending, execution (for commands), or delivery (for messages).

## Parameters

### *label*

is an optional assembler label.

### *filecb*

is the address of the file's NJE file control block. This value may be specified as an RX-type address or as register (2) - (12).

### **,BUFF=(0)**

### **,BUFF=address**

specifies the address of the buffer containing the record to be written. It may be specified as an RX-type address, as register (0), or as registers (2) - (12). If not specified, the address is assumed to be in R0.

### **,length**

is the length of the buffer containing the record to be written. The length may be specified as an assembler program label or constant, or as register (2) - (12). If not specified, the buffer length is assumed to be specified in field NJERCLN in the NJE file control block.

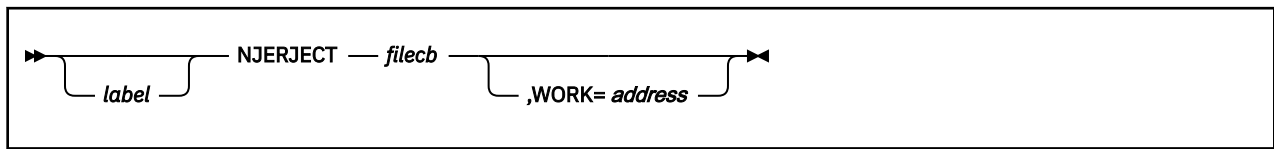
### **,WORK=address**

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may be specified as register (2) - (12). It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

## Return Codes

Return Code	Meaning
0	Record successfully written.
8	The NJE file control block had not been successfully opened by a previous NJEOPEN macro; the request is ignored.
12	An internal error occurred (for example, RSCS could not write the file to spool); the NJERJCOD field contains the reason code. The gateway program should issue NJEABORT to close the file and purge it from spool. It should also supply a reason code to tell the remote system to end the transmission.

## NJERJECT



### Purpose

The NJERJECT macro stops processing of an input (transmitting) file.

### Parameters

#### *label*

is an optional assembler label.

#### *filecb*

is the address of the file's NJE file control block. This value may be specified as an RX-type address or as register (2) - (12).

#### **,WORK=address**

specifies the address of the NJE work area that was originally passed to the gateway program as word 1 of the R1 parameter list. It may also be specified as a fullword where the work area address has been stored. If not specified, the *address* specified on the WORK parameter for the previous NJE macro is used.

### Usage Notes

Before you run this macro, ensure that the NJERJCOD field in the NJE file control block contains the reason the file was rejected. Often, this is a reason code sent by the receiving system. Depending on the reason code, RSCS will either:

- Purge the file from spool
- Hold the file
- Queue on the link again for later transmission

### Return Codes

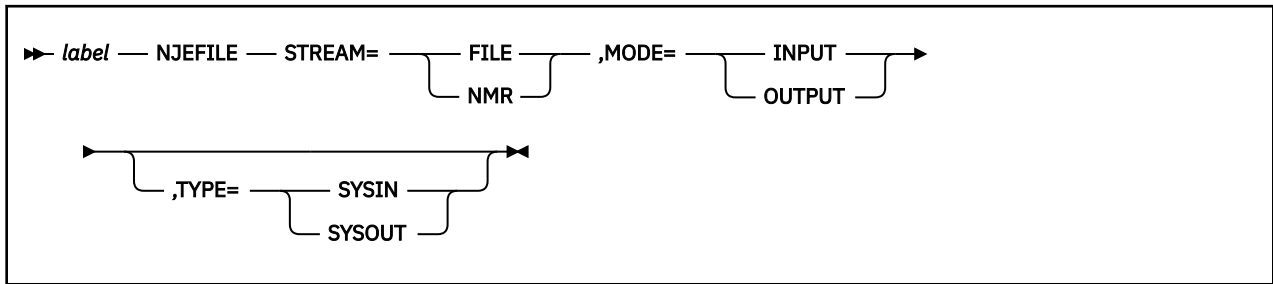
Return Code	Meaning
0	File is rejected. The input file is closed; it will be purged, held, or queued.
8	File not open. The file was not successfully opened by a previous NJEOPEN macro or was opened for output; the request is ignored.

## NJE File Control Block Fields

Your gateway program may examine or modify any of the following fields in the NJE file control block.

Field Name	Length/ Value	Examine/ Modify	Function
NJERCLEN	2 bytes	Modify	After an NJEGET function, it contains the length of the record read from spool. Before an NJEPUT function, it contains the length of the record to be written.
NJERJCOD	2 bytes	Modify	Contains a reason code indicating why a file is rejected (see “Reason Code Responses” on page 173). When NJEOPEN or NJEPUT issue return code 12 for an input file, this field shows the reason the file was rejected. The gateway program can examine this field to determine its recovery action.  The gateway program can also specify a reason code in this field before issuing the NJERJECT macro to reject a file.
NJESTAT	1 byte	Examine	Contains status flags that describe the state of the file. The following fields list the status flags accessible to the gateway program.
NJEOPN	X'80'	Examine	The NJE file control block has been opened by an NJEOPEN function.
NJEFIL	X'40'	Examine	The NJE file control block is for a file stream.
NJENMR	X'20'	Examine	The NJE file control block is for an NMR stream.
NJESTYPE	1 byte	Modify	Shows if the file-type stream is SYSIN or SYSOUT. This field should be set before NJEOPEN is issued for a MODE=OUTPUT NJE file control block. The following bits are defined in this field.
NJESYIN	X'80'	Modify	The NJE file control block is processing a SYSIN file.
NJESYOUT	X'40'	Modify	The NJE file control block is processing a SYSOUT file.
NJESMODE	1 byte	Modify	Shows if a file is INPUT or OUTPUT. This field should be set before NJEOPEN is issued for a file control block. The following bits are defined in this field.
NJEINP	X'80'	Modify	This is an input file (for transmission). Only the NJEGET and NJERJECT macros can be issued between invocations of NJEOPEN and NJECLOSE.
NJEOUTP	X'40'	Modify	This is an output file (being received). Only NJEPUT and NJEABORT macros can be issued between invocations of NJEOPEN and NJECLOSE.

## NJEFILE



### Purpose

The NJEFILE macro creates an NJE file control block, which the gateway routine can use to process an NJE stream. An NJE file can represent a file or a series of NMRs. Files are described as SYSIN or SYSOUT; it can be input (transmitting) or output (receiving).

### Parameters

#### *label*

is the assembler label that references the NJE file control block. You must specify a unique label on each NJEFILE macro.

#### **STREAM=**

specifies the type of stream (FILE or NMR) associated with the NJE file control block.

#### **MODE=**

specifies the type of processing the gateway program may perform on the file.

#### **INPUT**

The gateway program may perform NJEOPEN and NJECLOSE functions on the file. Between these operations, it may perform only NJEGET and NJERJECT functions.

#### **OUTPUT**

The gateway program may perform NJEOPEN and NJECLOSE functions on the file. Between these operations, it may perform only NJEPUT and NJEABORT functions.

#### **TYPE=**

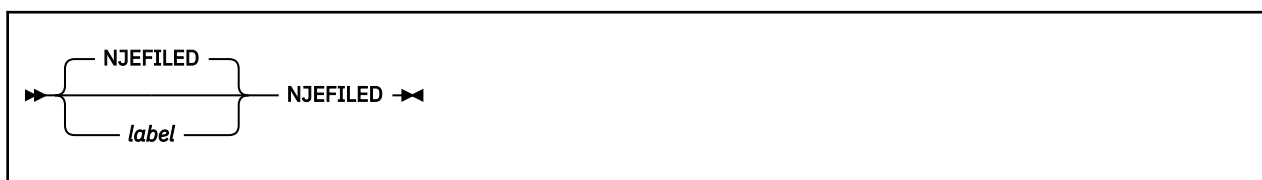
specifies the type of data (SYSIN or SYSOUT) processed on the stream when STREAM=FILE and MODE=OUTPUT are also specified.

### Usage Notes

The NJEFILE macro expands to produce DC instructions to define a file control block, which can be mapped with the NJEFILED macro. If your gateway program is reentrant, it should copy the model file descriptions, generated by NJEFILE, into a work area for the appropriate stream.

## NJEFIELD

---



### Purpose

The NJEFIELD macro creates a DSECT that maps the NJE file control blocks generated by the NJEFILE macro.

### Parameters

#### *label*

is the name used for the mapping DSECT. The default is NJEFIELD.



---

## Chapter 7. TCP/IP LPR Exit Points

The TCP/IP LPR link driver uses exits to build device-specific data streams for transmission and to control the remote host and port to which the transmission is destined. Exit routines are called at six points in the LPR link driver's operational cycle.

Each LPR exit routine contains entry points, which provide the following additional processing for each of the six calls.

Entry Point	Function
Initialization	Called when the 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 when the LPR link driver is terminating.

The termination routine is not necessary to support an LPR link driver; you do not have to supply this routine. An exit may require the termination routine for potential clean up processing such as returning any storage obtained in any of the other five exit routines.

The order in which the exit routines are listed above is not necessarily the order in which the LPR link driver will call them when processing a print stream.

The LPR exit routine module that contains the exit routines to be used for a specific LPR-type link must be identified on the PARM configuration statement for the link or in the link operational parameters on the RSCS DEFINE or START command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

---

### LPR Programming Considerations

The programming requirements for the LPR exit routines are described in the following sections.

#### Required Values

The first six fullwords of each LPR exit routine module must contain the following values:

**Word 1**

Address of the initialization routine

**Word 2**

Address of the TAG processing routine

**Word 3**

Address of the record processing routine

**Word 4**

Address of the end of file processing routine

**Word 5**

Address of the control file processing routine

**Word 6**

Address of the termination routine, or 0 if the routine is not provided

**Note:** For compatibility with exits written prior to these enhancements, the LPR link driver will accept 5 or 6 fullwords of addresses at the beginning of an exit routine module.

## Entry Conditions

When an LPR exit routine receives control, it may be passed the following information:

- Address of the CVT
- Address of the LINKTABL entry
- Remote host IP address
- Remote host name
- Remote host port
- Remote printer queue name
- Printer flag fields
- User-defined filter
- User-defined prefix string
- User-defined suffix string
- User-defined separator page setting
- User-defined translate table

All exit routines, except the LPR initialization and termination routines, also receive a pointer to the TAG element. The TAG element contains information about a file's characteristics. The exit routine also receives the EPARM value for the link and the address of the print record vector. The EPARM value, specified on the PARM configuration statement or on the RSCS DEFINE or START command, contains a parameter string that is associated with the LPR exit routine.

## Printer Flag Fields

Each LPR exit routine is passed the address of the following printer flag fields.

*Table 4. Printer Flag Fields and Values*

Bit	Field Name	Values
0	PASS=	<b>0</b> One pass is performed on the file; RSCS sends the file directly to the line printer daemon.
		<b>1</b> Two passes are performed on the file. On the first pass (CURRENT_PASS=1), RSCS first counts the number of bytes in the file; no data within the file is sent. On the second pass, RSCS then sends the file on to the line printer daemon.
1	CTLIST=	<b>0</b> Data is sent first.
		<b>1</b> Control file is sent first.
2	CURRENT_PASS=	<b>0</b> This is the second pass through the file.
		<b>1</b> This is the first pass through the file.

## Print Record Vector

The LPR exits use the logical print record vector to pass a data stream to a link driver exit for conversion from EBCDIC to ASCII or binary. The logical print record vector is also used to pass a control file between the exit routine and the link driver. The "Entry Conditions" section for each exit routine describes the contents of the print record vector.

## Exit Conditions

When an LPR exit routine returns control to RSCS, the registers contain these values:

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## LPR Exit Routines

The following sections describe each of the exits supported for LPR-type links. For more information, see ["Required Values" on page 187](#).

### LPR Initialization Routine

This exit routine initializes the LPR-type link. It can also customize where files are to be printed in the TCP/IP network by changing TCP/IP-specific information that was defined by the link PARM statement.

The exit routine is passed an address to these areas, which contain information about the line printer daemon:

- Remote host IP address
- Remote host port
- Remote printer queue name
- Link driver flag fields (see [Table 4 on page 188](#))
- Host name
- User-defined prefix string (length followed by data)
- User-defined suffix string (length followed by data)
- User-defined filter
- User-defined translation table
- User-defined separator page setting

If you specify RENT when link-editing this routine, any storage that will be used by the remaining entry points must be obtained by issuing a GCS GETMAIN macro. The address of this storage must be placed in word 6 of the parameter list so that the other routines can access the work area. In this case, it is **required** that a **termination** exit routine issue the GCS FREEMAIN macro to return the storage obtained in the **initialization** exit routine.

### Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> 0</p> <p><b>Word 4 (+12)</b> 0</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> 0 on entry; on return, it may contain the address of the work area for the LPR exit routines</p> <p><b>Word 7 (+24)</b> Address of a fullword containing the remote host IP address</p> <p><b>Word 8 (+28)</b> Address of a fullword containing the remote host port</p> <p><b>Word 9 (+32)</b> Address of the remote printer queue name</p> <p><b>Word 10 (+36)</b> Address of the LPR link driver flags (see <a href="#">Table 4 on page 188</a>)</p> <p><b>Word 11 (+40)</b> Address of a 255-character host name</p> <p><b>Word 12 (+44)</b> Address of a fullword containing the length of a user-defined prefix string followed by the 250-byte prefix string</p> <p><b>Word 13 (+48)</b> Address of a fullword containing the length of a user-defined suffix string followed by the 250-byte suffix string</p> <p><b>Word 14 (+52)</b> Address of a 1-character user-defined filter</p> <p><b>Word 15 (+56)</b> Address of a 256-character user-defined translate table</p> <p><b>Word 16 (+60)</b> Address of a 4-character user-defined separator page setting</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPR Initialization routine sets the register conditions described in [“Exit Conditions” on page 189](#).

## Return Codes

Return Code	Results
0	Tells RSCS that initialization processing is complete
4	Tells RSCS that an EPARM value was specified, but the exit routine does not need it; the LPR link driver terminates.
8	Tells RSCS that a specified EPARM value was not valid; the LPR link driver terminates.
12	Tells RSCS to terminate the LPR link driver.
16	Tells RSCS to terminate the LPR link driver.

## LPR TAG Processing Routine

This exit routine 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 into the print record portion of the print record vector. This exit may be called twice if doing two passes on the file.

The LPR TAG processing exit can also be used to customize where individual files are to be printed in a TCP/IP network. Your exit routine can change TCP/IP-specific information that was defined by the PARM statement for the LPR-type link. The exit routine is passed an address to the following information, which can be modified for the line printer daemon information:

- Remote host IP address
- Remote host port
- Remote printer queue name
- Link driver flag fields (see [Table 4 on page 188](#))
- Host name
- User-defined prefix string (length followed by data)
- User-defined suffix string (length followed by data)
- User-defined filter
- User-defined translation table
- User-defined separator page setting

The exit routine must specify the length of the data that is passed back to the LPR link driver in the data count field of the print record vector. If the exit routine does not generate data, it should set the data count field to zero. If the data count is negative or greater than 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a fullword containing the remote host IP address</p> <p><b>Word 8 (+28)</b> Address of a fullword containing the remote host port</p> <p><b>Word 9 (+32)</b> Address of the remote printer queue name</p> <p><b>Word 10 (+36)</b> Address of the LPR link driver flags (see <a href="#">Table 4 on page 188</a>)</p> <p><b>Word 11 (+40)</b> Address of a 255-character host name</p> <p><b>Word 12 (+44)</b> Address of a fullword containing the length of a user-defined prefix string followed by the 250-byte prefix string</p> <p><b>Word 13 (+48)</b> Address of a fullword containing the length of a user-defined suffix string followed by the 250-byte suffix string</p> <p><b>Word 14 (+52)</b> Address of a 1-character user-defined filter</p> <p><b>Word 15 (+56)</b> Address of a 256-character user-defined translate table</p> <p><b>Word 16 (+60)</b> Address of a 4-character user-defined separator page setting</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPR Tag processing routine sets the register conditions described in [“Exit Conditions” on page 189](#).

## Return Codes

Return Code	Results
0	Tells RSCS that the TAG element processing is complete.
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the LPR link driver.
12	Tells RSCS to terminate the LPR link driver.
16	Tells RSCS to terminate the LPR link driver.

## LPR Record Processing Routine

The record processing routine may translate the print data, for example from EBCDIC to ASCII. If the exit routine changes the length of the data, the data count field (pointed to by Bytes 0 - 1 in Word 3 of the parameter list) must reflect this change before returning to the link driver. 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 link driver sends the buffer to the TCP/IP line printer daemon. This exit is called for each record of the spool file being sent to a TCP/IP line printer daemon.

Your exit routine must set the data count field in the print record vector to reflect the length of the data passed to the link driver. If your exit routine does not send a particular print record, it should set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a fullword containing the remote host IP address</p> <p><b>Word 8 (+28)</b> Address of a fullword containing the remote host port</p> <p><b>Word 9 (+32)</b> Address of the remote printer queue name</p> <p><b>Word 10 (+36)</b> Address of the LPR link driver flags (see <a href="#">Table 4 on page 188</a>)</p> <p><b>Word 11 (+40)</b> Address of a 255-character host name</p> <p><b>Word 12 (+44)</b> Address of a fullword containing the length of a user-defined prefix string followed by the 250-byte prefix string</p> <p><b>Word 13 (+48)</b> Address of a fullword containing the length of a user-defined suffix string followed by the 250-byte suffix string</p> <p><b>Word 14 (+52)</b> Address of a 1-character user-defined filter</p> <p><b>Word 15 (+56)</b> Address of a 256-character user-defined translate table</p> <p><b>Word 16 (+60)</b> Address of a 4-character user-defined separator page setting</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address



## Exit Conditions

On return, the LPR record processing routine sets the register conditions described in [“Exit Conditions” on page 189](#).

## Return Codes

Return Code	Results
0	Tells RSCS that the spool record element processing is complete.
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the LPR link driver.
12	Tells RSCS to terminate the LPR link driver.
16	Tells RSCS to terminate the LPR link driver.

## LPR End of File Routine

This exit routine allows for additional information to be sent to the TCP/IP line printer daemon. It is called after the last spool file record has been processed. This enables any specific device-dependent information (for example, feed paper to the top of a new page) to be transmitted.

Your exit routine must set the data count field in the print record vector to reflect the length of the data that is passed to the link driver. If your exit routine does not generate any data, it should set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a fullword containing the remote host IP address</p> <p><b>Word 8 (+28)</b> Address of a fullword containing the remote host port</p> <p><b>Word 9 (+32)</b> Address of the remote printer queue name</p> <p><b>Word 10 (+36)</b> Address of the LPR link driver flags (see <a href="#">Table 4 on page 188</a>)</p> <p><b>Word 11 (+40)</b> Address of a 255-character host name</p> <p><b>Word 12 (+44)</b> Address of a fullword containing the length of a user-defined prefix string followed by the 250-byte prefix string</p> <p><b>Word 13 (+48)</b> Address of a fullword containing the length of a user-defined suffix string followed by the 250-byte suffix string</p> <p><b>Word 14 (+52)</b> Address of a 1-character user-defined filter</p> <p><b>Word 15 (+56)</b> Address of a 256-character user-defined translate table</p> <p><b>Word 16 (+60)</b> Address of a 4-character user-defined separator page setting</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPR end of file processing routine sets the register conditions described in [“Exit Conditions”](#) on page 189.

## Return Codes

Return Code	Results
0	Tells RSCS that the end of file processing is complete.
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the LPR link driver.
12	Tells RSCS to terminate the LPR link driver.
16	Tells RSCS to terminate the LPR link driver.

## LPR Control File Routine

This exit routine creates control file information that is sent to the TCP/IP line printer daemon. It is entered once for each spool file that is selected for transmission; this enables the exit routine to specify any device-dependent information.

Your exit routine must set the data count field in the print record vector to reflect the length of the data that is passed to the link driver. If your exit routine does not generate any data, it should set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

On entry, bytes 0 and 1, which are pointed to by the address of word 3, contain the length of the data file name. This is followed by the data file name, which has this format:

- X'DFA' (prefix)
- 3-byte sequence number
- File origin node name, which may be up to 8 bytes long

This name is from the HOSTDAFN LPR parm statement or the file's TAG and should be used in control file commands, where appropriate.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the control file record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Control file record data (where <i>n</i> is a maximum of 1280); on entry, this is the length of the data file name</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a fullword containing the remote host IP address</p> <p><b>Word 8 (+28)</b> Address of a fullword containing the remote host port</p> <p><b>Word 9 (+32)</b> Address of the remote printer queue name</p> <p><b>Word 10 (+36)</b> Address of the LPR link driver flags (see <a href="#">Table 4 on page 188</a>)</p> <p><b>Word 11 (+40)</b> Address of a 255-character host name</p> <p><b>Word 12 (+44)</b> Address of a fullword containing the length of a user-defined prefix string followed by the 250-byte prefix string</p> <p><b>Word 13 (+48)</b> Address of a fullword containing the length of a user-defined suffix string followed by the 250-byte suffix string</p> <p><b>Word 14 (+52)</b> Address of a 1-character user-defined filter</p> <p><b>Word 15 (+56)</b> Address of a 256-character user-defined translate table</p> <p><b>Word 16 (+60)</b> Address of a 4-character user-defined separator page setting</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPR control file processing routine sets the register conditions described in [“Exit Conditions”](#) on page 189.

## Return Codes

Return Code	Results
0	Tells RSCS that the control file processing is complete.
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the LPR link driver.
12	Tells RSCS to terminate the LPR link driver.
16	Tells RSCS to terminate the LPR link driver.

## LPR Termination Routine

This exit routine is called just before the LPR-type link terminates to perform any special termination processing that might be needed. As supplied, RSCS does not provide for any special processing when an LPR link is terminated. This exit routine is optional.

### Attention

This exit might be required if any of the other LPR exit routines (such as initialization) obtain storage from GCS which has not yet been returned to GCS.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> 0</p> <p><b>Word 4 (+12)</b> 0</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> 0</p> <p><b>Word 8 (+28)</b> Address of a fullword containing remote host port</p> <p><b>Word 9 (+32)</b> Address of the remote printer queue name</p> <p><b>Word 10 (+36)</b> Address of the LPR link driver flags (see <a href="#">Table 4 on page 188</a>)</p> <p><b>Word 11 (+40)</b> Address of a 255-character host name</p> <p><b>Word 12 (+44)</b> Address of a fullword containing the length of a user-defined prefix string followed by the 250-byte prefix string</p> <p><b>Word 13 (+48)</b> Address of a fullword containing the length of a user-defined suffix string followed by the 250-byte suffix string</p> <p><b>Word 14 (+52)</b> Address of a 1-character user-defined filter</p> <p><b>Word 15 (+56)</b> Address of a 256-character user-defined translate table</p> <p><b>Word 16 (+60)</b> Address of a 4-character user-defined separator page setting</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPR Termination routine sets the register conditions described in [“Exit Conditions” on page 189](#).

## Sample LPR Exit Routines

The following sections describe the sample LPR exit routine modules supplied by IBM. These exit routine modules are provided in the sample load library, RSCSEXIT LOADLIB.

Generally, the sample exit routines respond to options supplied on the CP TAG, SPOOL, or CLOSE commands to perform any needed translations of the print data. These commands are usually specified in the following forms (*cuu* is the virtual device number or a symbolic name and *option* is the command operand and values):

```
tag dev cuu option
spool cuu option
close cuu option
```

#### Attention

The sample LPR exit routines are provided for illustrative purposes and on an *as is* basis only. However, you may be able use the sample exits with little or no modifications, depending on your installation's needs and configuration.

If you choose to support the LPR link drivers differently or to support additional devices, you can modify a sample module or use the sample as a guide for creating your own exit routines.

## LPRXONE Routine

The LPRXONE sample exit routine performs functions for one printer queue. It also performs simple translation of data to ASCII.

#### Attention

ASCII translation is controlled by the value specified on the *userid* operand of the TAG command. The following values for *userid* have special meaning:

##### ASCII

The file will not be translated into ASCII. No carriage control will be added by the exit. Any imbedded control characters within the file will be left as is.

##### ASCIIC

The file will not be translated into ASCII. Carriage controls will be added after each data record by the exit. Any imbedded control characters within the file will be translated to an ASCII # character.

If *userid* is set to any other value, the file will be translated and carriage control will be performed.

If a file is sent from an NJE network, the NJE external writer name can be used to specify the *userid* value.

## Available EPARM Parameters

When using the LPRXONE routine, you can specify parameters in the EPARM value on the PARM configuration statement or on the RSCS DEFINE or START command.

#### Notes:

1. Because the EPARM field is limited to 239 bytes, these parameters may be useful only for small amounts of data.
2. Any command received through the CP SMSG facility or the RSCS console will be truncated at 132 bytes.
3. EPARM parameters must be separated by one or more blanks. They can be specified in any order.

The following parameters are supported:

#### Config=*ddname*

specifies the DDNAME which has been defined as an exit configuration file. If the DDNAME does not exist, the LPRXONE initialization routine will pass back a return code to cause the LPR-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the LPRXONE exit, causing existing defaults to be used for values which can be defined by the configuration file. For more information, see [“LPRXONE Configuration file” on page 203](#).

**CONVert=No****CONVert=Yes**

specifies whether to allow protocol conversion. The default is NO. This function has been added for printers moved from coax connected protocol converters to LAN attached, connected to multiprotocol cards for TCP/IP. Some applications use the functions built in to protocol converters.

The LPRXONE exit will look for strings starting with a header defined by the PCL exit parameter and ending with a trailing X'4A' character, which surround pairs of bytes that spell out PCL commands. These pairs of bytes will be packed into single bytes that are the equivalent PCL bytes, which are already in ASCII. Multiple PCL strings can be contained in each record of the print file as long as each string contains the X'4A' trailing character.

In addition, the LPRXONE exit will look for the SNA Character String (SCS) transparency strings contained anywhere within each record. The SCS transparency string is defined with a X'35' followed by a 1-byte length, followed by data for the defined length that is sent unaltered (not translated). Multiple SCS transparency strings can be contained within a single record.

**FF=**

specifies how printer form-feeds are performed.

**TOP**

Form-feed is sent in the front of the file.

**Bottom**

Form-feed is sent after the file; this is the default.

**None**

Form-feed is not sent.

**FILter=filter**

specifies the printer filter used in the control file sent to the line printer daemon. One EBCDIC character is passed; if it is uppercase alphabetic, it will be translated to lowercase. The default is f.

**Form=value**

specifies the default form sent to the printer when one has not been supplied when the file was spooled to RSCS.

**PCL=hex\_string**

specifies a 2- to 8-character hexadecimal string which defines the header for PCL strings. When the CONVERT=YES exit parameter is specified, the LPRXONE exit will search for this string in each record of the file to be printed. An even number of characters must be specified; the default is 6A79.

**Prefix=hex\_string**

specifies an optional hexadecimal string to be sent in front of each file; this string is not translated. Up to 200 bytes of data can be specified. By default, a prefix string is not sent with each file.

The prefix string can be split with part sent before the separator page and part sent after. The string will be split if the X'FF04' divider characters are detected within the prefix string. The part before the divider characters will be sent prior to the separator page with the remaining sent after.

**Sep=**

specifies whether a separator page will be printed for each file.

**Yes**

Prints a separator page; this is the default. The origin user ID and node ID and distribution information are printed in large characters; other file information is printed in small characters in the Times-Bold font.

**No**

Does not print a separator page

**Host**

An L control file record is sent to request that the host produce the separator page.

**2p**

Produces a two-page separator page. This is useful with duplexing so that the print data starts on a fresh page.



**SUFFIX=hex\_string**

specifies an optional hexadecimal string to be sent after each file; the string is not translated. Up to 200 bytes of data can be specified. By default, a suffix string is not sent with each file.

## LPRXONE Configuration file

The LPRXONE sample exit routine can read a configuration file. This configuration file can supply the following:

- Translation tables to override the ones used by the exit
- Overrides for the control file created by LPRXONE

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be specified on the Config= parameter in the EPARM value on the PARM statement for the link in the RSCS CONFIG file or in the link operational parameters on the RSCS DEFINE or START command.

The following is an example of a DDNAME entry in the PROFILE GCS in which LPRONE is the defined DDNAME and LPR CONFIG is the name of the configuration file:

```
'FILEDEF LPRONE DISK LPR CONFIG *'
```

The following is an example of the PARM statement for an LPR-type link named LPR using the DDNAME defined on the FILEDEF statement in the PROFILE GCS:

```
PARM LPR EXIT=LPRXONE EPARM='C=LPRONE'
```

## Layout of the LPRXONE Configuration File

The following rules apply to the LPRXONE configuration file:

- An asterisk (\*) in column one denotes a comment line.
- Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry.
- All configuration entries must be capitalized.

The following configuration records are supported:

**ASCII=string**

specifies a table for translating ASCII control characters, overriding the default used by the exit. LPRXONE uses this translation table when files are already in ASCII and the user ID field of the TAG is set to ASCII. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple ASCII= records to replace the 256-byte translation table.

**DOMAINNAME=string**

specifies a domain name, up to 255 characters, to be appended after the host name of the H control file record. A period (.) will be inserted between the host name and domain name. This record can be used to add a domain name after the host name, which by default is the node name where the file originated or as specified in the HOSTNAME= record.

**HOSTNAME=string**

specifies a host name, up to 255 characters, for the H control file record, overriding the default, which is the node name where the file originated.

**TOASCII=string**

specifies a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCII= records to replace the 256-byte translation table.

**TOASCIIIC=string**

specifies a table for EBCDIC to ASCII translation of the LPR control file, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCIIIC= records to replace the 256-byte translation table.

**USERNAME=*string***

specifies a user name, up to 32 characters, for the P control file record, overriding the default name used by the exit, which is the user name of the file originator. This record can be used to cause all error messages to be sent to a central location.

## LPRXPSE Routine

The LPRXPSE exit routine handles one printer queue to a PostScript-only printer. This exit routine assumes that the remote printer is PostScript only, or that it will switch into PostScript mode when it receives a %!PS string following an EOT (X'04') character.

### Available EPARM Parameters

When using the LPRXPSE routine, you can specify parameters in the EPARM value on the PARM configuration statement or on the RSCS DEFINE or START command.

**Notes:**

1. Because the EPARM field is limited to 239 bytes, these parameters may be useful only for small amounts of data.
2. Any command received through the CP SMSG facility or the RSCS console will be truncated at 132 bytes.
3. EPARM parameters must be separated by one or more blanks. They can be specified in any order.

The following parameters are supported:

**Config=*ddname***

specifies the DDNAME which has been defined as an exit configuration file. If the DDNAME does not exist, the LPRXPSE initialization routine will pass back a return code to cause the LPR-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the LPRXPSE exit, causing existing defaults to be used for values which can be defined by the configuration file. For more information, see [“LPRXPSE Configuration file” on page 206](#).

**Ehandler=**

specifies whether a PostScript error handler will be downloaded to the printer the first time a file is sent to the printer after the link is started. This error handler enables any errors to be printed, so the information will not be lost.

**Yes**

The error handler is downloaded; this is the default.

**No**

The error handler is not downloaded.

**EOT=**

specifies whether EOT characters will be inserted.

**Yes**

EOT characters will be inserted after the separator page, data file, and trailer page; this is the default.

**No**

EOT characters will not be inserted.

**FIter=*filter***

specifies the printer filter used in the control file sent to the line printer daemon. One EBCDIC character is passed; if it is uppercase alphabetic, it will be translated to lowercase. The default is f.

**Form=*OrFnFsLs***

specifies the default orientation, font name, font size, and additional leading size to use when printing plain text files, overriding the defaults used by the exit. The spool file form name can be used to further override the values specified here.

**Note:** The actual fonts selected must be installed and used by the printer.

The following values can be specified for *OrFnFsLs* (or allowed to default as indicated):

***Or***

is the file orientation:

**PO**

Portrait (default)

**LA**

Landscape

***Fn***

is the font name code:

**CB**

Courier-Bold

**CI**

Courier-Oblique

**CP**

Courier (default)

**CX**

Courier-BoldOblique

**HB**

Helvetica-Bold

**HI**

Helvetica-Oblique

**HP**

Helvetica

**HX**

Helvetica-BoldOblique

**SP**

Symbol

**TB**

Times-Bold

**TI**

Times-Italic

**TP**

Times-Roman

**TX**

Times-BoldItalic

***Fs***

is the font size, 04 - 99. The default is 11 for portrait and 10 for landscape orientation.

***Ls***

is the additional leading size, 0.0 - 9.9. This value is added to the font size to give leading, and is specified as 00 - 99. The default is 09 for portrait and 12 for landscape.

**Note:** Any entry not one of the above will cause the default to be used for *Or*, *Fn*, *Fs*, and *Ls*. The value supplied here will be substituted for the form if the form on the spool file does not start with P+, P-, LA, or PO.

**Prefix=hex\_string**

specifies an optional hexadecimal string to be sent in front of each file; this string is not translated. Up to 200 bytes of data can be specified. By default, a prefix string is not sent with each file.

The prefix string can be split with part sent before the separator page and part sent after. The string will be split if the X'FF04' divider characters are detected within the prefix string. The part before the divider characters will be sent prior to the separator page with the remaining sent after.

**Sep=**

specifies whether a separator page will be printed for each file.

**Yes**

Prints a separator page; this is the default. The origin user ID, node ID, and distribution information are printed in large characters; other file information is printed in small characters in the Times-Bold font.

**No**

Does not print a separator page.

**Host**

Sends an L control file record to request that the host produce the separator page.

**2p**

Produces a two-page separator page. This is useful with duplexing so that the print data starts on a fresh page.

**SUFFIX=hex\_string**

specifies an optional hexadecimal string to be sent after each file; the string is not translated. Up to 200 bytes of data can be specified. By default, a suffix string is not sent with each file. See note “5” on [page 163](#).

**Trailer=**

specifies whether a trailer page will be printed.

**Yes**

Prints a trailer page after the file. It is identical to the header page, with the addition of a count of the bytes in the file.

**No**

Does not print a trailer page; this is the default.

## LPRXPSE Configuration file

The LPRXPSE sample exit routine can read a configuration file. This configuration file can supply the following:

- Translation table to override the one used by the exit
- Postscript program to override the one sent to the printer when printing plain text files
- Additional font names used when printing plain text files
- Overrides for the control file created by LPRXPSE

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be specified on the Config= parameter in the EPARM value on the PARM statement for the link in the RSCS CONFIG file or in the link operational parameters on the RSCS DEFINE or START command.

The following is an example of a DDNAME entry in the PROFILE GCS in which LPRPSE is the defined DDNAME and LPR CONFIG is the name of the configuration file:

```
'FILEDEF LPRPSE DISK LPR CONFIG *'
```

The following is an example of the PARM statement for an LPR-type link named LPR using the DDNAME defined on the FILEDEF statement in the PROFILE GCS:

```
PARM LPR EXIT=LPRXPSE EPARM='C=LPRPSE'
```

## Layout of the LPRXPSE Configuration File

The following rules apply to the LPRXPSE configuration file:

- An asterisk (\*) in column one denotes a comment line.
- Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry.

- All configuration entries must be capitalized.

The following configuration records are supported:

**DOMAINNAME=string**

specifies a domain name, up to 255 characters, to be appended after the host name of the H control file record. A period (.) will be inserted between the host name and domain name. This record can be used to add a domain name after the host name, which by default is the node name where the file originated or as specified in the HOSTNAME= record.

**FONT=xxname**

specifies a 2-character font name code (xx) followed by a 32-character font name. There should be no blanks between the code and the full font name. Multiple records can be provided for supplying as many additional fonts as required. The font name code should be unique on each FONT= record. In addition, the fonts must be loaded and available at the printer.

The available fonts are:

**CB**

Courier-Bold

**CI**

Courier-Oblique

**CP**

Courier (exit default)

**CX**

Courier-BoldOblique

**HB**

Helvetica-Bold

**HI**

Helvetica-Oblique

**HP**

Helvetica

**HX**

Helvetica-BoldOblique

**SP**

Symbol

**TB**

Times-Bold

**TI**

Times-Italic

**TP**

Times-Roman

**TX**

Times-BoldItalic

**HOSTNAME=string**

specifies a host name, up to 255 characters, for the H control file record, overriding the default, which is the node name where the file originated.

**PSCRIPT='string'**

specifies a replacement PostScript program to be used when printing a plain text file. The PostScript program must be enclosed within quotes. Anything after the ending quote will be ignored allowing for comments. For example:

```
PSCRIPT='this is line one'  comment for line one
PSCRIPT='this is line two'
```

Multiple PSCRIPT= records can be provided in order to supply the entire program. LPRXPSE will add a carriage return (X'0A') after each record, and will translate the record from EBCDIC to ASCII.

**Note:** When replacing the PostScript program, the ability to tailor the file orientation, font name, font size, and additional leading size through a FORM is lost. The supplied PostScript program must define all of these.

**TOASCII=string**

specifies a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCII= records to replace the 256-byte translation table.

**TOASCIIIC=string**

specifies a table for EBCDIC to ASCII translation of the LPR control file, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCIIIC= records to replace the 256-byte translation table.

**USERNAME=string**

specifies a user name, up to 32 characters, for the P control file record, overriding the default name used by the exit, which is the user name of the file originator. This record can be used to cause all error messages to be sent to a central location.

## Using the FORM Operand of the CP SPOOL Command

When using the LPRXPSE exit routine, you can also specify the following values on the FORM operand of the SPOOL command.

**FORM=value**

specifies how the file will be printed:

**P-SCRIPT**

PostScript programs.

**P+SCRIPT**

PostScript programs (streaming); see note [“2” on page 209](#).

**P-ASCII**

PostScript programs in ASCII; see note [“3” on page 209](#).

**P+ASCII**

PostScript ASCII (streaming); see note [“4” on page 209](#).

**OrFnFsLs**

Text file information; if all defaults are used, the value is POCP1109; see note [“1” on page 209](#).

**Or**

is the file orientation:

**PO**

Portrait (default)

**LA**

Landscape

**Fn**

is the font name code:

**CB**

Courier-Bold

**CI**

Courier-Oblique

**CP**

Courier (default)

**CX**

Courier-BoldOblique

<b>HB</b>	Helvetica-Bold
<b>HP</b>	Helvetica
<b>HI</b>	Helvetica-Oblique
<b>HX</b>	Helvetica-BoldOblique
<b>SP</b>	Symbol
<b>TB</b>	Times-Bold
<b>TI</b>	Times-Italic
<b>TP</b>	Times-Roman
<b>TX</b>	Times-BoldItalic

***F*s** is the font size, 04 - 99. The default is 11 for portrait and 10 for landscape orientation.

***L*s** is the additional leading size, 0.0 - 9.9. This value is added to the font size to give leading, and is specified as 00 - 99. The default is 09 for portrait and 12 for landscape.

#### Notes:

1. If the FORM *value* does not start with the characters P0 or LA, the exit routine checks the first record for the string %!PS in EBCDIC and ASCII. If found, it will treat the file as a PostScript file.
2. Streaming means that the spool file is treated as a stream of bytes without regard to record boundaries. Because CP spooling removes trailing blanks, records are padded with blanks up to 80 bytes. Wide PostScript files can be printed by packing them into blocks of 80 bytes, separating the records with linefeeds and punching them to the driver.
3. ASCII means that the data is ASCII and need not be translated before being sent.
4. ASCII streaming is useful for files that were received using the BINARY subcommand of the FTP command. These files should be punched 80 bytes per record to the driver.





---

## Chapter 8. TCP/IP LPD Exit Points

The LPD link drivers use exits to build specific data streams for acceptance into the RSCS network and to control the remote node and user ID to which the print data is destined. Exit routines are called at seven points in the LPD link driver's operational cycle.

Each LPD exit routine contains entry points, which provide the following additional processing for each of the seven calls:

Entry Point	Function
Initialization	Called when the LPD link driver is being initialized.
Print Command Processing	Called when the LPD link driver receives a print command from an LPR client.
Print Job 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 an LPR client for the LPD link driver.
End of File Processing	Called when a file has been completely read from an LPR client for the LPD link driver.
Control File Processing	Called for each line of a control file read from an LPR client for the LPD link driver.
Termination	Called when the LPD link driver is terminating.

The termination routine is not necessary to support an LPD link driver; you do not have to supply this routine. An exit may require the termination routine for potential clean up processing such as returning any storage obtained in any of the other six exit routines.

The order in which the exit routines are listed above is not necessarily the order in which the LPD link driver will call them when processing a print stream from an LPR client.

The LPD exit routine module that contains the exit routines to be used for a specific LPD-type link must be identified on the PARM configuration statement for the link or in the link operational parameters on the RSCS DEFINE or START command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

---

### LPD Programming Considerations

The programming requirements for the LPD exit routines are described in the following sections.

#### Required Values

The first seven fullwords of each LPD exit routine module must contain these values:

**Word 1**

Address of the initialization routine

**Word 2**

Address of the print command processing routine

**Word 3**

Address of the print job command processing routine

**Word 4**

Address of the data processing routine

**Word 5**

Address of the end of file processing routine

**Word 6**

Address of the control file processing routine

**Word 7**

Address of the termination routine, or 0 if the routine is not provided

## Entry Conditions

When an LPD exit routine receives control, it may be passed the following information:

- Address of the CVT
- Address of the LINKTABL entry
- Address of a SOCKADDR structure for the remote host

This is a 16-byte structure containing the addressing family, port number, and IP address of the line printer remote host. This information is **not passed** to the initialization or termination routines.

All exit routines, except the initialization and termination routines, also receive a pointer to the TAG element. The TAG element contains information about a file's characteristics. The exit routine also receives the EPARM value for the link and the address of the print record vector. The EPARM value, specified on the PARM configuration statement or on the RSCS DEFINE or START command, contains a parameter string that is associated with the LPD exit routine.

## Order of the Control File and Data File

It is recommended that the control file be sent by the LPR client in order for the LPD driver and exits to work in as simple and seamless a fashion as possible. Otherwise, the exit routines will have to make assumptions about the data prior to obtaining the file attributes from the control file.

## Response Messages

All exit routines, except the initialization and termination routines, can send a response message to the TCP/IP LPR client. In most cases this will be in the form of a negative acknowledgement. The sending of appropriate positive acknowledgments is the responsibility of RSCS and not any of the exit routines.

RFC 1179 requires acknowledgments to be returned to an LPR client when a line printer daemon receives:

- A printer job command
- Control file command
- Data associated with the control file
- Data file command
- Data associated with the data file

A positive acknowledgement is a single byte containing all zero bits. A negative acknowledgement is one or more bytes with the first byte containing a nonzero bit pattern (this could be in the form of an error message).

A return code of 4 from the Print Command and Print Job Command exit routines can be used to send a message to the TCP/IP LPR client and have the exit routine called again. A return code of 8 from all exit routines, except the initialization and termination routines, can be used to send a response message to the TCP/IP LPR client, close and purge the spool file if already created, and close the connection.

## Print Record Vector

The LPD exits use the logical print record vector to pass a data stream to a link driver exit for conversion from ASCII to EBCDIC. The logical print record vector is also used to pass a control file between the exit

routine and the link driver. The "Entry Conditions" section for each exit routine describes the contents of the print record vector.

## Exit Conditions

When an LPD exit routine returns control to RSCS, the registers contain these values:

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## LPD Exit Routines

The following sections describe each of the exits supported for LPD-type links. For more information, see ["Required Values" on page 211](#).

### LPD Initialization Routine

This exit routine initializes the LPD-type link. The exit is not passed any link options and therefore cannot change the TCP/IP port the link is listening on during this exit routine processing.

If you specify RENT when link-editing this routine, any storage that will be used by the remaining entry points must be obtained by issuing a GCS GETMAIN macro during this exit routine processing. The address of this storage must be placed in word 6 of the parameter list so that the other routines can access the work area. In this case, it is **required** that a **termination** exit routine issue the GCS FREEMAIN macro to return the storage obtained in the **initialization** exit routine.

### Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 on entry; on return, it may contain the address of the work area for the LPD exit routines <b>Word 7 (+24)</b> 0
R2 - R12	Not applicable
R13	Save area address

Register	Contents
R14	Return address

## Exit Conditions

On return, the LPD Initialization routine sets the register conditions described in [“Exit Conditions” on page 213](#).

## Return Codes

Return Code	Results
0	Tells RSCS that initialization processing is complete.
4	Tells RSCS that an EPARM value was specified, but the exit routine does not need it; the LPD link driver terminates.
8	Tells RSCS that a specified EPARM value was not valid; the LPD link driver terminates.
12	Tells RSCS to terminate the LPD link driver.
16	Tells RSCS to terminate the LPD link driver.

## LPD Print Command Processing Routine

This exit routine is called when a print command is received from a TCP/IP LPR client. The command received is contained within the print record vector and is in ASCII format. This exit routine carries out any appropriate data translation from ASCII to EBCDIC, returning a command ready to be processed by RSCS or a response message in ASCII to be sent to the TCP/IP LPR client.

This routine is also responsible for filling in the file's TAG text fields based on information received from the LPR client. The exit routine inserts the TAG characteristics into the TAG text fields portion of the parameter list.

The possible print commands that can be received, and the response from RSCS when this exit routine completes with return code 0, are:

### Print any waiting jobs

RSCS will send a positive acknowledgment to the LPR client and close the connection.

### Receive a printer job

RSCS will use the QUEUE name returned in the print record vector when issuing message DMTLPD214I and continue receiving data from the LPR client.

### Send queue state (short and long)

RSCS will send a negative acknowledgment to the LPR client in the form of an *Unsupported print command* message and close the connection.

### Remove jobs

RSCS will send a negative acknowledgment to the LPR client in the form of an *Unsupported print command* message and close the connection.

This exit routine must set the data count field in the print record vector to reflect the length of the data passed back to the LPD link driver. If this exit routine does not generate data, it **must** set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Print command data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPD TAG processing routine sets the register conditions described in [“Exit Conditions” on page 213](#).

## Return Codes

Return Code	Results
0	Tells RSCS to process the translated command contained within the print record vector.
4	Tells RSCS to send the response message contained within the print record vector to the TCP/IP LPR client and call this exit routine again.
8	Tells RSCS to send the response message contained within the print record vector to the TCP/IP LPR client, flush the spool file if created, and close the connection.

Return Code	Results
12	Tells RSCS to terminate the LPD link driver.
16	Tells RSCS to terminate the LPD link driver.

## LPD Print Job Command Processing Routine

This exit routine is called when a print job command is received from a TCP/IP LPR client. The command received is contained within the print record vector and is in ASCII format. This exit routine carries out any appropriate data translation from ASCII to EBCDIC, returning a command ready to be processed by RSCS or a response message in ASCII to be sent to the TCP/IP LPR client.

The possible print job commands that can be received, and the response from RSCS when this exit routine completes with return code 0, are:

### Abort job

RSCS will send a positive acknowledgment to the LPR client and close the connection.

### Receive control file

RSCS will receive the control file from the LPR client.

### Receive data file

RSCS will receive the data file from the LPR client.

This exit routine must set the data count field in the print record vector to reflect the length of the data that is passed back to the LPD link driver. If this exit routine does not generate data, it **must** set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Print Job command data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPD record processing routine sets the register conditions described in [“Exit Conditions” on page 213](#).

## Return Codes

Return Code	Results
0	Tells RSCS to process the translated command contained within the print record vector.
4	Tells RSCS to send the response message contained within the print record vector to the TCP/IP LPR client and call this routine again.
8	Tells RSCS to send the response message contained within the print record vector to the TCP/IP LPR client, flush the spool file if created, and close the connection.

Return Code	Results
12	Tells RSCS to terminate the LPD link driver.
16	Tells RSCS to terminate the LPD link driver.

## LPD Data Processing Routine

The record processing routine carries out appropriate translation from ASCII to EBCDIC of the print data to be spooled. This exit is called whenever a portion of the data file to be printed is received from TCP/IP. On exit the print record vector may contain EBCDIC data to be spooled, or it may contain a response message in ASCII to be sent to the TCP/IP LPR client. If the data returned is in EBCDIC, then a CCW opcode associated with the data **must also** be returned in the print record vector.

Incoming data for the file **may not arrive** on any kind of record boundary. This exit will have to save data in a local buffer to accumulate a record over multiple calls or there may be multiple records in a buffer.

This exit routine must set the data count field in the print record vector to reflect the length of the data passed back to the LPD link driver. If this exit routine does not generate data, it **must** set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable



Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print record vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable on entry; on return, contains the CCW opcode associated with the print line</p> <p><b>Byte 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPD end of file processing routine sets the register conditions described in [“Exit Conditions”](#) on page 213.

## Return Codes

Return Code	Results
0	Tells RSCS to write the current print data to spool if returned length is greater than 0.
4	Tells RSCS to write the current print data to spool if returned length is greater than 0, and call this exit routine again.
8	Tells RSCS to send response message contained within the print record vector to the TCP/IP LPR client, flush the spool file if created, and close the connection.

Return Code	Results
12	Tells RSCS to terminate the LPD link driver.
16	Tells RSCS to terminate the LPD link driver.

## LPD End of File Routine

This exit routine allows for additional EBCDIC data to be spooled, or a response message in ASCII to be sent to the TCP/IP LPR client. It is called after the last piece of data has been received from the TCP/IP LPR client. This enables any specific information in EBCDIC to be forwarded. On entry, the print record vector is empty. On exit the print record vector may contain EBCDIC data to be spooled, or it may contain a response message in ASCII to be sent to the TCP/IP LPR client. If the data returned is in EBCDIC, then a CCW opcode associated with the data **must also** be returned in the print record vector.

This exit routine must set the data count field in the print record vector to reflect the length of the data that is passed back to the LPD link driver. If this exit routine does not generate any data, it **must** set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print record vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the print record</p> <p><b>Byte 2</b> Not applicable on entry; on return, contains the CCW opcode associated with the print line</p> <p><b>Byte 3 - <i>n</i></b> Print record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPD control file processing routine sets the register conditions described in [“Exit Conditions”](#) on page 213.

## Return Codes

Return Code	Results
0	Tells RSCS to write the current print data to spool if returned length is greater than 0.
4	Tells RSCS to write the current print data to spool if returned length is greater than 0, and call this exit routine again.
8	Tells RSCS to send response message contained within the print record vector to the TCP/IP LPR client, flush the spool file if created, and close the connection.

Return Code	Results
12	Tells RSCS to terminate the LPD link driver.
16	Tells RSCS to terminate the LPD link driver.

## LPD Control File Routine

This exit routine is entered for each line of control file received from a TCP/IP LPR client. The line is contained within the print record vector and is in ASCII. Either the line is returned in EBCDIC for RSCS processing, or a response message in ASCII is returned to be sent to the TCP/IP LPR client.

This exit routine must set the data count field in the print record vector to reflect the length of the data passed back to the LPD link driver. If this exit routine does not generate data, it **must** set the data count field to zero. If the data count is negative or exceeds 1280 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical print record vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the control file record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Control file record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable

Register	Contents
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPD Termination routine sets the register conditions described in [“Exit Conditions”](#) on page 213.

## Return Codes

Return Code	Results
0	Tells RSCS that the control file processing is complete.
4	Tell RSCS to ignore this control file line.
8	Tells RSCS to send response message contained within the print record vector to the TCP/IP LPR client, flush the spool file if created, and close the connection.
12	Tells RSCS to terminate the LPD link driver.
16	Tells RSCS to terminate the LPD link driver.

## LPD Termination Routine

This exit routine is called just before the LPD-type link terminates to perform any special termination processing that might be needed. As supplied, RSCS does not provide for any special processing when an LPD link is terminated. This exit routine is optional.

### Attention

This exit may be required if any of the other LPD exit routines (such as initialization) obtain storage from GCS which has not yet been returned to GCS.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Address of the work area established by the initialization routine <b>Word 7 (+24)</b> 0
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the LPD Termination routine sets the register conditions described in [“Exit Conditions” on page 213](#).

## Sample LPD Exit Routine

The following sections describe the sample LPD exit routine module supplied by IBM. This exit routine module is provided in the sample load library, RSCSEXIT LOADLIB.

Generally, the sample exit routine responds to options supplied on a control file sent from a TCP/IP LPR client. These options are supplied when using an LPR command, whose specific format depends on the platform from which it is issued.

### Attention

The sample LPD exit routine is provided for illustrative purposes and on an *as is* basis only. However, you may be able to use the sample exit with little or no modifications, depending on your installation's needs and configuration.

If you choose to support the LPD link drivers differently or to support additional devices, you can modify the sample module or use the sample as a guide for creating your own exit routines.

## LPDXMANY Routine

The LPDXMANY sample exit routine performs functions to support a single LPD printer queue. Multiple LPD link drivers can be defined using this sample exit. It also performs simple translation of data to EBCDIC.

The spool file created will be of type VAFP. A record width up to 255 characters will be supported. Any records received with a width greater than 255 characters will be split into multiple records. The CMS receive command will create a file with a record length of 204. Any data in a record past 204 will be

truncated. CP will also truncate the data with a record length greater than 204 if the spool file is destined to a CP system printer.

### Attention

The LPDXMANY sample LPD link driver exit will operate more consistently and seamlessly if the control file is sent **prior** to the data file. LPDXMANY will accept the control file after the data file, but information required for file processing may not be set up as desired.

The LPDXMANY routine parses the queue name on the printer job command received from the LPR client, as shown in the following examples:

#### **KERRY@MAINE**

Sets the node ID to MAINE and the user ID to KERRY, causing the file to be sent to user KERRY at node MAINE.

#### **KERRY%MAINE**

Sets the node ID to MAINE and the user ID to KERRY, causing the file to be sent to user KERRY at node MAINE.

#### **KERRY@**

Sets the node ID to the local node name and the user ID to KERRY, causing the file to be sent to user KERRY on the local system.

#### **KERRY%**

Sets the node ID to the local node name and the user ID to KERRY, causing the file to be sent to user KERRY on the local system.

#### **@LASER1**

Sets the node ID to LASER1 and the user ID to SYSTEM, causing the file to be sent to the network node LASER1.

#### **%LASER2**

Sets the node ID to LASER2 and the user ID to SYSTEM, causing the file to be sent to the network node LASER2.

#### **LASER3**

Sets the node ID to LASER3 and the user ID to SYSTEM, causing the file to be sent to the network node LASER3.

#### **Notes:**

1. If you are using an LPDXMANY configuration file, LPDXMANY will first look for a queue name record with that name, or a record called DEFAULT. See [“LPDXMANY Configuration file” on page 226](#). If LPDXMANY cannot find a matching queue name record (or a DEFAULT record), or if it cannot find the configuration file, it will parse the queue name as described above.
2. When parsing the queue name, LPDXMANY will assume that any queue name which does not include the @ or % operator is a node ID.
3. LPDXMANY will limit the length of the user ID and node ID to 8 characters each and will discard any extra data in those fields of the queue name.

## Supported Control File Commands

The control file commands supported by this exit are:

#### **H**

Host name - The first 8 characters of the host name will be used for the distribution field of the TAG.

#### **N**

File name - The first 8 characters of the file name will be used for the data set name field of the TAG. Some parsing will be done to remove dots and slashes. For example, the string:

```
c:\AUTOEXEC.BAT
```

will become AUTOEXEC BAT.

### I

Indent - Defines the maximum allowed line width. The default is 255.

### C

Class name - The class name will be used to set the form field and class field of the TAG, and to pass a job name in. LPDXMANY will parse the class name looking for the following:

#### **F=***formname*

specifies the 1- to 8-character form name that will be used for the form field. Any blanks within the name will be translated to underscores (\_).

#### **C=***class*

specifies the 1-character class name.

#### **J=***jobname*

specifies the 1- to 8-character job name.

**Note:** This must be used with the JOBName=*userid* option.

### J

Job name - The first 8 characters of the job name will be used for the data set name field of the TAG if not already set by the N record.

### P

User name - The first 8 characters of the user name will be used for the origin user field of the TAG if not already set by J=*jobname* passed in the C record.

### T

Title - The first 8 characters of the title will be used for the title printed at the top of every page.

### W

Width - Defines the line width, with a maximum size limited by the length specified in the I record. The default is 255 or the maximum specified in the I record (up to 255). Records received with a width greater than what is specified using either the W record or I record or the default maximum of 255 will be split into multiple records.

### f

Print formatted file - Requests that page ejects and title pages be inserted into the created spool file.

### l

Print file leaving control characters - Used as the default filter.

### p

Print file with p $\pi$  format - Requests that page ejects be inserted into the created spool file.

**Note:** The following control file commands will be used to determine the TAG copy count: f, l, p, r, o, v. The first occurrence of any in the list will be accepted. A count of the number of times this command occurs in the control file will determine the TAG copy count. Any others in the list after the first one is accepted will be ignored. The default TAG copy count is 1.

## Available EPARM Parameters

When using the LPDXMANY routine, you can specify the following parameter in the EPARM value on the PARM configuration statement or on the RSCS DEFINE or START command.

#### **Config=***ddname*

specifies the *ddname* which has been defined as an exit configuration file. If the *ddname* does not exist, the LPDXMANY initialization routine will pass back a return code to cause the LPD-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the LPDXMANY exit, causing existing defaults to be used for values which can be defined by the configuration file.

## LPDXMANY Configuration file

The LPDXMANY exit routine can read a configuration file. This configuration file can supply the following:



- Overrides for processing when a file is received from a remote LPR command, based on the printer queue name
- Translate table to override the one used by the exit

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be specified on the Config= parameter in the EPARM value on the PARM statement for the link in the RSCS CONFIG file or in the link operational parameters on the RSCS DEFINE or START command.

The following is an example of a DDNAME entry in PROFILE GCS in which LPDX is the defined DDNAME and LPD CONFIG is the name of the LPDXMANY configuration file:

```
'FILEDEF LPDX DISK LPD CONFIG *'
```

The following is an example of the PARM statement for an LPD-type link named LPD using the DDNAME defined on the FILEDEF statement in PROFILE GCS:

```
PARM LPD EXIT=LPDXMANY PORT=994 EPARM='C=LPDX'
```

## Layout of the LPDXMANY Configuration File

The following rules apply to the LPDXMANY configuration file:

- An asterisk (\*) in column one denotes a comment line.
- Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry.
- All configuration entries must be capitalized.

The following configuration records are supported:

### **LOCAL\_NODE=string**

provides a 1- to 8-character origin node name that LPDXMANY assigns to the created spool file. If this option is not specified, LPDXMANY will use the link name assigned to the LPD-type link as the origin node name.

### **TOASCII=string**

provides a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCII= records to replace the 256-byte translation table.

### **TOEBCCMD=string**

provides a table for ASCII to EBCDIC translation of the LPR control file and commands, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOEBCCMD= records to replace the 256-byte translation table.

### **TOEBCDIC=string**

provides a table for ASCII to EBCDIC translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOEBCDIC= records to replace the 256-byte translation table.

### **queuenam**

provides the ability to override defaults used by LPDXMANY on a printer queue name basis when receiving a file from a remote host. Multiple unique printer queue name records can be specified.

When a printer queue name arrives, LPDXMANY will first look for a matching queue name record. If none is found, it will look for a record with the name DEFAULT. A DEFAULT record can be used to define parameters for any printer queue not defined by its own configuration record. If a DEFAULT record is not found, LPDXMANY will use the existing defaults.

The format of the printer queue name record is:

```
queuenam ppos lpage class forms jobn dest pagination translation userid nodeid  
tcp$lbin
```

The following rules apply to the printer queue name record:

- One record is allowed per line; continuation is not supported.
- The parameters of the record must be separated by one or more blanks.
- The parameters are not column dependent but they are position dependent.
- An asterisk (\*) can be used for any parameter except *queuenam*e to tell LPDXMANY to use the existing default.

The parameters of the printer queue name record are defined as follows:

### DEFAULT

#### ***queuenam*e**

is a printer queue name up to 32 characters, or DEFAULT.

LPDXMANY parses the printer queue name into a user ID and node ID, as follows:

#### ***userid@nodeid***

Node ID and user ID will be set as specified.

#### ***userid%nodeid***

Node ID and user ID will be set as specified.

#### ***userid@***

Node ID will be set to the local node name, user ID will be set as specified.

#### ***userid%***

Node ID will be set to the local node name, user ID will be set as specified.

#### ***@nodeid***

Node ID will be set as specified, user ID will be set to SYSTEM.

#### ***%nodeid***

Node ID will be set as specified, user ID will be set to SYSTEM.

#### ***nodeid***

Node ID will be set as specified, user ID will be set to SYSTEM.

The user ID and node ID parsed from the printer queue name can be overridden within the record. If the *queuenam*e value is DEFAULT or any other value that does not conform to one of the variations listed above, the user ID and node ID should be set (overridden) within the record. If both are overridden, the *queuenam*e value is not parsed. If neither is overridden, the *queuenam*e value is assumed to be a printer queue name, which is parsed into the user ID and node ID, which must be valid (either of which can still be overridden within).

### Notes:

1. When parsing the queue name, LPDXMANY will assume that any queue name which does not include the @ or % operator is a node ID.
2. LPDXMANY will limit the length of the user ID and node ID to 8 characters each and will discard any extra data in those fields of the queue name.

#### ***ppos***

is the logical record length. This value can be 1 - 1280. The default is 255.

The virtual printer type will be defined based on the *ppos* value:

<b><i>ppos</i></b>	<b>Virtual Printer</b>
1 - 132	1403
133 - 150	3211
151 - 204	3800
205 - 1280	VAFP

#### ***lpage***

is the number of lines per page. This value can be 1 - 99. The default is 66.

**class**

is the 1-character spool file class. The default is blank.

**forms**

is the 1- to 8-character spool file form. The default is blank.

**jobn**

is the 1- to 8-character job name. The default is SYSTEM.

**dest**

is the 1- to 8-character PSF destination. The default is blank. When using a PSF destination, LPDXMANY will set the user ID field of the TAG to SYSTEM.

When using a PSF destination, the printer queue name can be in the form `SYSTEM@nodeid`. Alternatively, the printer queue name can be anything unique as long as the user ID field is SYSTEM or \* and the node ID field is specified.

**pagination**

specifies how pagination will be performed.

**PAGE**

LPDXMANY will always paginate regardless of the control file print filter.

**NOPAGE**

LPDXMANY will paginate only as defined by the f or p control file print filters. This is the default. To be effective, the control file **must** be received prior to the data file.

**translation**

specifies whether to translate the data file prior to spooling.

**TRAN**

Translate data received into EBCDIC removing any CR/LF/FF (carriage return, line feed, form feed) control characters. This is the default.

**NOTRAN**

Do not translate data received into EBCDIC, but remove any CR/LF/FF (carriage return, line feed, form feed) control characters.

**ASISCC**

Leave the data received as is; do not translate nor remove any control characters. ASISCC would be desired when data received is destined to be printed by an LPR-type link.

**userid**

is the 1- to 8-character user ID to which the file should be spooled. The default is derived from the printer queue name.

**Note:** If a PSF destination is provided by the *dest* field, then the *userid* field can be set only to SYSTEM.

**nodeid**

is the 1- to 8-character destination node ID to which the file should be spooled. The default is derived from the printer queue name.

**tcpxlbibin**

is the file name of a TCP/IP translation table (TCPXLBIN file) to be used for this printer queue name. The file must exist on any accessed disk. If this parameter is not specified, the translation table used will be either the default table within LPDXMANY or the override specified on one or more TOEBCDIC= records in the LPDXMANY configuration file.

**Printer Queue Name Examples**

This example would allow a file to be received without translating, and spooled to an LPR link using the LPRXONE exits, causing the file to be sent unaltered to a line printer daemon:

```
LPR@NODEONE 1280 50 * STDN * * * ASISCC ASCII
```

This example defines a printer queue used to spool input to the system printer:

```
LOCALNODE * * * * * SYSTEM
```

This example shows the defaults defined within LPDXMANY:

```
DEFAULT 255 66 * * SYSTEM * NOPAGE TRAN * * *
```

## Using an LPD-Type Link as a Print Server

It is feasible to use RSCS for routing workstation printing for delivery to IP (internet) attached printers using LPR-type links. To do so, you will need to use the LPRXFORM exit as well as the LPDXMANY CONFIG file. For more information, see [\*z/VM: RSCS Networking Operation and Use\*](#).

---

## Chapter 9. TCP/IP UFT Exit Points

The TCP/IP UFT link driver uses exits to build specific data streams for transmission and to control the remote host and port to which the transmission is destined. Exit routines are called at six points in the UFT link driver's operational cycle.

Each UFT exit routine contains entry points, which provide the following additional processing for each of the six calls:

Entry Point	Function
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 connecting to the remote UFT 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.

The termination routine is not necessary to support a UFT link driver; you do not have to supply this routine. An exit may require the termination routine for potential clean up processing such as returning any storage obtained in any of the other five exit routines.

The order in which the exit routines are listed above is not necessarily the order in which the UFT link driver will call them when processing a spool file.

The UFT exit routine module that contains the exit routines to be used for a specific UFT-type link must be identified on the PARM configuration statement for the link or in the link operational parameters on the RSCS DEFINE or START command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

---

## UFT Programming Considerations

The programming requirements for the UFT exit routines are described in the following sections.

### Required Values

The first six fullwords of each UFT exit routine module must contain these values:

**Word 1**

Address of the initialization routine

**Word 2**

Address of the TAG processing routine

**Word 3**

Address of the record processing routine

**Word 4**

Address of the end of file processing routine

**Word 5**

Address of the command processing routine

**Word 6**

Address of the termination routine, or 0 if the routine is not provided

## Entry Conditions

When a UFT exit routine receives control, it will be passed the following information:

- Address of the CVT
- Address of the LINKTABL entry
- Address of a UFTBLOK structure containing the following pointers:
  - 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, derived from an INMR02 NETDATA control record
  - Address of a doubleword containing the file size, in bytes, derived from an INMR02 NETDATA control record
  - Address of a doubleword containing the number of files, derived 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, derived from an INMR02 NETDATA control record
  - Address of a 1-character field containing the server's UFT level

This information is **not passed** to the initialization or termination routines.

All exit routines, except the UFT initialization and termination routines, also receive a pointer to the TAG element. The TAG element contains information about a file's characteristics. The exit routine also receives the EPARM value for the link and the address of the data record vector. The EPARM value, specified on the PARM configuration statement or on the RSCS DEFINE or START command, contains a parameter string that is associated with the UFT exit routine.

## UFT Commands

The command and end of file routines will have the sole responsibility for creating UFT commands that RSCS will send to the TCP/IP UFT daemon. RSCS will generate only the DATA, QUIT, and ABORT commands, relying on the exit routines to generate all other appropriate UFT commands.

The command and end of file routines should generate only the DATA command when that exit is also going to send additional data to the UFT daemon. They **should not** generate the DATA command in response to actual data created by the record processing routines. RSCS will generate the DATA command when it has buffered enough data, that was output from the record processing routine, to send to the remote daemon.

A return code of 8 from the command and end of file exit routines can be used to send a UFT command to the TCP/IP UFT daemon.

## Data Record Vector

The UFT exits use the logical data record vector to pass a data stream to a link driver exit for conversion from EBCDIC to ASCII or binary, if appropriate. The logical data record vector is also used to pass a UFT

command from the exit to the link driver. The "Entry Conditions" section for each exit routine describes the contents of the data record vector.

## Exit Conditions

When a UFT exit routine returns control to RSCS, the registers contain these values:

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## UFT Exit Routines

The following sections describe each of the exits supported for UFT-type links. For more information, see ["Required Values" on page 231](#).

### UFT Initialization Routine

This routine initializes the UFT-type link. The exit is not passed any link options, such as the remote port to which to connect.

If you specify RENT when link-editing this routine, any storage that will be used by the remaining entry points must be obtained by issuing a GCS GETMAIN macro during this exit routine processing. The address of this storage must be placed in word 6 of the parameter list so that the other routines can access the work area. In this case, it is **required** that a **termination** exit routine issue the GCS FREEMAIN macro to return any persistent storage obtained in the **initialization** exit routine.

### Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 on entry; on return, it may contain the address of the UFT exits work area <b>Word 7 (+24)</b> Address of the UFTBLOK
R2 - R12	Not applicable
R13	Save area address

Register	Contents
R14	Return address

## Exit Conditions

On return, the UFT Initialization routine sets the register conditions described in [“Exit Conditions”](#) on page 233.

## Return Codes

Return Code	Results
0	Tells RSCS that initialization processing is complete.
4	Tells RSCS that an EPARM value was specified, but the exit routine does not need it; the UFT link driver terminates.
8	Tells RSCS that an EPARM value was specified which was not valid; the UFT link driver terminates.
12	Tells RSCS to terminate the UFT link driver.
16	Tells RSCS to terminate the UFT link driver.

## UFT TAG Processing Routine

This routine examines a file's TAG element. Based on a file's characteristics, the exit routine can be used to customize where individual files are to be sent in a TCP/IP network. Your exit can change TCP/IP-specific information that was defined by the PARM statement for the UFT-type link, or provided with the file. The exit routine is passed an address of a control block containing addresses to the following information, which can be modified, or used by later exit processing, for the UFT information:

- The remote host IP address
- The remote host name
- The remote host port
- The remote user name
- The transform to be used
- The translate table to be used

In addition, this exit routine can be used to reject the transmission of a file to a remote UFT daemon.

This exit routine does not generate data, it **must** set the data count field to zero. If the data count is not zero, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable



Register	Contents
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical data vector that contains: <b>Byte 0 - 1</b> 0 <b>Byte 2 - n</b> Not applicable <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Address of the work area established by the initialization routine <b>Word 7 (+24)</b> Address of the UFTBLOK
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFT TAG processing routine sets the register conditions described in [“Exit Conditions”](#) on page 233.

## Return Codes

Return Code	Results
0	Tells RSCS that the TAG element processing is complete.
4	Tells RSCS to reject processing this file and place it on hold.
8	Tells RSCS to terminate the UFT link driver.
12	Tells RSCS to terminate the UFT link driver.
16	Tells RSCS to terminate the UFT link driver.

## UFT Record Processing Routine

The record processing routine carries out any appropriate translation of the data, for example from EBCDIC to ASCII. If the exit routine changes the length of the data, the data count field (pointed to by Bytes 0 - 1 in Word 3 of the parameter list) must reflect this change before returning to the link driver. When the link driver regains control from this entry point, the data from the data record moves into the link driver's output buffer. When it is full, the link driver sends the buffer to the remote TCP/IP UFT daemon. This exit is driven for each record of the spool file being sent to a TCP/IP UFT daemon.

Each spool file record read will be in NETDATA format. Therefore, the spool file data **may not be read** on any kind of record boundary. This exit will have to save data in a local buffer to accumulate a record over multiple calls or there may be multiple records in a buffer.

Your exit routine must set the data count field in the data record vector to reflect the length of the data passed to the link driver. If your routine does not send a particular data record, it should set the data count field to zero. A data count that is negative or exceeds 1280 bytes terminates the link with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical data vector that contains: <b>Byte 0 - 1</b> Number of bytes in the data record <b>Byte 2</b> CCW opcode associated with the data record <b>Byte 3 - n</b> Data record data (where <i>n</i> is a maximum of 1280) <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Address of the work area established by the initialization routine <b>Word 7 (+24)</b> Address of the UFTBLOK
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFT record processing routine sets the register conditions described in [“Exit Conditions” on page 233](#).

## Return Codes

Return Code	Results
0	Tells RSCS that the spool file record element processing is complete.
4	RSCS adds the current record to the buffer and calls this exit routine again.
8	Tells RSCS to terminate the UFT link driver.

Return Code	Results
12	Tells RSCS to terminate the UFT link driver.
16	Tells RSCS to terminate the UFT link driver.

## UFT End of File Routine

This exit routine allows for additional information to be sent to the TCP/IP UFT daemon. It is called after the last spool file record has been processed. This allows for the following:

- Any specific device-dependent information to be transmitted
- EOF UFT command to be transmitted
- Any other UFT commands to be transmitted
- Any necessary clean up to be performed

Your routine must set the data count field in the data record vector to reflect the length of the data that is passed to the link driver. On exit the data vector may contain ASCII or binary data to be transmitted. If your routine does not generate any data, it should set the data count field to zero. A data count that is negative or exceeds 1280 bytes terminates the link with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical data vector that contains:</p> <p><b>Byte 0 - 1</b> Number of bytes in the data record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - <i>n</i></b> Data record data (where <i>n</i> is a maximum of 1280)</p> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of the UFTBLOK</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFT end of file processing routine sets the register conditions described in [“Exit Conditions”](#) on page 233.

## Return Codes

Return Code	Results
0	Tells RSCS to transmit the current command or data to the remote UFT daemon; end of file processing is complete.
4	RSCS adds the current data or command to the buffer and calls this exit routine again.
8	RSCS transmits the current command or data to the remote UFT daemon and waits for positive acknowledgement; when received, call this exit routine again.
12	Tells RSCS to terminate the UFT link driver.
16	Tells RSCS to terminate the UFT link driver.

## UFT Command Routine

This exit routine creates one UFT command to be sent to the TCP/IP UFT daemon. If more than one UFT command is to be transmitted to the remote daemon, this exit routine should return the command with appropriate return code to be called again. The exit will be responsible for remembering which commands have already been transmitted, which commands still require transmitting, and for clean up (in the End of File routines) when a negative acknowledgement has been received and transmission has been aborted.

This exit routine can also be used to create additional data, such as header information, that RSCS passes to the TCP/IP UFT daemon.

Your routine must set the data count field in the data record vector to reflect the length of the data that is passed to the link driver. If your routine does not generate any data, it should set the data count field to zero. A data count that is negative or exceeds 1280 bytes terminates the link with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> Address of the logical data vector that contains: <b>Byte 0 - 1</b> Number of bytes in the UFT command <b>Byte 2</b> Not applicable <b>Byte 3 - n</b> Not applicable on entry; on return, contains one UFT command (where <i>n</i> is a maximum of 1280) <b>Word 4 (+12)</b> Address of the TAG element <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Address of the work area established by the initialization routine <b>Word 7 (+24)</b> Address of the UFTBLOK
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFT command processing routine sets the register conditions described in [“Exit Conditions”](#) on page 233.

## Return Codes

Return Code	Results
0	Tells RSCS to transmit the current command or data to the remote UFT daemon; UFT command processing is complete.
4	RSCS adds the current data or command to the buffer and calls this exit routine again.
8	RSCS transmits the current command or data to the remote UFT daemon and waits for positive acknowledgement; when received, call this exit routine again.
12	Tells RSCS to terminate the UFT link driver.
16	Tells RSCS to terminate the UFT link driver.

## UFT Termination Routine

This exit routine is called just before the UFT-type link terminates to perform any special termination processing that might be needed. As supplied, RSCS does not provide for any special processing when a UFT-type link is terminated. This exit routine is optional.

**Attention**

This exit might be required if any of the other UFT exit routines (such as initialization) obtain persistent storage from GCS which has not yet been returned to GCS.

**Entry Conditions**

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Address of the work area established by the initialization routine <b>Word 7 (+24)</b> 0
R2 - R12	Not applicable
R13	Save area address
R14	Return address

**Exit Conditions**

On return, the UFT Termination routine sets the register conditions described in [“Exit Conditions” on page 233](#).

**Sample UFT Exit Routine**

The following sections describe the sample UFT exit routine module supplied by IBM. This exit routine module is provided in the sample load library, RSCSEXIT LOADLIB.

Generally, the sample exit routine responds to options supplied on the CP TAG, SPOOL, or CLOSE command, or on the CMS SENDFILE command, to perform any needed translations of the data as well as when constructing UFT commands.

**Attention**

The sample UFT exit routine is provided for illustrative purposes and on an *as is* basis only. However, you may be able use the sample exit with little or no modifications, depending on your installation's needs and configuration.

If you choose to support the UFT link drivers differently or to support additional devices, you can modify the sample module or use the sample as a guide for creating your own exit routines.

## UFTXOUT Routine

The UFTXOUT sample exit routine performs functions to support transmitting one file to a remote UFT daemon at a time. Multiple UFT link drivers can be defined using this sample exit. It also performs simple translation of data to ASCII.

### Supported UFT Commands

The UFT commands created by UFTXOUT are the minimum necessary for transmitting the data. These include:

#### CLASS

From the TAG (TAGCLASS), if specified. If TRANSFORM=SP00L is specified within the spool file, a second operand will be provided with the device type of 1403, 3211, 3800, PUN, or VAFF. Print files with a logical record length equal to or less than 1280 are supported.

#### COPY

From the TAG (TAGCOPY).

#### DATE

The date, time, and time zone of the file from the UFTBLOK (as specified by an INMR02 NETDATA control record), if specified, otherwise from the TAG (TAGINTOD).

#### DEST

From the TAG (TAGDEST), if specified.

#### DIST

From the TAG (TAGDIST), if specified.

#### EOF

#### FILE

This only needs to be an estimated size. The value specified in the UFTBLOK (from an INMR02 NETDATA control record) will be used if specified; otherwise zero is used.

In addition, the second required parameter will be the originator of the file (TAGINVM).

#### FORM

From the TAG (TAGFORMN), if specified.

#### NAME

This will be in the form *filename.filetype*. From the UFTBLOK (as specified by an INMR02 NETDATA control record).

#### OWNER

From the OWNER= record of the configuration file, if specified.

#### RECFMT

From the UFTBLOK (as specified by an INMR02 NETDATA control record).

#### RECLEN

From the UFTBLOK (as specified by an INMR02 NETDATA control record).

#### TYPE

This will be determined from the transform field of the UFTBLOK. UFTXOUT supports the following transform types and specifies the indicated TYPE code.

Type	Code	Processing
ASCII	A	Data will be translated and X'0D0A' added to the end of each record.
BINARY	I	File data will be transmitted unaltered.
EBCDIC	E	X'15' will be added to the end of each record.
MAIL	M	Data will be translated to ASCII.
NETDATA	N	File will be transmitted in NETDATA format.

Type	Code	Processing
SPOOL	V M	A 2-byte length followed by a CCW opcode will precede each data record. This allows VM spool files to be transferred to another VM system unaltered.
TEXT	A	Data will be translated and X'0D0A' added to the end of each record.
VARREC	V	A 2-byte length will precede each data record.  <b>Note:</b> VARREC does not support a logical record length greater than 65535.
<b>Note:</b> The ASCII, BINARY, EBCDIC, NETDATA, TEXT, or VARREC transform type can be specified on the UFTASYNCR option of the CMS SENDFILE command. UFTXOUT also supports transform types MAIL and SPOOL. Any other type received will be rejected.		

**USER**

This will be as parsed from the data prior to the at sign (@) character in the DESTADDR= keyword.

**Available EPARM Parameters**

When using the UFTXOUT routine, you can specify the following parameter in the EPARM value on the PARM configuration statement or on the RSCS DEFINE or START command.

**Config=ddname**

specifies the DDNAME which has been defined as an exit configuration file. If the DDNAME does not exist, the UFTXOUT initialization routine will pass back a return code to cause the UFT-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the UFTXOUT exit, causing existing defaults to be used for values which can be defined by the configuration file.

**UFTXOUT Configuration File**

The UFTXOUT sample exit routine can read a configuration file. This configuration file can supply the following:

- Translation tables to override the ones used by the exit
- Overrides for the UFT commands created by UFTXOUT

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be supplied on the Config= link exit parameter statement when defining the UFT-type link in the RSCS CONFIG file.

The following is an example of a DDNAME entry in the PROFILE GCS in which UFTOUT is the defined DDNAME and UFTO SCONFIG is the name of the configuration file:

```
'FILEDEF UFTOUT DISK UFTO SCONFIG *'
```

The following is an example of the RSCS CONFIG parameter (PARM) for a UFT-type link called UFT using the DDNAME defined on the FILEDEF statement in the PROFILE GCS:

```
PARM UFT EXIT=UFTXOUT EPARM='C=UFTOUT'
```

**Layout of the UFTXOUT Configuration File**

The following rules apply to the UFTXOUT configuration file:

- An asterisk (\*) in column one denotes a comment line.
- Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry.



- All configuration entries must be capitalized.
- Entries can span multiple records.

The following configuration records are supported:

**OWNERNAME=string**

specifies an owning user ID name, up to 32 characters, for the OWNER UFT command.

**TOASCII=string**

provides a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCII= records to replace the 256-byte translation table.

**TOASCIIC=string**

provides a table for EBCDIC to ASCII translation of UFT commands, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) may be specified on multiple TOASCIIC= records to replace the 256-byte translation table.



---

## Chapter 10. TCP/IP UFTD Exit Points

The TCP/IP UFTD link driver uses exits to build specific data streams for acceptance into the RSCS network and to control the remote node and user ID to which the data is destined. Exit routines are called at six points in the UFTD link driver's operational cycle.

Each UFTD exit routine contains entry points, which provide the following additional processing for each of the six calls.

Entry Point	Function
Initialization	Called when the 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 UFTD link driver receives a UFT command from a UFT client.
Data Processing	Called when data has been read from a TCP/IP UFT client for the UFTD link driver.
End of File Processing	Called when a file has been completely read from a TCP/IP UFT client for the UFTD link driver.
Termination	Called when the UFTD link driver is terminating.

The termination routine is not necessary to support a UFTD link driver; you do not have to supply this routine. An exit may require the termination routine for potential clean up processing such as returning any storage obtained in any of the other five exit routines.

The order in which the exit routines are listed above is not necessarily the order in which the UFTD link driver will call them when processing a data stream from a UFT client.

The UFTD exit routine module that contains the exit routines to be used for a specific UFTD-type link must be identified on the PARM configuration statement for the link or on the RSCS DEFINE or START command. For more information, see [z/VM: RSCS Networking Planning and Configuration](#) and [z/VM: RSCS Networking Operation and Use](#).

---

### UFTD Programming Considerations

The programming requirements for the UFTD exit routines are described in the following sections.

#### Required Values

The first six fullwords of each UFTD exit routine module must contain the following values:

**Word 1**

Address of the initialization routine

**Word 2**

Address of the connect processing routine

**Word 3**

Address of the UFT command processing routine

**Word 4**

Address of the data processing routine

**Word 5**

Address of the end of file processing routine

## Word 6

Address of the termination routine, or 0 if the routine is not provided

## Entry Conditions

When a UFTD exit routine receives control, it will be passed the following information:

- Address of the CVT.
- Address of the LINKTABL entry.
- Address of a SOCKADDR structure for the remote host. This is a 16-byte structure containing the addressing family, port number, and IP address of the remote UFT client. This information is **not passed** to the initialization or termination routines.

All exit routines, with the exception of the initialization and termination routines, also receive a pointer to the TAG element. The TAG element contains information about a file's characteristics. The exit routine also receives the EPARM value for the link and the address of the data record vector. The EPARM value, specified on the PARM configuration statement for the link or on the RSCS DEFINE or START command, contains a parameter string that is associated with the UFTD exit routine.

## Order of the UFT Commands and Data

It is recommended that UFT commands be sent first by the UFT client in order for the UFTD driver and exits to work in as simple and seamless a fashion as possible. Otherwise, the exit routines will have to make assumptions about the data prior to obtaining the file attributes from UFT commands.

## Response Messages

All exit routines, with the exception of the initialization and termination routines, will have the sole responsibility for creating both positive and negative response messages which RSCS will send to the TCP/IP UFT client. In most cases this will be in the form of a positive acknowledgement. RSCS **will not** generate any response messages, instead relying on the exit routines to do so.

Acknowledgments are returned to a UFT client whenever a UFT command is received, in the form of a response code number. A positive acknowledgement is a response code number of *1nn* or *2nn*.

A return code of 0 from the Connect and Command exit routines can be used to send a response number to the TCP/IP UFT client. A return code of 4 from the Connect and Command exit routines can be used to send a response number to the TCP/IP UFT client and have the exit routine called again. A return code of 8 from all exit routines, with the exception of the initialization, end of file, and termination routines, can be used to send a response message to the TCP/IP UFT client, close and purge the spool file if already created, and close the connection.

## Data Record Vector

The UFTD exits use the logical data record vector to pass a data stream to a link driver exit for conversion from ASCII or binary to EBCDIC. The logical data record vector is also used to pass UFT commands between the link driver and the exit routine as well as pass response messages between the exit routine and the link driver.

Handling of data within the data record vector for the UFTD-type link is different than for other links which use a data or print record vector (such as the ASCII-type, LPD-type, LPR-type, and UFT-type links). For UFTD-type links, the data record vector is used as follows:

- On entry to the exit, the data record vector may contain:

### Byte 0 - 1

Number of bytes in the data record.

### Byte 2

Not applicable

**Byte 3 - *n***

Data or UFT command received from the remote client

- On return from the exit, the data record vector may contain:

**Byte 0 - 1**

Number of bytes at the address pointed to by bytes 4 - 7, which **must not** exceed 32760 bytes

**Byte 2**

The CCW associated with any data to be spooled

**Byte 3**

Not applicable

**Byte 4-7**

An address pointing to the data to be spooled or pointing to the response message to be sent to the remote client

The "Entry Conditions" section for each exit routine describes the contents of the data record vector.

## Exit Conditions

When a UFTD exit routine returns control to RSCS, the registers contain these values:

Register	Contents
R0 - R1	Not applicable
R2 - R13	Restored to the same values as on entry
R14	Not applicable
R15	Return code

## UFTD Exit Routines

The following sections describe each of the exits supported for UFTD-type links. For more information, see ["Required Values" on page 245](#).

### UFTD Initialization Routine

This exit routine initializes the UFTD-type link. The exit is not passed any link options and therefore cannot change the TCP/IP port the link is listening on during this exit routine processing.

If you specify RENT when link-editing this routine, any storage that will be used by the remaining entry points must be obtained by issuing a GCS GETMAIN macro during this exit routine processing. The address of this storage must be placed in word 6 of the parameter list so that the other routines can access the work area. In this case, it is **required** that a **termination** exit routine issue the GCS FREEMAIN macro to return any persistent storage obtained in the **initialization** exit routine.

### Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> 0 on entry; on return, it may contain the address of the work area for the UFTD exit routines <b>Word 7 (+24)</b> 0
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFTD Initialization routine sets the register conditions described in [“Exit Conditions” on page 247](#).

## Return Codes

Return Code	Results
0	Tells RSCS that initialization processing is complete.
4	Tells RSCS that an EPARM value was specified, but the exit routine does not need it; the UFTD link driver terminates.
8	Tells RSCS that a specified EPARM value was not valid; the UFTD link driver terminates.
12	Tells RSCS to terminate the UFTD link driver.
16	Tells RSCS to terminate the UFTD link driver.

## UFTD Connect Processing Routine

This exit routine is called when a connect request has been received, and accepted, from a TCP/IP UFT client. This exit routine determines if the file should be received from the remote host and processed. This exit routine returns a response message in ASCII to be sent to the TCP/IP UFT client.

This exit routine must set the data count field in the data record vector to reflect the length of the data that is passed back to the UFTD link driver. If the data count is negative, zero, or exceeds 32760 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical data record vector that contains:</p> <ul style="list-style-type: none"> <li>On entry to the exit: <p><b>Byte 0 - 1</b> 0</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - n</b> Not applicable</p> </li> <li>On return from the exit: <p><b>Byte 0 - 1</b> Number of bytes at the address pointed to by bytes 4 - 7, which <b>must not</b> exceed 32760 bytes</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3</b> Not applicable</p> <p><b>Byte 4 - 7</b> An address pointing to the response message to be sent to the remote client</p> </li> </ul> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFTD connect processing routine sets the register conditions described in [“Exit Conditions”](#) on page 247.

## Return Codes

Return Code	Results
0	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client and continue processing.
4	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client and call this exit routine again.
8	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client, reject the connect request, and close the connection.
12	Tells RSCS to terminate the UFTD link driver.
16	Tells RSCS to terminate the UFTD link driver.

## UFTD Command Processing Routine

This exit routine is called when a command is received from a TCP/IP UFT client. The command received is contained within the data record vector and is in ASCII format. This exit routine carries out any appropriate data translation from ASCII to EBCDIC, returning a response message in ASCII to be sent to the TCP/IP UFT client.

This routine is also responsible for filling in the file's TAG text fields based on information received from the UFT client. The exit routine inserts the TAG characteristics into the TAG text fields portion of the parameter list.

This exit routine must set the data count field in the data record vector to reflect the length of the data that is passed back to the UFTD link driver. If the data count is negative or exceeds 32760 bytes the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable



Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical data record vector that contains:</p> <ul style="list-style-type: none"> <li>On entry to the exit: <p><b>Byte 0 - 1</b> Number of bytes in the data record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - n</b> UFT command data</p> </li> <li>On return from the exit: <p><b>Byte 0 - 1</b> Number of bytes at the address pointed to by bytes 4 - 7, which <b>must not</b> exceed 32760 bytes</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3</b> Not applicable</p> <p><b>Byte 4 - 7</b> Address of the response message to be sent to the remote client</p> </li> </ul> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFTD command processing routine sets the register conditions described in [“Exit Conditions”](#) on page 247.

## Return Codes

Return Code	Results
0	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client, if the returned length is greater than zero, and continue processing.
4	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client and call the end-of-file processing exit routine.
8	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client and flush the spool file if created.
12	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client, flush the spool file if created, and close the connection.
16	Tells RSCS to terminate the UFTD link driver.

## UFTD Data Processing Routine

The record processing routine carries out appropriate translation from ASCII to EBCDIC of the data to be spooled. This exit is called whenever a portion of the data is received from a TCP/IP UFT client. On exit the data record vector may contain EBCDIC data to be spooled, or it may contain a response message in ASCII to be sent to the TCP/IP UFT client. If the data returned is in EBCDIC, then a CCW opcode associated with the data **must also** be returned in the data record vector.

Incoming data for the file **may not arrive** on any kind of record boundary. This exit will have to save data in a local buffer to accumulate a record over multiple calls or there may be multiple records in a buffer.

This exit routine must set the data count field in the data record vector to reflect the length of the data that is passed back to the UFTD link driver. If the data count is zero, a zero length record will be spooled for print files. If this exit routine does not generate data, it **must** set the data count field to zero. If the data count is negative or exceeds 32760 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical data record vector that contains:</p> <ul style="list-style-type: none"> <li>On entry to the exit: <p><b>Byte 0 - 1</b> Number of bytes in the data record</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - n</b> Data received from the remote client</p> </li> <li>On return from the exit: <p><b>Byte 0 - 1</b> Number of bytes at the address pointed to by bytes 4 - 7, which <b>must not</b> exceed 32760 bytes</p> <p><b>Byte 2</b> CCW opcode associated with the data</p> <p><b>Byte 3</b> Not applicable</p> <p><b>Byte 4 - 7</b> Address pointing to the data to be spooled or pointing to the response message to be sent to the remote client</p> </li> </ul> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFTD data processing routine sets the register conditions described in [“Exit Conditions” on page 247](#).

## Return Codes

Return Code	Results
0	Tells RSCS to write the current data to spool.
4	Tells RSCS to write the current data to spool and call this exit routine again.
8	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client; receipt of this data section is complete.
12	Tells RSCS to send the response message contained within the data record vector to the TCP/IP UFT client, flush the spool file if created, and close the connection.
16	Tells RSCS to terminate the UFTD link driver.

## UFTD End of File Routine

This exit routine allows for additional EBCDIC data to be spooled, as well as any necessary clean up to be performed. It is called after the last piece of data has been received from the TCP/IP UFT client, and an EOF UFT command has been received and responded to. This enables any specific information in EBCDIC to be forwarded. On entry, the data record vector is empty. On exit the data record vector may contain EBCDIC data to be spooled. If data is returned, then a CCW opcode associated with the data **must also** be returned in the data record vector.

This exit routine must set the data count field in the data record vector to reflect the length of the data that is passed back to the UFTD link driver. If this exit routine does not generate data, it **must** set the data count field to zero. If the data count is negative or exceeds 32760 bytes, the link terminates with user ABEND 011.

## Entry Conditions

Register	Contents
R0	Not applicable

Register	Contents
R1	<p>Address of a parameter list that contains:</p> <p><b>Word 1 (+0)</b> Address of the CVT</p> <p><b>Word 2 (+4)</b> Address of the LINKTABL entry</p> <p><b>Word 3 (+8)</b> Address of the logical data record vector that contains:</p> <ul style="list-style-type: none"> <li>On entry to the exit: <p><b>Byte 0 - 1</b> 0</p> <p><b>Byte 2</b> Not applicable</p> <p><b>Byte 3 - n</b> Not applicable</p> </li> <li>On return from the exit: <p><b>Byte 0 - 1</b> Number of bytes at the address pointed to by bytes 4 - 7, which <b>must not</b> exceed 32760 bytes</p> <p><b>Byte 2</b> CCW opcode associated with the data</p> <p><b>Byte 3</b> Not applicable</p> <p><b>Byte 4 - 7</b> Address of the data to be spooled</p> </li> </ul> <p><b>Word 4 (+12)</b> Address of the TAG element</p> <p><b>Word 5 (+16)</b> Address of the EPARM value</p> <p><b>Word 6 (+20)</b> Address of the work area established by the initialization routine</p> <p><b>Word 7 (+24)</b> Address of a SOCKADDR structure for the remote host that contains:</p> <p><b>Byte 0 - 1</b> Addressing Family</p> <p><b>Byte 2 - 3</b> Port number</p> <p><b>Byte 4 - 7</b> IP host address</p> <p><b>Byte 8 - 15</b> Reserved</p>
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFTD end of file processing routine sets the register conditions described in [“Exit Conditions”](#) on page 247.

## Return Codes

Return Code	Results
0	Tells RSCS to write the current data to spool, if the returned length is greater than zero, and close the spool file.
4	Tells RSCS to write the current data to spool, if the returned length is greater than zero, and call this exit routine again.
8	Tells RSCS to terminate the UFTD link driver.
12	Tells RSCS to terminate the UFTD link driver.
16	Tells RSCS to terminate the UFTD link driver.

## UFTD Termination Routine

This exit routine is called just before the UFTD-type link terminates to perform any special termination processing that might be needed. As supplied, RSCS does not provide for any special processing when a UFTD link is terminated. This exit routine is optional.

### Attention

This exit might be required if any of the other UFTD exit routines (such as initialization) obtain persistent storage from GCS which has not yet been returned to GCS.

## Entry Conditions

Register	Contents
R0	Not applicable
R1	Address of a parameter list that contains: <b>Word 1 (+0)</b> Address of the CVT <b>Word 2 (+4)</b> Address of the LINKTABL entry <b>Word 3 (+8)</b> 0 <b>Word 4 (+12)</b> 0 <b>Word 5 (+16)</b> Address of the EPARM value <b>Word 6 (+20)</b> Address of the work area established by the initialization routine <b>Word 7 (+24)</b> 0
R2 - R12	Not applicable
R13	Save area address
R14	Return address

## Exit Conditions

On return, the UFTD Termination routine sets the register conditions described in [“Exit Conditions”](#) on page 247.

## Sample UFTD Exit Routine

The following sections describe the sample UFTD exit routine module supplied by IBM. This exit routine module is provided in the sample load library, RSCSEXIT LOADLIB.

Generally, the sample exit routine responds to options supplied on a command sent from a TCP/IP UFT client. These options are supplied when using an appropriate `sendfile` command, whose specific format depends on the platform from which it is issued.

### Attention

The sample UFTD exit routine is provided for illustrative purposes and on an *as is* basis only. You may be able use the sample exit with little or no modifications, depending on your installation's needs and configuration. However, it is not intended to support all possible UFT commands.

If you choose to support the UFTD link drivers differently or to support additional devices, you can modify the sample module or use the sample as a guide for creating your own exit routines.

## UFTXIN Routine

The UFTXIN sample exit routine performs functions to support a single UFTD connect request. Multiple UFTD link drivers can be defined using this sample exit. It also performs simple translation of data to EBCDIC.

### Attention

The UFTXIN sample UFTD link driver exit will operate more consistently and seamlessly if UFT commands are sent **prior** to the data. UFTXIN will accept UFT commands sent after the data, but information required for file processing might not be set up as desired.

The UFTXIN routine parses the user name on the USER command received from the UFT client, as shown in the following examples:

#### **KERRY@GDLVM7**

Sets the node ID to GDLVM7 and the user ID to KERRY, causing the file to be sent to user KERRY at node GDLVM7.

#### **KERRY%GDLVM7**

Sets the node ID to GDLVM7 and the user ID to KERRY, causing the file to be sent to user KERRY at node GDLVM7.

#### **KERRY@**

Sets the node ID to the local node name and the user ID to KERRY, causing the file to be sent to user KERRY on the local system.

#### **KERRY%**

Sets the node ID to the local node name and the user ID to KERRY, causing the file to be sent to user KERRY on the local system.

#### **KERRY**

Sets the node ID to the local node name and the user ID to KERRY, causing the file to be sent to user KERRY on the local system.

### **@LASER3**

Sets the node ID to LASER3 and the user ID to SYSTEM, causing the file to be sent to the network node LASER3.

### **%LASER2**

Sets the node ID to LASER2 and the user ID to SYSTEM, causing the file to be sent to the network node LASER2.

### **Notes:**

1. UFTXIN will use only the first 32 characters of the user name.
2. If you are using a UFTXIN configuration file, UFTXIN will first look for a user name record with that name, or a record called DEFAULT. See [“UFTXIN Configuration File” on page 259](#). If UFTXIN cannot find a matching user name record (or a DEFAULT record), or if it cannot find the configuration file, it will parse the user name as described above.
3. When parsing the user name, UFTXIN will assume that any user name which does not include the @ or % operator is a user ID.
4. UFTXIN will limit the length of the user ID and node ID to 8 characters each and will discard any extra data in those fields of the user name.

## **Supported UFT Commands**

The UFT commands supported by the UFTXIN exit are:

### **ABORT**

Causes the spool file to be closed and purged.

### **CLASS**

First character of the first operand of the data will be used for the class field of the TAG (TAGCLASS). The second operand, if specified when TYPE is V M (the operand is ignored for all other TYPE values), will indicate the file was from VM spool and whether it was a 1403, 3211, 3800, VAFP, or PUN device.

### **COPY**

Will be used for the copy count field of the TAG (TAGCOPY).

### **DATE**

Will be used to create the file origin TOD value in the TAG (TAGINTOD).

### **DESTination**

First 8 characters of the data will be used for the destination field of the TAG (TAGDEST).

### **DISTRibution**

First 8 characters of the data will be used for the distribution field of the TAG (TAGDIST).

### **EOF**

This causes the SPOOL file to be closed and delivered.

### **FILE**

First 8 characters of the from data will be used for the origin user field of the TAG (TAGINVM).

### **FORM**

First 8 characters of the data will be used for the form field of the TAG (TAGFORMN).

### **NAME**

First 8 characters of the file name and first 8 characters of the file extension will be used for the data set name field of the TAG (TAGDSN). Some parsing will be done to remove dots and slashes. For example, the string:

```
C:\WINDOWS\SYSTEM\AUTOEXEC1234.BAT
```

will become AUTOEXEC.BAT.

### **OWNER**

First 8 characters of the data will be used for the origin user field of the TAG (TAGINVM) overriding the information supplied on the FILE UFT command if provided.



**RECFmt**

Used when creating the INMR2 control record.

**RECLen**

Defines the maximum allowed line width. This will become TAG field TAGRECLN.

**TITLE**

Will be used for the title line on each page if pagination was requested in the matching user name record (or DEFAULT record) defined within the UFTXIN configuration file.

**TYPE**

Determines whether to translate data into EBCDIC and how to spool the data. UFTXIN supports these data types (TYPE codes):

Data	Code	Processing
ASCII	A	Data will be translated to EBCDIC and spooled in NETDATA format. Trailing CR/LF (X'0D0A') characters will be removed.
Binary	I or B	Data will be spooled in NETDATA format.
EBCDIC	E	Data will be spooled in NETDATA format. Trailing NL (X'15') characters will be removed.
Mail	M	Data will be translated to EBCDIC and spooled in NETDATA format.
NETDATA	N	Data will be spooled as received.
Spool	V M	Data will be spooled as is after the CCW opcode and the 2-byte length characters are removed.
Variable	V	Data will be spooled in NETDATA format after the 2-byte length characters are removed.

**USER**

First 17 characters of the data will be used for the destination user field of the TAG (TAGTOVM) and the destination node field of the TAG (TAGTOLOC).

In addition, the first 32 characters of the data will be used to search for a matching user name record within the UFTXIN configuration file.

In addition, the UFTXIN routine supports comment records **\*** and **#**.

**Available EPARM Parameters**

When using the UFTXIN routine, you can specify the following parameter in the EPARM value on the PARM configuration statement for the link or on the RSCS DEFINE or START command.

**Config=ddname**

specifies the DDNAME which has been defined as an exit configuration file. If the DDNAME does not exist, the UFTXIN initialization routine will pass back a return code to cause the UFTD-type link to issue an error message and drain. If this exit parameter is not used, a configuration file is not read by the UFTXIN exit, causing existing defaults to be used for values which can be defined by the configuration file.

**UFTXIN Configuration File**

The UFTXIN sample exit routine can read a configuration file. This configuration file can supply the following:

- Overrides for processing when a file is received from a remote UFT client, based on the user name
- A translate table to override the one used by the exit
- Override the host name used in the herald response message
- Provide a domain name used in the herald response message

The configuration file can have any desired file name and file type and must be on a disk accessed by the RSCS user ID. This file must be defined with a FILEDEF statement in the PROFILE GCS. The DDNAME used must be specified on the Config= parameter in the EPARM value on the PARM statement for the link in the RSCS CONFIG file or in the link operational parameters on the RSCS DEFINE or START command.

The following is an example of a DDNAME entry in the PROFILE GCS in which UFTIN is the defined DDNAME and UFTI SCONFIG is the name of the UFTXIN configuration file:

```
'FILEDEF UFTIN DISK UFTI SCONFIG *'
```

The following is an example of the PARM statement for a UFTD-type link named UFTD using the DDNAME defined on the FILEDEF statement in the PROFILE GCS:

```
PARM UFTD EXIT=UFTXIN EPARM='C=UFTIN'
```

## Layout of the UFTXIN Configuration File

The following rules apply to the UFTXIN configuration file:

- An asterisk (\*) in column one denotes a comment line.
- Any line that does not have an asterisk (\*) in column one will be interpreted as a configuration entry.
- All configuration entries must be capitalized.
- Entries can span multiple records.

The following configuration records are supported.

### **DOMAINNAME=string**

specifies a domain name, up to 255 characters, to be appended after the host name of the positive herald response message (the message sent in response to a connect request from a remote UFT client). A period (.) will be inserted between the host name and domain name. This record can be used to add a domain name after the host name, which by default is the local RSCS node name.

### **HOSTNAME=string**

specifies a host name, up to 255 characters, used within the positive herald response message (the message sent in response to a connect request from a remote UFT client), overriding the default, which is the local RSCS node name.

### **TOASCII=string**

provides a table for EBCDIC to ASCII translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) can be specified on multiple TOASCII= records to replace the 256-byte translation table.

### **TOEBCCMD=string**

provides a table for ASCII to EBCDIC translation of the UFT commands, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) can be specified on multiple TOEBCCMD= records to replace the 256-byte translation table.

### **TOEBCDIC=string**

provides a table for ASCII to EBCDIC translation, overriding the default used by the exit. Up to 512 hexadecimal characters (0 - 9, A - F) can be specified on multiple TOEBCDIC= records to replace the 256-byte translation table.

### **username**

provides the ability to override defaults used by UFTXIN on a user name basis when receiving a file from a remote UFT client. Multiple unique user name records can be specified.

When a user name arrives, UFTXIN will first look for a matching user name record. If none is found, it will look for a record with the name DEFAULT. A DEFAULT record can be used to define parameters for any user name not defined by its own configuration record. If a DEFAULT record is not found, UFTXIN will use the existing defaults.

The format of the user name record is:

```
username ppos lpage class forms jobn dest pagination userid nodeid tcp$lb$in
netdata
```

The following rules apply to the user name record:

- One record is allowed per line; continuation is not supported.
- The parameters of the record must be separated by one or more blanks.
- The parameters are not column dependent but they are position dependent.
- An asterisk (\*) can be used for any parameter except *username* to tell UFTXIN to use the existing default.

The parameters of the user name record are defined as follows.

## DEFAULT

### *username*

is a user name up to 32 characters, or DEFAULT.

UFTXIN parses the user name into a user ID and node ID, as follows:

#### ***userid@nodeid***

Node ID and user ID will be set as specified.

#### ***userid%nodeid***

Node ID and user ID will be set as specified.

#### ***userid@***

Node ID will be set to the local node name, user ID will be set as specified.

#### ***userid%***

Node ID will be set to the local node name, user ID will be set as specified.

#### ***userid***

Node ID will be set to the local node name, user ID will be set as specified.

#### ***@nodeid***

Node ID will be set as specified, user ID will be set to SYSTEM.

#### ***%nodeid***

Node ID will be set as specified, user ID will be set to SYSTEM.

The user ID and node ID parsed from the user name can be overridden within the record. If the *username* value is DEFAULT or any other value that does not conform to one of the variations listed above, the user ID and node ID should be set (overridden) within the record. If both are overridden, the *username* value is not parsed. If neither is overridden, the *username* value is assumed to be a user name, which is parsed into the user ID and node ID, which must be valid (either of which can still be overridden within).

## Notes:

1. When parsing the user name, UFTXIN will assume that any user name which does not include the @ or % operator is a user ID.
2. UFTXIN will limit the length of the user ID and node ID to 8 characters each and will discard any extra data in those fields of the user name.

### *ppos*

is the logical record length. This value can be 1 - 65535 when the file will be spooled in NETDATA format; otherwise, it can be 1 - 32760. The default is 255 if the RECLen UFT command is not received.

If the file is not spooled as NETDATA, then the following virtual printer type will be defined based on the *ppos* value:

<i>ppos</i>	Virtual Printer
1 - 132	1403

<i><b>ppos</b></i>	<b>Virtual Printer</b>
133 - 150	3211
151 - 204	3800
205 - 1280	VAFP

***lpage***

is the number of lines per page, 1 - 99. The default is 66. This parameter is valid only if the pagination parameter has been set to PAGE.

***class***

is the 1-character spool file class. The default is blank if the CLASS UFT command is not received.

***forms***

is the 1- to 8-character spool file form. The default is blank if the FORM UFT command is not received.

***jobn***

is the 1- to 8-character job name. The default is SYSTEM.

***dest***

is the 1- to 8-character PSF destination. The default is blank if the DESTINATION UFT command is not received. When using a PSF destination, UFTXIN will set the user ID field of the TAG to SYSTEM.

When using a PSF destination, the user name can be in the form *SYSTEM@nodeid*. Alternatively, the user name can be anything unique as long as the *userid* parameter in the record is either SYSTEM or \* and the *nodeid* parameter is specified.

***pagination***

specifies how pagination will be performed.

**PAGE**

UFTXIN will paginate.

**NOPAGE**

UFTXIN will not paginate. This is the default.

***userid***

is the 1- to 8-character user ID to which the file should be spooled. The default is derived from the user name.

**Note:** If a PSF destination is provided by the *dest* parameter, then the *userid* parameter can be set only to SYSTEM.

***nodeid***

is the 1- to 8-character destination node ID to which the file should be spooled. The default is derived from the user name.

***tcpxlb***

is the file name of a TCP/IP translation table (TCPXLBIN file) to be used for this user name. The file must exist on any accessed disk. If this parameter is not specified, the translation table used will be either the default table within UFTXIN or the override specified on one or more TOEBCDIC= records in the UFTXIN configuration file.

***netdata***

specifies whether the data should be spooled in NETDATA format.

**YES**

Spools the data in NETDATA format. This is the default.

**NO**

Spools the data as plain records, not in NETDATA format. This is the desired format if the destination of the file is a printer.

**Note:** This option does not apply if the transform type (UFT TYPE command code) is N (NETDATA) or V M (Spool).

### ***User Name Examples***

This example would allow a file to be received without translating, and spooled to an LPR link using the LPRXONE exits, causing the file to be sent unaltered to a remote UFT daemon:

```
LPR@NODEONE 1280 50 * STDN * * * ASCII * * NO
```

This example defines a user name used to spool input to the system printer:

```
LOCALNODE * * * * * SYSTEM * * * NO
```

This example shows the defaults defined within UFTXIN.

```
DEFAULT 255 66 * * SYSTEM * NOPAGE * * * YES
```



---

# Chapter 11. RSCS Macros

This section describes the RSCS macros that you can use with your exit routines. For information about the conventions used in the syntax diagrams, see [“Syntax, Message, and Response Conventions”](#) on page [xv](#).

---

## Specifying Parameters

There are two ways to specify parameters: *positional* and *keyword*. You must write positional parameters in a specific order. If you omit a positional parameter and specify another parameter to its right, you must include a comma before the omitted parameter. You can omit the comma when the positional parameter is followed by a keyword parameter or a blank.

A keyword parameter is immediately followed by an equal sign and an optional variable. You can specify keyword parameters, but you must specify them to the right of any positional parameters in the macro.

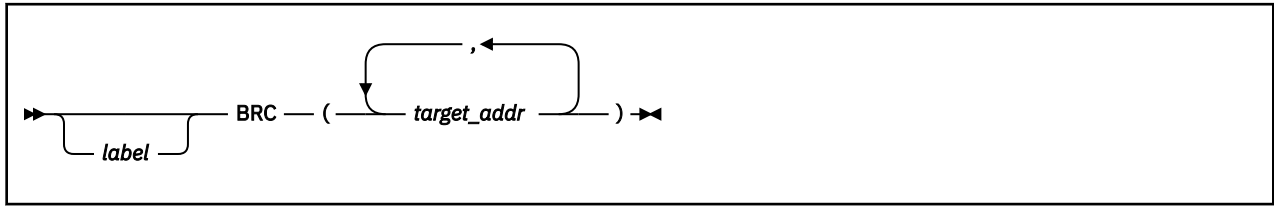
---

## Program Structure Macros

RSCS provides program structure macros to generate standard OS register saving and savearea conventions. The macros also create standard entry and exit conditions in RSCS modules and your exit routines. You can use these macros in reentrant and nonreentrant environments. Reentrant modules can also use these macros to allocate storage for dynamic work areas.

<b>Attention</b>
These macros require knowledge of systems programming and assembler language conventions. The macros do not provide complete syntax checking; some combinations of keywords and parameters are not supported. Please read the parameter descriptions and the usage notes carefully before attempting to use these macros.

## BRC – Branch on Return Code



### Purpose

The BRC macro branches to one of a series of specified target addresses using the return code, passed in register 15, from an exit routine. The exit routine must return standard return code values (multiples of 4, starting with 0). Use this macro after issuing RCALL to call an exit routine.

The BRC macro generates a branch table. For any return code, RSCS processes two branch on condition (BC) instructions.



**Attention:** The IBM High Level Assembler (HLASM) supports the ESA/390 opcode BRC, which conflicts with the use of the RSCS BRC macro. Therefore the BRC assembler instruction cannot be used in parts which use the RSCS BRC macro.

### Parameters

#### *label*

is any valid assembler statement label.

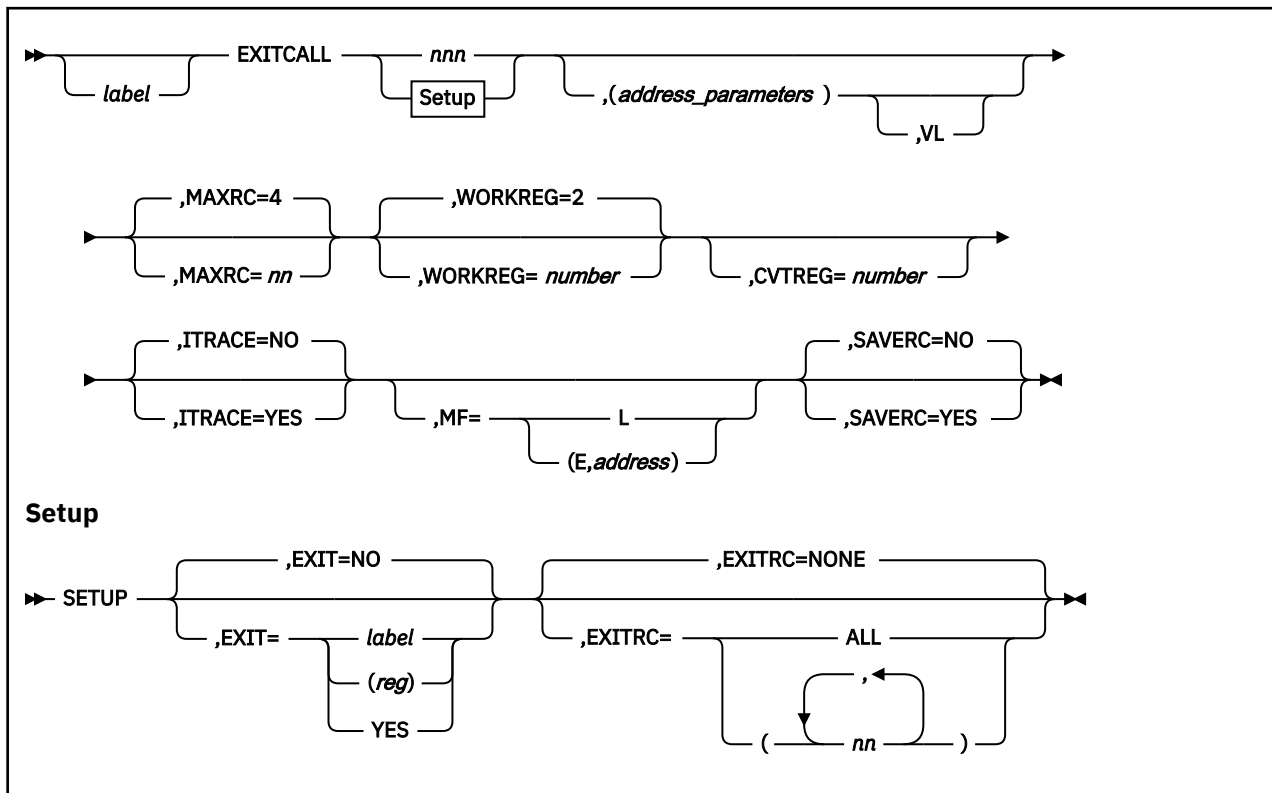
#### *target\_addr*

is the address of a label where control is passed for a return code. If you specify multiple addresses, they must be separated by commas. Each label corresponds to a return code, starting with return code 0 and continuing for the return code sequence (4, 8, 12, and so on). Addresses can be specified as any value that is valid in an RX-type instruction.

To pass control to the next sequential instruction following the BRC macro, specify the value NSI.



## EXITCALL – Providing an Exit Point



### Purpose

The EXITCALL macro passes control to an exit routine, if the exit point has been enabled by the EXIT statement or command. The linkage established is the same as that created by a BASR instruction; the issuing program expects to regain control. The macro allows an address parameter list to be constructed. Standard, list, and execute forms of the EXITCALL macro can be coded.

### Parameters

#### *label*

is any valid assembler statement label.

#### *nnn*

is a number, 0 - 255, that identifies an exit point. You can specify the *nnn* number symbolically or with an absolute expression. If you specify MF=L, you must omit the *nnn* and specify a comma (,) to show its absence.

#### SETUP

identifies special conditions to be processed by the next EXITCALL *nnn* invocation, which must follow immediately after this invocation. When specified, SETUP also initializes the ECXBLOK area, which stores exit routine return codes, to zeros.

#### ,EXIT=

identifies another routine that RSCS calls before calling the next exit routine associated with this exit point. RSCS calls this routine only if the exit routine issues the return code specified on the EXITRC parameter.

#### NO

Tells RSCS to continue its usual processing for this exit point. This is the default.

**label**

Specifies the assembler label of the routine to be called.

**(Rn)**

Specifies that register *n* contains the address of the routine to be called.

**YES**

Tells RSCS to call the next sequential instruction.

**,EXITRC=**

identifies the return code for which RSCS calls the routine specified on the EXIT parameter.

**NONE**

Specifies that RSCS does not call additional routines when the exit routine issues any return code. This is the default.

**ALL**

Specifies that RSCS calls the specified routine when the exit routine issues any valid return code.

**nn**

Specifies a return code for which the routine is called.

**address\_parameters**

specifies one or more addresses, separated by commas, passed to the exit routine. Each address is expanded, in order, to a fullword on a fullword boundary. R1 contains the address of the first parameter. If you do not specify address parameters, the contents of R1 do not change.

If you code the standard form of the macro, you can specify addresses as any valid A-type address constant or as R2 to R12. You can specify registers symbolically or with an absolute expression; they must be coded within parentheses.

If you specify the list form of the macro, you *must* state address parameters. Specify address as any valid A-type address constant, or show their absence with commas.

If you specify the execute form of the macro, you can specify addresses as any value that is valid in an RX-type instruction or as R2 to R12. You can specify registers symbolically or with an absolute expression; they must be coded within parentheses.

**,VL**

sets the high-order bit of the last address parameter in the macro expansion to 1. The exit routine can test this bit to find the end of the parameter list. VL is coded only if address parameters are designated. It should be used only when a variable number of parameters can be passed to the called program.

**,MAXRC=code**

specifies the maximum acceptable return code, *code*, issued by the exit routine. If this parameter is omitted, the default is 4.

**,WORKREG=number**

specifies an internal work register, *number*, destroying the register's original contents; R2 is the default. You can specify registers symbolically or with an absolute expression. Do not specify registers 0, 1, 12, 13, 14, or 15 as a work register. If the module containing this EXITCALL macro uses many base registers, registers 9, 10, and 11 may also be unavailable.

**,CVTREG=number**

specifies a register, *number*, that contains the address of the CVT control block. You can specify registers symbolically or with an absolute expression. The contents of this register are not destroyed, unless register 1, 14, or 15 is specified. Do not specify registers 0, 12, or 13. If the module containing this EXITCALL macro uses many base registers, registers 9, 10, and 11 may also be unavailable. If you do not specify a register, RSCS generates a LOAD instruction and a V-type address constant to get the address of the CVT.

**,ITRACE=**

specifies whether an ITRACE macro is called within the EXITCALL expansion to trace the call to the exit routine.

**NO**

Does not generate an ITRACE macro. This is the default.

**YES**

Generates an ITRACE macro. This option is intended for IBM use only.

**,MF=**

specifies the format of the macro.

**L**

List form.

**(E,address)**

Execute form. The *address* is any valid RX-type address or a register, 1 - 12, that was previously loaded with the specified address. You can specify the register symbolically or with an absolute expression; it must be coded within parentheses. For example, Register 1 must be specified as (1).

**,SAVERC=**

specifies whether the return code from an exit routine should be stored in the ECXBLOK.

**NO**

Does not store the return code. This is the default.

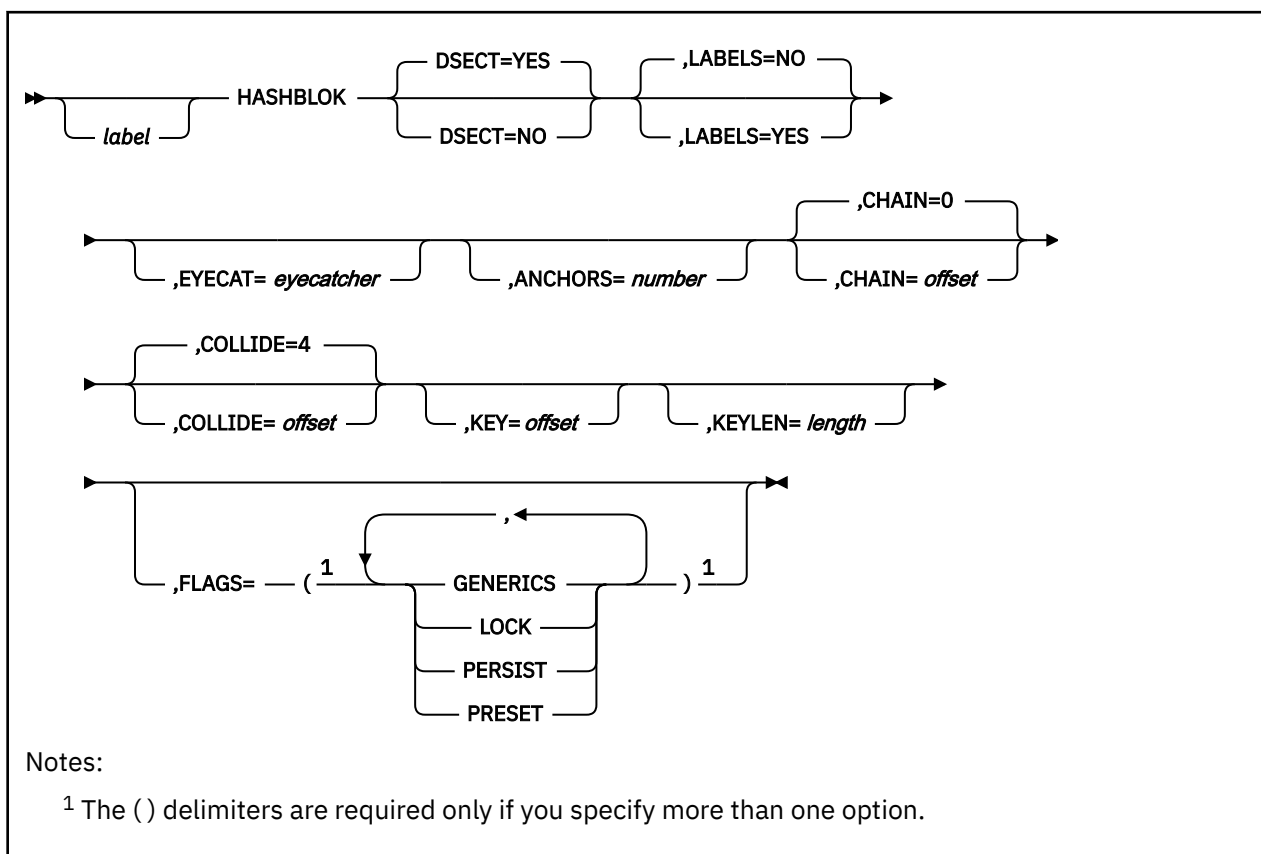
**YES**

Stores the return code.

**Usage Notes**

1. EXITCALL generates literal expressions. You must ensure that literals are not generated outside the boundaries of the CSECT that contains the EXITCALL macro. Use of an assembler LTORG statement is recommended.
2. The EXITCALL macro expansion uses the TEXITS label generated by the CVT; you must code the CVT macro in any module that calls EXITCALL.
3. When the code generated by the EXITCALL macro is processed, the contents of registers 14, 15, and the specified WORKREG register are altered. If you code address parameters, R1, and possibly R0, are also altered.
4. If the exit routine issues a return code that exceeds the MAXRC value or is not a multiple of 4, user abend 1xx is generated (where xx is the number of the exit point in hexadecimal).

## HASHBLOK – Defining a Hash Table



### Purpose

The HASHBLOK macro defines the characteristics of a hash table. A HASHBLOK macro can be used to describe the hashing function to be used for any type of data area.

### Parameters

#### *label*

is any valid assembler statement label.

#### **DSECT=**

specifies whether a DSECT statement is generated.

#### **YES**

Generates a DSECT statement. This is the default.

#### **NO**

Does not generate a DSECT statement. The macro expansion is generated as a continuation of the current CSECT or DSECT.

#### **,LABELS=**

specifies whether assembler labels are generated for each statement in the macro expansion.

#### **NO**

Does not generate labels. This is the default.

#### **YES**

Generates labels.

#### **,EYECAT=***eyecatcher*

specifies a value used to identify the macro invocation within storage.

**,ANCHORS=*number***

specifies the size of the hash index.

**,CHAIN=*offset***

specifies the offset value of the chain pointer within the data area to be hashed. The default is 0.

**,COLLIDE=*offset***

specifies the offset into a data area to be used as a collision chain. The default is 4.

**,KEY=*offset***

specifies the offset into the data area for the hash key, which is used for the hash table index.

**,KEYLEN=*length***

specifies the length of the hash key.

**,FLAGS=**

specifies one or more of the following characteristics for the hash table:

**GENERIC**

The hash table supports generic searches for hash table entries.

**LOCK**

Disables interrupts for other routines when the hash table is being updated.

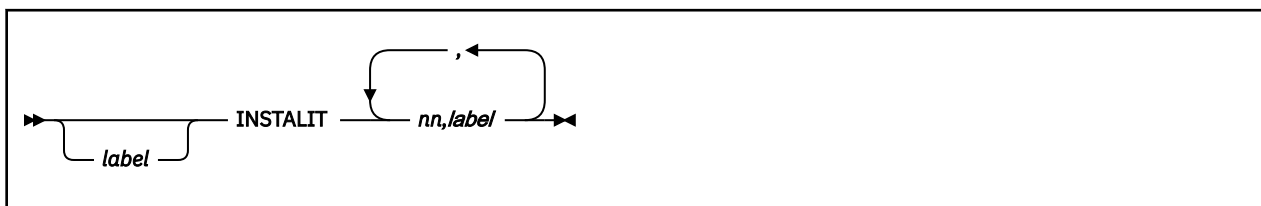
**PERSIST**

Persistent storage creates the hash table. The HASHBLOK remains in storage after the routine that created it is detached.

**PRESET**

The value specified on the ANCHORS parameter is used as the size of the hash table.

## INSTALIT – Adding a Record Format Table



### Purpose

The INSTALIT macro adds an ITRACE record format table to the list of format tables that RSCS recognizes. Do not use this macro to replace any IBM-defined trace types for the ITRACE macro.

### Parameters

#### *label*

is any valid assembler statement label.

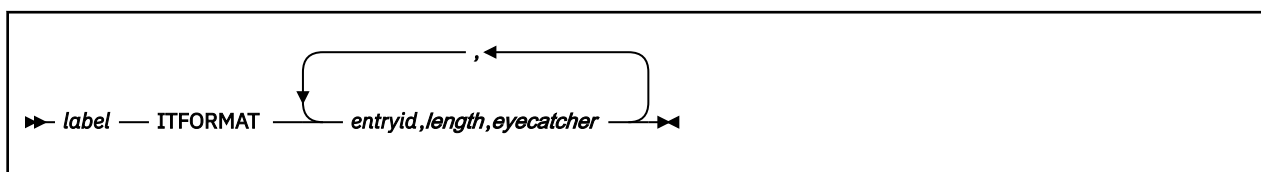
#### *nn*

is the number of the event type for this trace record. If you specify an *nn* value on more than one invocation of the INSTALIT macro, only the last occurrence of that *nn* is added to the table.

#### *label*

is the label specified on the ITFORMAT macro that defines the format of the specified *nn* type.

## ITFORMAT – Building a Format Table



### Purpose

The ITFORMAT macro builds a format table for an internal trace record you have defined in the ITRACE macro. The resulting table must then be added to the list of IBM-defined trace formats by running the INSTALIT macro.

### Parameters

#### *label*

is any valid assembler statement label. You must specify a label when calling this macro.

#### *entryid*

identifies the data area to be placed in the internal trace record.

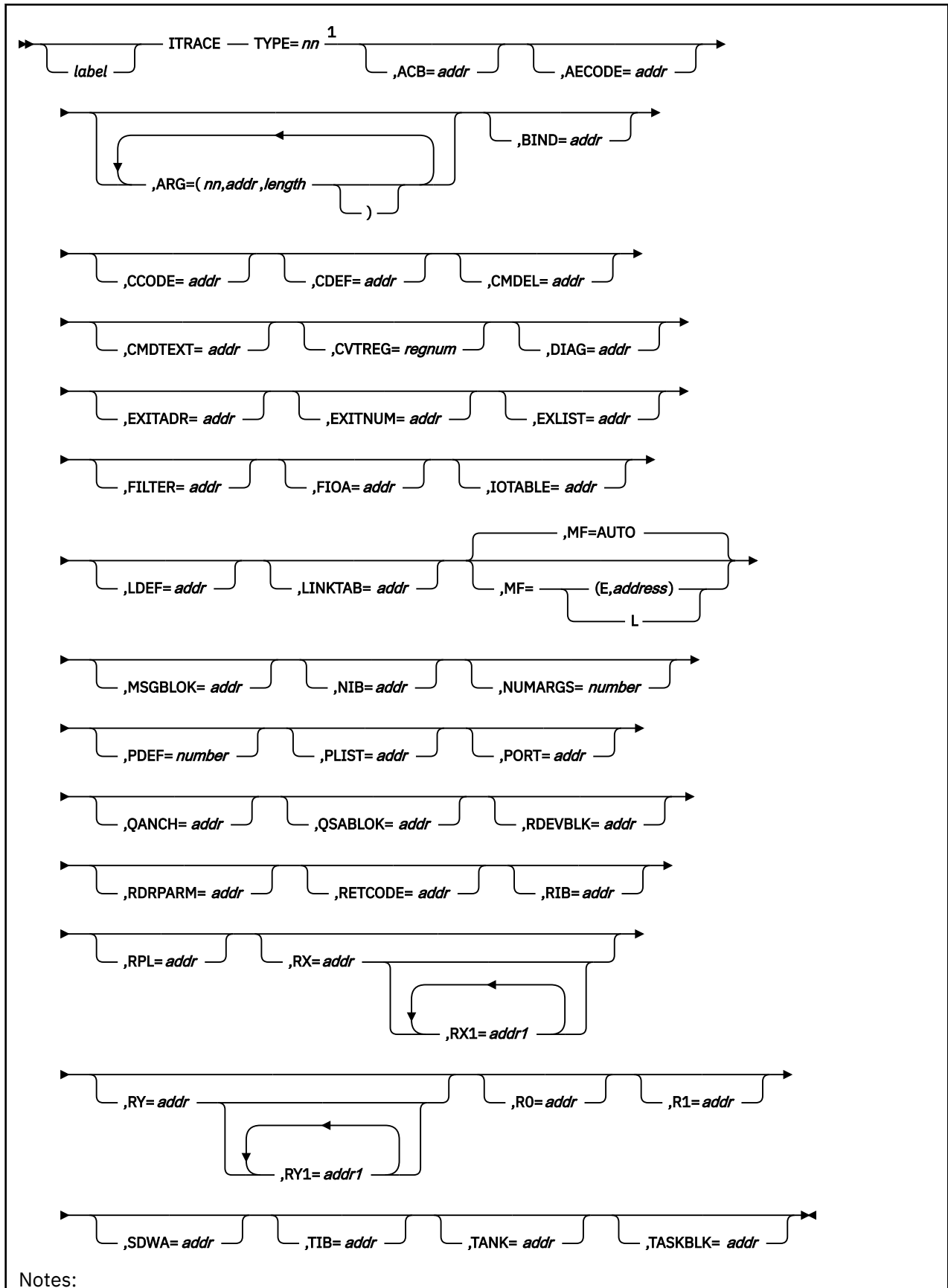
#### *length*

is the length of the data area. This value cannot exceed the size of the internal trace table.

#### *eyecatcher*

is a 10-byte character string that identifies the data area.

## ITRACE – Tracing an Event





<sup>1</sup> TYPE is not required if MF=L is specified.

## Purpose

The ITRACE macro records events that occur in an exit routine and places them in the RSCS trace table.

## Parameters

### ***label***

is any valid assembler statement label.

### **TYPE=*nn***

specifies the format of the trace record for the event being recorded in the RSCS internal trace table. (See usage note [“1” on page 277](#) for the IBM-defined values.) This parameter is not needed for the MF=L form.

### **,ACB=*addr***

specifies a pointer to the ACB control block used by the SNA control task when the RSCS/VTAM interface begins and ends.

### **,AECODE=*addr***

specifies the pointer to the system completion code that identifies the type of program interruption (for example, SOC4).

### **,ARG=(*nn,addr,length*)**

specifies the free-form method of specifying fields and data areas to be placed in a RSCS internal trace record. This is useful when using customer or IBM-defined format tables; however, you must use the correct format for each type of record.

### **,BIND=*addr***

specifies a pointer to the BIND area pointed to by an RPL.

### **,CCODE=*addr***

specifies the pointer to the condition code returned by a diagnose code.

### **,CDEF=*addr***

specifies a pointer to the CDEF control block that defines a command or statement name. It also contains a pointer to the next transition level.

### **,CMDEL=*addr***

specifies a pointer to a command element.

### **,CMDTEXT=*addr***

specifies a pointer to the command or statement text issued.

### **,CVTREG=*regnum***

specifies a register that contains the address of the CVT.

### **,DIAG=*addr***

specifies the pointer to the diagnose code for which a trace entry is to be recorded.

### **,EXITADR=*addr***

specifies a pointer to the exit routine being called.

### **,EXITNUM=*addr***

specifies the pointer to the exit point for which the entry is to be recorded.

### **,EXLIST=*addr***

specifies a pointer to a VTAM EXLST block.

### **,FILTER=*addr***

specifies a pointer to a filter program.

### **,FIOA=*addr***

specifies a pointer to a file I/O area.

**,IOTABLE=addr**

specifies a pointer to an IOTABLE control block.

**,LDEF=addr**

specifies a pointer to an LDEF control block to define the parser transition level for a state while processing a command or statement.

**,LINKTAB=addr**

specifies a pointer to a LINKTABL entry.

**,MF=**

specifies the format of the macro.

**AUTO**

The macro automatically obtains its required work area from the RSCS TASKBLOK associated with this task; the macro expansion generates a call to DMTTASKG. This is the default.

**L**

List form. When you specify this form, you must include a *label*.

**(E,address)**

Execute form. The *address* is either an address that is valid in an RX-type instruction or a register, 1 - 12, that contains the specified address. A register may be specified symbolically or within parentheses.

**,MSGBLOK=addr**

specifies a pointer to the message block for a message.

**,NIB=addr**

specifies a pointer to the Node Initialization Block for a VTAM request.

**,NUMARGS=number**

specifies the number of double words for the macro to generate for the execute form of the macro invocation. This parameter can be specified only on the list form of the macro; it is an alternative to specifying commas for all other parameters.

**,PDEF=addr**

specifies a pointer to a PDEF control block that defines the parser transition level for processing a command or statement.

**,PLIST=addr**

specifies a pointer to an initialization vector passed to a task when it is attached.

**,PORT=addr**

specifies a pointer to a PORT entry of an auto-dial or auto-answer task.

**,QANCH=addr**

specifies the queue anchor for command or message elements that is passed to DMTCOMNQ and DMTCOMDQ.

**,QSABLOK=addr**

specifies a pointer to a QSABLOK data area.

**,RDEVBLK=addr**

specifies a pointer to a spool file request element, which a link driver task passes to DMTAXMRQ to request a file.

**,RDRPARM=addr**

specifies a pointer to the parameter list mapped by the RDR macro.

**,RETCODE=addr**

specifies the pointer to the value of the return code in register 15, generally on completion of a VTAM request or macro.

**,RIB=addr**

specifies a pointer to the Receiving Information Block.

**,RPL=addr**

specifies a pointer to the RPL passed to, or received from, VTAM during processing of VTAM requests issued by a session driver or the RSCS SNA control task.

**,RX=addr,RX1=addr1**

specifies the pointer to the general storage registers that contain addresses or function code passed to a CP DIAGNOSE code; the registers may also contain a return code from the DIAGNOSE function.

**,RY=addr,RY1=addr1**

specifies the pointer to the general storage registers that contain addresses or function code passed to a CP DIAGNOSE code; the registers may also contain a return code from the DIAGNOSE function.

**,R0=addr**

specifies the pointer to the value of register 0, generally on completion of a VTAM request or macro.

**,R1=addr**

specifies the pointer to the value of register 1, generally on completion of a call to a subroutine, that returns a pointer to a data area.

**,SDWA=addr**

specifies the pointer to the system diagnostic work area provided by GCS when an abend occurs.

**,TIB=addr**

specifies a pointer to a Transmission Information Block.

**,TANK=addr**

specifies a pointer to a holding area for NJE buffers that are not compressed.

**,TASKBLK=addr**

specifies a pointer to a TASKBLOK data area.

## Usage Notes

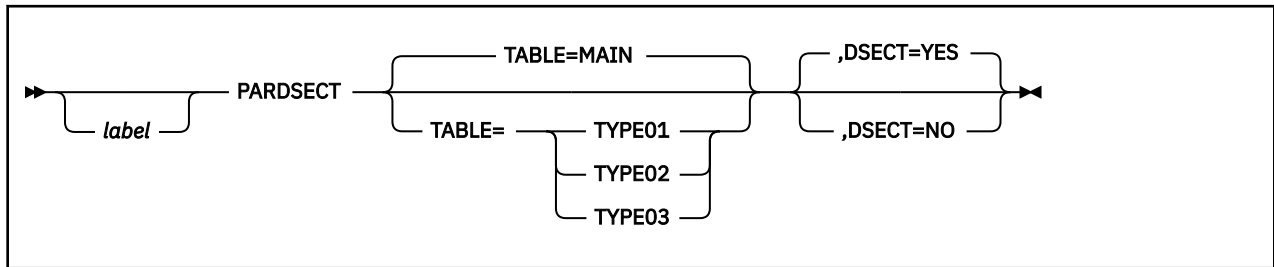
1. You can specify the following IBM-defined values on the TYPE=*nn* parameter:

<i>nn</i>	Trace Record
00	Start of a task
01	End of a task
02	Call to a GCS exit routine
03	I/O interrupt
04	Call to DMTAXMRQ (request a file)
05	Call to DMTCOMNQ (add an element to a queue)
06	Call to DMTCOMDQ (remove a queue element)
07	Call to DMTMGXEP (issue a message)
08	Call to allocate storage
09	Command entry
0B	Initial call to VTAM
0C	Completion of VTAM call
0D	Call to an exit (EXITCALL)
0E	Exit point return
0F	Call to DMTRDREP
10	Call to DMTUROEP or DMTUROFL
11	Call to DMTIOTHD or DMTIOTST
12	NJE record sent
13	A scheduled event
14	Data buffers sent and received

<i>nn</i>	Trace Record
15	Diagnosis code
16	VTAM exit routine
17	NJE job header
18	NJE data set header
19	NJE job trailer
1A	NJE request to open
1B	NJE permission granted
1C	NJE end of file
1D	NJE receiver online
1E	NJE reject
1F	NJEabend
20	NJE connection, RJE connection, or sign-on record
21	Used for exit routines
22	Call to deallocate storage
23	NJE record received
24	IUCV calls to TCP/IP
25	Normal IUCV completion from TCP/IP
26	IUCV immediate error returns
27	Send and receive buffer trace for MRJE
28 - EF	Reserved for IBM use
24 - DF	Not used
E0 - EF	Reserved for IBM use
F0 - FF	Reserved for customer use

2. If your exit routine is reentrant, you must specify MF=L and NUMARGS=*nn* to ensure that the parameter lists for tasks are not overlaid when the ITRACE macro is called.
3. ITRACE does not check if the *nn* value specified on the TYPE parameter represents a defined trace record type. If you specify an undefined TYPE value, ITRACE does not create a trace record.

## PARDSECT – Defining a Keyword Table



### Purpose

The PARDSECT macro creates a keyword table. This keyword table maps the entries generated by the PARKEY macro. You can place the PARDSECT macro anywhere before the end of the module that is using the PAREND, PARKEY, and PAROPT macros. See [Figure 31 on page 283](#) for an example of using the PARDSECT, PAREND, PARKEY, and PAROPT macros.

### Parameters

#### *label*

is any valid assembler statement label.

#### **TABLE=**

specifies the type of table to be used. See the usage notes below for examples of what DSECT each type of table generates.

##### **MAIN**

Creates a main keyword table. This is the default.

##### **TYPE01**

Creates a character keyword table.

##### **TYPE02**

Creates a subkeyword table.

##### **TYPE03**

Creates a number keyword table.

#### **,DSECT=**

specifies whether a DSECT statement is generated.

##### **YES**

Generates a DSECT statement. This is the default.

##### **NO**

Does not generate a DSECT statement. The macro expansion is generated as a continuation of the current CSECT or DSECT.

### Usage Notes

1. If you specify TABLE=MAIN, RSCS generates the following DSECT:

```
PARKWORD DC CL8' ' VALID KEYWORD NAME.
PAROPADR DC A(0) ADDRESS FOR KEYWORD SPECIFICATION
PARTAADR DC A(0) ADDRESS FOR RESULT TO BE RETURNED
PARPRADR DC A(0) ADDRESS OF EXTERNAL PROC. ROUTINE
PARKYLEN DC AL1(0) MIN. LENGTH FOR THAT KEYWORD.
PARKYTYP DC X'00' KEYWORD TYPE X'01',X'02' OR X'03'
PARKYDUP DC X'00' X'MN', DUPLICATE/CONFLICT INDICATOR
PARESERY DC X'00' RESERVED BYTE
PARKELN EQU *-&LABEL MAIN KEYWORD ENTRY LENGTH
```

2. If you specify TABLE=TYPE01, RSCS generates the following DSECT:

PAROLENL	DC	F'0'	MINIMUM LENGTH OF OPTION STRING
PAROLENH	DC	F'0'	MAXIMUM LENGTH OF OPTION STRING
PAROLEN1	EQU	*-&LABEL	TYPE01 KEYWORD OPTION ENTRY LENGTH

3. If you specify TABLE=TYPE02, RSCS generates the following DSECT:

PARSUBKY	DC	CL8' '	SUB-KEYWORD NAME
PARSUBKL	DC	AL1(0)	MINIMUM LENGTH OF THE OPTION
PARSOPCD	DC	X'00'	CODE FOR THIS OPTION
PARSNOPC	DC	X'00'	ANTI-CODE FOR THIS OPTION
PAROLEN2	EQU	*-&LABEL	TYPE02 KEYWORD OPTION ENTRY LENGTH

4. If you specify TABLE=TYPE03, RSCS generates the following DSECT:

PARNUMTP	DC	CL2' '	DC, DD, HC OR HD
PARVCOUN	DC	H'0'	COUNT OF 'V' VALUES
PARVENT	DS	0F	VALUE ENTRY FOLLOWS

## PAREND – Defining the End of a Keyword Table

---



### Purpose

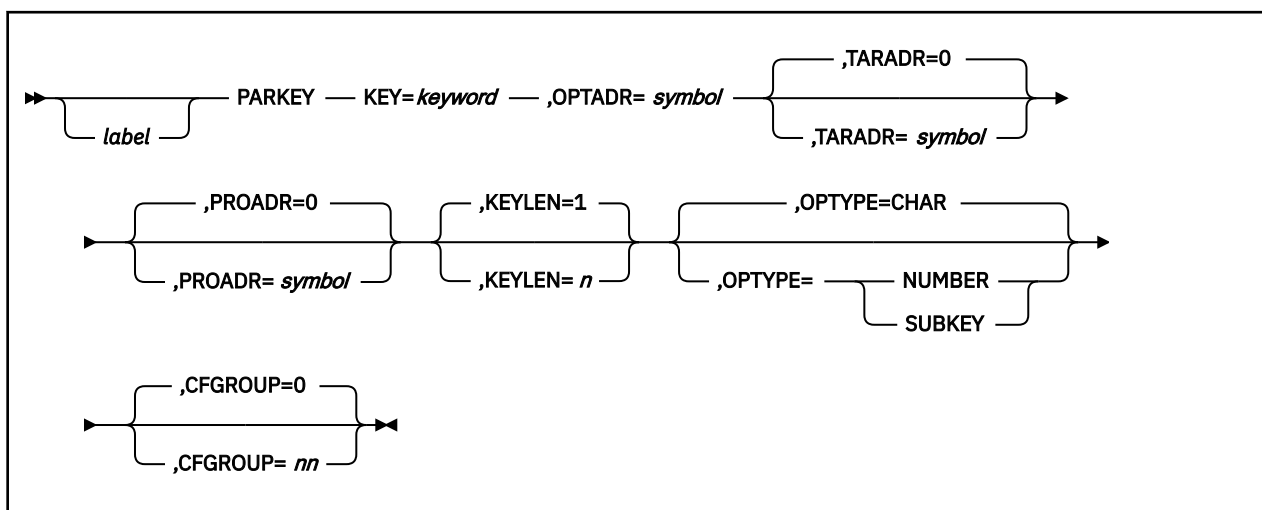
The PAREND macro defines the end of a keyword table. You must specify PAREND after the last use of the PARKEY macro. See [Figure 31 on page 283](#) for an example of using the PARDSECT, PAREND, PARKEY, and PAROPT macros.

### Parameters

#### *label*

is any valid assembler statement label.

## PARKEY – Defining a Keyword



### Purpose

The PARKEY macro defines the characteristics of a keyword. You can specify the PARKEY macro only after you have specified all the PAROPT macros. See [Figure 31 on page 283](#) for an example of using the PARDSECT, PAREND, PARKEY, and PAROPT macros.

### Parameters

#### **label**

is any valid assembler statement label.

#### **KEY=keyword**

specifies the 1- to 8-character alphanumeric keyword. This parameter is required.

#### **,OPTADR=symbol**

specifies the label of the PAROPT macro that describes this option. This parameter is required.

#### **,TARADR=0**

#### **,TARADR=symbol**

specifies the target address or label of an address constant (ADCON) where RSCS will store the result of the parsing of this keyword. If this parameter is omitted, RSCS assumes that TARADR=0, which means there is no result, just a return code.

#### **,PROADR=0**

#### **,PROADR=symbol**

specifies the address of the external processing routine of this option. If this parameter is omitted, RSCS assumes that PROADR=0, which means there is no external processing routine.

#### **,KEYLEN=n**

specifies the number of characters in the minimum abbreviation for the keyword. This value can be any decimal number, 1 - 8. The default is 1.

#### **,OPTYPE=**

specifies the type of option.

#### **CHAR**

The option is character data. This is the default.

#### **NUMBER**

The option is a number.

#### **SUBKEY**

The option is a subkeyword.



**,CFGROUP=nn**

specifies the conflict group for keywords or options that should not be specified together. This value can be any decimal number, 0 - 15. The default is 0, which means there are no conflict groups.

**Examples**

Figure 31 on page 283 shows an example of how to define keywords with the PARDSECT, PAREND, PARKEY, and PAROPT macros.

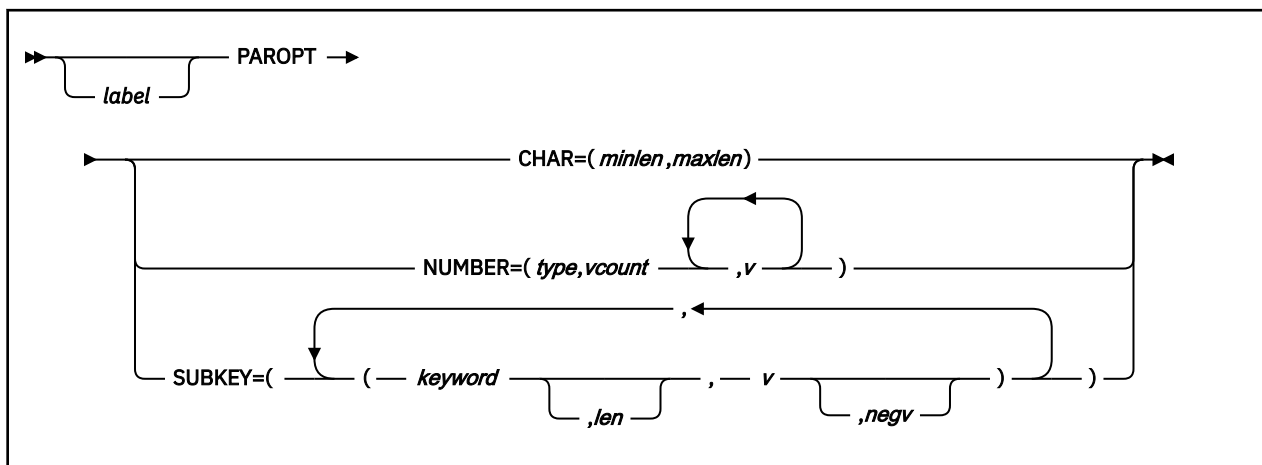
```

EXMOD      RMOD
*****
*
*  MODULE NAME -          *-----*
*                      *      EXMOD      *
*                      *-----*
*
*
EXMODEP    RENTRY  WORK=YES,NUMSAVE=16,RENT=YES,WRKBASE=10,          X
                BASES=2,ARGBASE=R2,                                X
                ARGVS=(@CVT,@PARML,@PARMS,@LINKTAB,@DPARML,@DPARMS)
*
*-----*
*      PAROPT DEFINITIONS
*      TRANS KEYWORD (TRKEY) VALID OPTIONS (EXTRANR)
*      ITO KEYWORD  (ITOKEY) VALID VALUES (EXITO)
*-----*
EXTRANR    PAROPT  SUBKEY=((APL,3,X'80',X'A1'),                      X
                (TEXT,4,X'40',X'61'),                                X
                (DBCS,4,X'10',X'31'),                                X
                (DBCSAPL,7,X'08',X'29'),                             X
                (DBCSTEXT,8,X'04',X'25'),                             X
                (ASISCC,6,X'02',X'2B'))
*
EXITO      PAROPT  NUMBER=(DC,2,0,100)
*
*-----*
*      TARGETS FOR:
*      ITOKEY (PARKEY) (EXITO OPTIONS)
*      TRKEY  (PARKEY) (EXTRANR OPTIONS)
*-----*
ITOVAL     DC      F'-1'          ITO PARM VALUE
*
EXTRAN     DC      AL1(0)         TERMINAL DEVICE TRANSLATION
*
*-----*
*      PARKEY DEFINITIONS
*-----*
TRKEY      PARKEY  KEY=TRANS,          X
                KEYLEN=2,              X
                TARADR=EXTRAN,         X
                OPTADR=EXTRANR,        X
                OPTYPE=SUBKEY
*
ITOKEY     PARKEY  KEY=ITO,            X
                KEYLEN=3,              X
                TARADR=ITOVAL,         X
                OPTADR=EXITO,          X
                OPTYPE=NUMBER
*
PAREND
*
*-----*
*      PARDSECT
*-----*
EXPARS     PARDSECT  TABLE=MAIN
END

```

Figure 31. Defining Keywords

## PAROPT – Defining Options for a Keyword



### Purpose

The PAROPT macro defines the valid options for a keyword and starts the keyword table. You must specify the PAROPT macro before specifying any PARKEY macros. See [Figure 31 on page 283](#) for an example of using the PARDSECT, PAREND, PARKEY, and PAROPT macros.

### Parameters

#### *label*

is any valid assembler statement label.

#### **CHAR=(minlen,maxlen)**

specifies the minimum and maximum lengths of a character option.

#### **NUMBER=(type,vcount,v)**

specifies the characteristics of a number keyword.

##### *type*

specifies the type of number to be used.

##### **DC**

Decimal and continuous

##### **DD**

Decimal and discrete

##### **HC**

Hexadecimal and continuous

##### **HD**

Hexadecimal and discrete

##### *vcount*

specifies the number of values to follow.

##### *v*

is a number value. For discrete numbers, you specify a list of individual numbers. For continuous numbers, you must specify one or two pairs of numbers to indicate one or two ranges of numbers. If you specify one pair of numbers (*v1*,*v2*), *v1* is the minimum number in the range and *v2* is the maximum number in the range. If you specify two pairs of numbers (*v1*,*v2*,*v3*,*v4*), *v1* and *v2* are the minimum and maximum numbers in the first range, and *v3* and *v4* are the minimum and maximum numbers in the second range.

#### **SUBKEY=**

specifies the characteristics of each subkeyword.

***keyword***

is the name of the subkeyword.

***len***

is the length of the minimum abbreviation for the subkeyword.

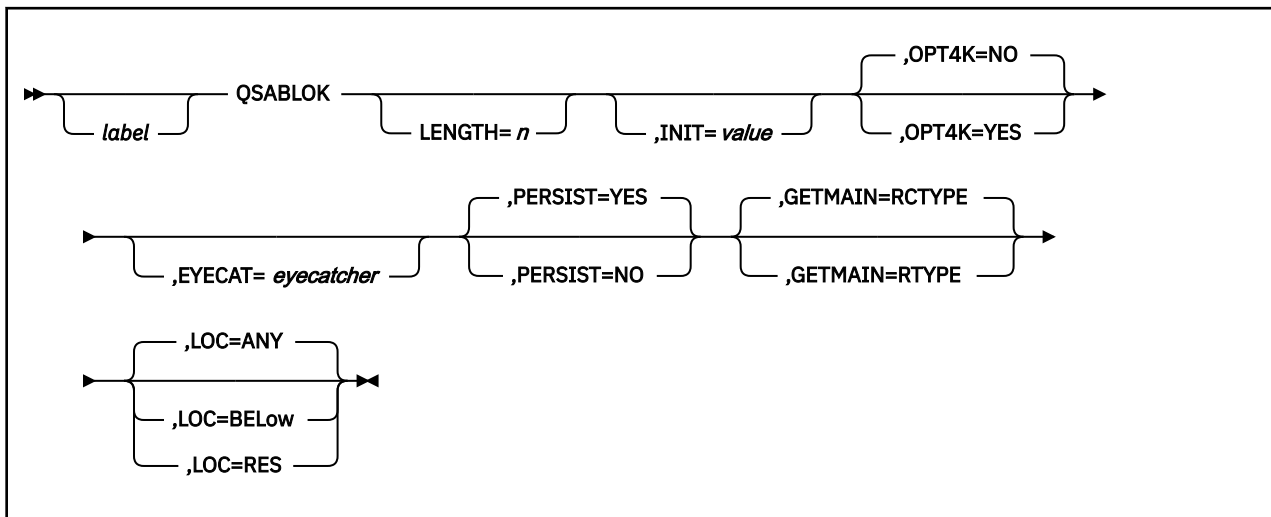
***v***

is the positive value flag for the subkeyword.

***negv***

is the negative value flag for the subkeyword.

## QSABLOK – Defining a Storage Request



### Purpose

The QSABLOK macro defines the characteristics of the storage that your exit routine requires.

### Parameters

#### *label*

is any valid assembler statement label.

#### **LENGTH=*n***

specifies the length of the required piece of storage. If the LENGTH value is greater than the INIT value, the amount of storage specified by the INIT value is initialized as requested. The remaining LENGTH value is then initialized to zeros.

#### **,INIT=*value***

specifies the number of bytes of storage to be initialized. The *value* can be specified as a number or as an equate symbol of a data area from which you want to initialize storage. If specifying an equate, the associated data area must follow the QSABLOK invocation. If you specify 0, the storage area is initialized to zeros.

#### **,OPT4K=**

specifies whether RSCS should acquire a page of storage.

##### **NO**

RSCS acquires only the amount of storage specified on the LENGTH parameter. This is the default.

##### **YES**

RSCS acquires a page of storage and subdivides it as needed.

#### **,EYECAT=*eyecatcher***

specifies a value that identifies the macro invocation within storage.

#### **,PERSIST=**

specifies the type of GCS subpool from which the storage should be acquired.

##### **YES**

The storage is acquired from a persistent subpool. This is the default.

##### **NO**

The storage is acquired from a nonpersistent subpool. This storage will be acquired from subpool 0. It will automatically be released when the task that initially acquired the storage terminates.

#### **,GETMAIN=**

specifies the type of GETMAIN macro to be issued to acquire the storage.

**RCTYPE**

The storage is acquired by a conditional GETMAIN request. If the storage cannot be acquired, the exit routine will receive a return code. This is the default.

**RTYPE**

The storage is acquired by an unconditional GETMAIN request. If the request fails, the calling task will abend.

**,LOC=**

specifies the location of the requested block of storage.

**ANY**

The requested storage can be located anywhere. This is the default.

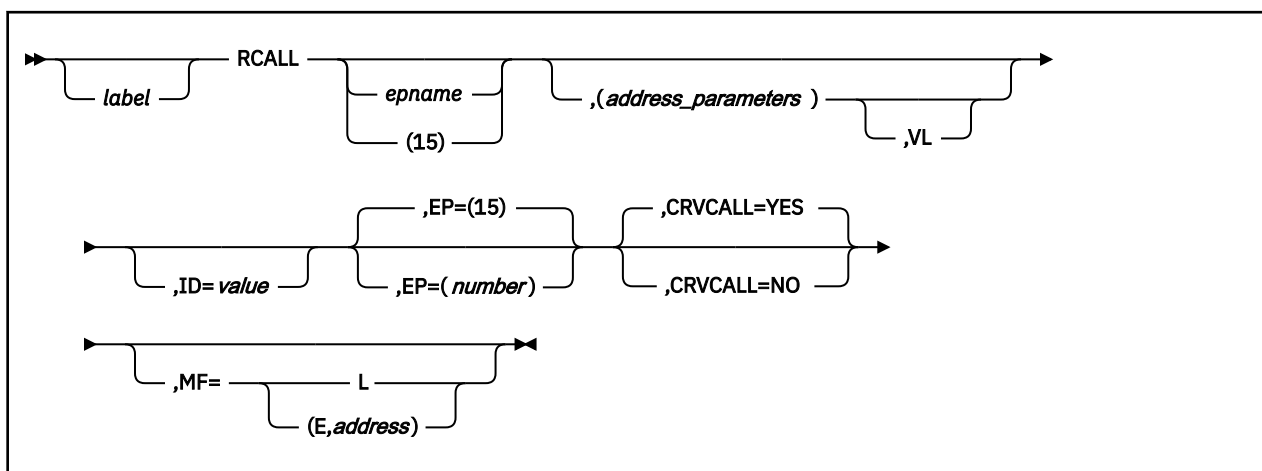
**BELOW**

The requested storage is to be allocated entirely below 16 MB.

**RES**

The location of the virtual storage requested is to be allocated based on the location of the requesting routine. If the requesting routine resides below 16 MB, the virtual storage will be allocated below 16 MB. If the requesting routine resides above 16 MB, virtual storage may be located anywhere.

## RCALL – Passing Control to a Routine



### Purpose

The RCALL macro passes control to a specified routine. The linkage established is the same as that created by a BASR instruction; the calling routine expects to regain control. An address parameter list can be constructed and a calling sequence identifier can be provided. You can code standard, list, and execute forms of the RCALL macro.

### Parameters

#### *label*

is any valid assembler statement label.

#### *epname*

is the name of the entry point that is given control. If the *epname* begins with DMT, the name is used in the macro as the parameter of a V-type address constant; otherwise, the name is used in the macro as the parameter of an A-type address constant. You can also specify the *epname* as (15), if register 15 contains the address of the entry point to be given control.

If you specify MF=L, you must omit the *epname* and specify a comma to show its absence.

#### *address\_parameters*

specifies one or more addresses, separated by commas, to be passed to the called routine. Each address is expanded, in the order designated, to a fullword on a fullword boundary. When control is passed, R1 contains the address of the first parameter. If no address parameters are specified, the contents of R1 are unchanged.

If you code the standard form of RCALL, you can specify addresses as A-type address constants or as registers 2 - 12. You can specify registers symbolically or with an absolute expression; they must be coded within parentheses.

If you code the list form of RCALL, you must specify address parameters. You can specify addresses as any valid A-type address constant, or show their absence with commas.

If you specify the execute form of RCALL, you can specify addresses as any value that is valid in an RX-type instruction or as registers 2 - 12. You can specify registers symbolically or with an absolute expression; they must be coded within parentheses.

#### *,VL*

sets the high-order bit of the last address parameter in the macro expansion to 1; the routine can test this bit to find the end of the list. VL is specified only if address parameters are designated. Use this option only if the number of parameters passed to the routine varies.

**,ID=value**

specifies a unique value, inserted in the macro expansion, used to find information in a dump. The *value* is a symbol or number with a maximum value of 4095. The last fullword of the macro is a NOP instruction containing the ID value in the low-order 2 bytes. When the routine is called, R14 contains the address of this instruction.

**,EP=(number)**

specifies a register that contains the address of the entry point to receive control. This must be the same register specified on the EP parameter of the RENTRY macro for that entry point; register 15 is the default.

You can specify other registers, except 0, 12, 13, or 14, only when calling an internal entry point. If the module containing this RCALL macro uses many base registers, registers 9, 10, and 11 may be unavailable. You can specify this register symbolically or with an absolute expression; it must be coded within parentheses.

**,CRVCALL=**

specifies whether this routine should use the CRV to call an exit routine whose name begins with DMT. Use this parameter only in an exit routine whose module name also begins with DMT.

**YES**

The CRV finds the addresses of other routines. This is the default.

**NO**

The CRV is not used to access other routines; you must use V-constants to access other routines.

**,MF=**

specifies the format of the macro.

**L**

List form.

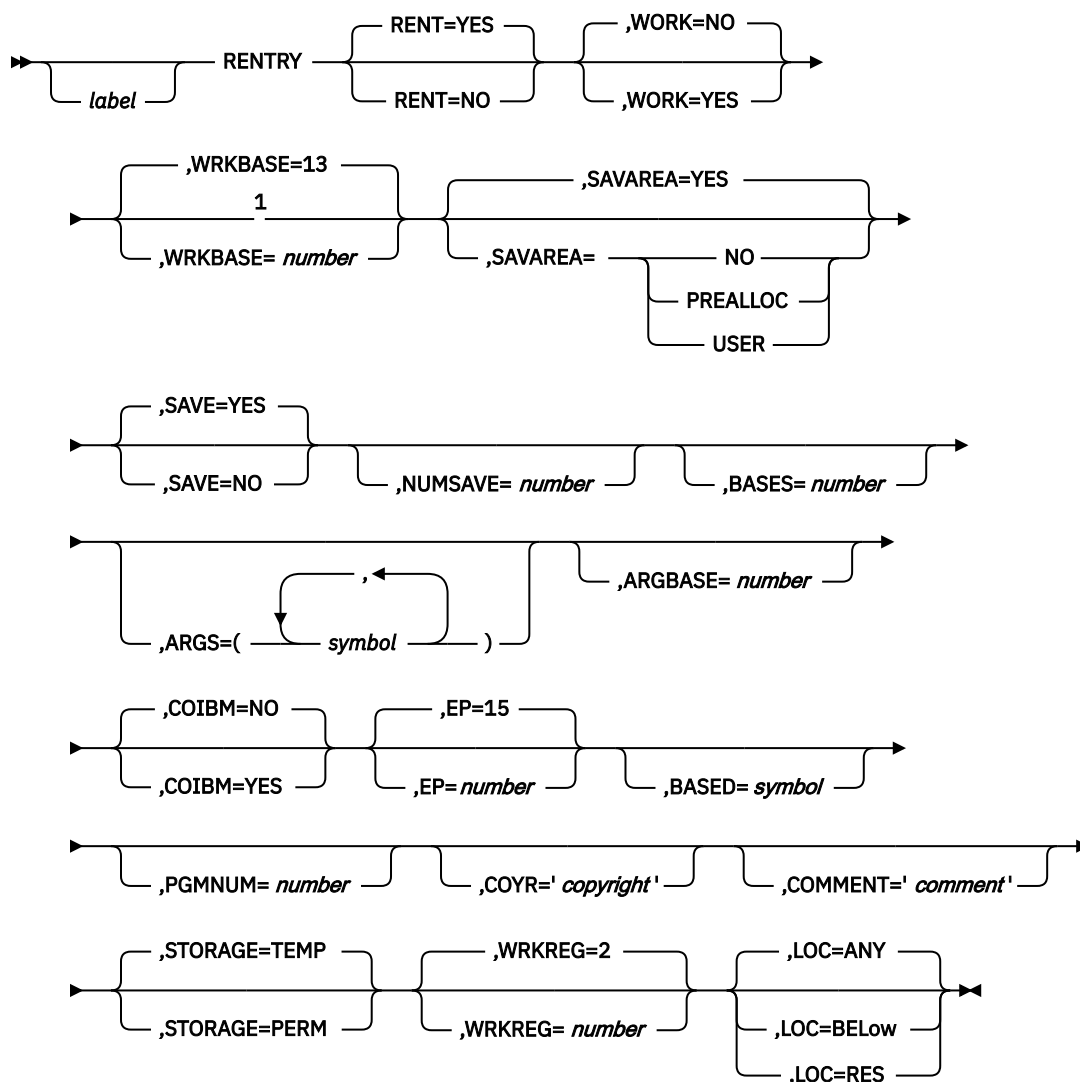
**(E,address)**

Execute form. The *address* is any address that is valid in an RX-type instruction, or a register, 1 - 12, previously loaded with the address. You can specify the register symbolically or with an absolute expression; it must be coded within parentheses.

**Usage Notes**

1. The RCALL macro generates literal statements. You must ensure that literals are not generated outside the boundaries of the CSECT containing the RCALL macro; use of an assembler LTORG statement is recommended.
2. If you specify RCALL in an entry point that specifies SAVAREA=NO on its RENTRY macro, an error will occur.
3. To access an entry point that begins with DMT but does not reside in the RSCS LOADLIB, you should specify CRVCALL=NO. You should then ensure that you establish proper V-constant linkage to that routine.

## RENTY – Defining a Module Entry Point



### Notes:

<sup>1</sup> If WRKBASE is not specified and WORK=YES is specified, register 13 is used.

### Purpose

The RENTRY macro defines a module entry point and generates the following entry point instructions:

- An ENTRY statement
- Entry point label
- Machine-readable eye catcher
- Base register definitions (USING statement and instructions to load the base registers)
- Register saveareas (inline or obtained dynamically for reentrant code)
- Standard OS register and savearea conventions



## Parameters

### *label*

is the entry point name. The format of the name determines the type of entry point code that is generated:

- If the first 6 characters of the entry point name match the module name specified on the RMOD macro, RENTRY generates an *external* entry point. This entry point is made known externally to the module by use of an assembler ENTRY statement. The addressability that is generated uses the first address of the module as the start of the domain of the first base register.
- If the first 6 characters of the entry point name do not match those specified on the RMOD macro, RENTRY generates an *internal* or local entry point. This entry point can be called only from within the module. The addressability that is generated uses the entry point address as the start of the domain of the base register.

### RENT=

specifies whether the generated entry point code is reentrant.

#### YES

The entry point code is reentrant. This is the default.

**Note:** The parameters you specify on the RENTRY macro have no connection to the load module attributes. Thus, if you specify RENT=YES on the RENTRY macro, it has no effect on the link-edit process. For more information about link-editing, see [“Link-Editing Considerations” on page 33](#).

#### NO

The entry point code is nonreentrant.

### ,WORK=

specifies whether the entry point automatically obtains a dynamic work area.

#### NO

Does not obtain a dynamic work area. This is the default.

#### YES

Obtains a dynamic work area for this entry point. You must also specify SAVEAREA=YES on this RENTRY macro and include RWORK and RWORKEND macros in this module (see [“RWORK – Defining the Start of a Module Work Area” on page 302](#)). Each entry point must provide code to obtain addressability to the dynamic work area. You can specify this option only once within a module.

### ,WRKBASE=*number*

specifies a register that provides addressability for the work area defined by the RWORK macro. Any register can be specified, except 0, 12, 14, or 15. If multiple base registers were specified by the BASES parameter, registers 9, 10, and 11 may not be available for use. If this parameter is not specified and WORK=YES is specified, register 13 is used.

### ,SAVEAREA=

specifies whether a register savearea is generated for this entry point.

#### YES

Generates a savearea. This is the default. You must specify this option if you also specified WORK=YES.

#### NO

Does not generate a savearea. If you specify SAVAREA=NO, this entry point cannot call any other entry point using standard linkage conventions.

### PREALLOC

Specifies that the caller has provided a savearea for use by this entry point at 26 fullwords past the address in register 13.

**Note:** You must specify SAVAREA=PREALLOC in exit routines for numbered exits.

**USER**

Specifies that this module will provide code in the entry point to obtain a savearea to store the caller's registers. This code must immediately follow the RENTRY statement.

**,SAVE=**

specifies whether the caller's registers are saved.

**YES**

The registers are saved. This is the default. The caller must provide the address of a 26-fullword savearea in register 13.

**NO**

Does not save the registers.

**,NUMSAVE=number**

specifies the number of 26-fullword saveareas that are generated. If this parameter is omitted, one savearea is generated.

**,BASES=number**

specifies the number of base registers defined for this entry point. The *number* value must be 1 - 4. If this parameter is omitted, one base register is defined. The first (or only) base register defined is register 12. Additional base registers are defined in descending numeric order. This parameter is ignored for an internal entry point; only register 12 is defined for use as a base register.

**,ARGS=(symbol)**

specifies one or more symbolic names, separated by commas, that are associated with any parameter list provided by the caller. This parameter list is in the format generated by the OS CALL, LINK, or ATTACH macros. These macros generate a DSECT that can be used to obtain the caller's parameters. Addressability for the DSECT containing the symbolic names is provided by the register specified by the ARGBASE parameter.

**,ARGBASE=number**

specifies a register that provides addressability for the symbolic names specified by the ARGS parameter. The register is specified by a *number* value; R1 is the default. You can specify any register except 0, 12, 13, 14, or 15. If multiple base registers are specified by the BASES parameter, registers 9, 10, and 11 might not be available.

**,COIBM=**

specifies whether the machine-readable IBM copyright notification is generated.

**NO**

Does not generate the copyright. This is the default.

**YES**

Generates the copyright. COIBM=YES is intended for IBM use only and should be specified only for modules that are not part of the RSCS load module (for example, RSCS modules called by the Dump Viewing Facility).

**,EP=number**

specifies a register containing the address of the entry point when this entry point is given control. This register must be loaded by the caller. You can specify any register except 0, 12, 13, or 14. If the module containing this RENTRY macro uses multiple base registers, registers 9, 10, and 11 might not be available. Only internal entry points can use a register other than register 15. If this parameter is not specified, register 15 is assumed.

**,BASED=symbol**

specifies the name of a previously generated internal entry point on which addressability for the current entry point is based. Because it lets several entry points share common code, you should specify this option only on a RENTRY macro that defines an internal entry point.

**,PGMNUM=number**

specifies the program number for the machine-readable copyright notice that is generated when COIBM=YES is specified. The default value is the program number of the current RSCS function level. This option is intended for IBM use only.

**,COYR='copyright'**

specifies the copyright year (or years) in the machine-readable copyright notification generated by COIBM=YES. The *copyright* value must be enclosed within single quotation marks. The default value is '1979, *year*', where *year* is the year of the general availability of the current RSCS function level. This option is intended for IBM use only.

**,COMMENT='comment'**

specifies a character string that is generated after the machine-readable eye catcher. This option is ignored if you specify COIBM=YES. The *comment* string must be enclosed in single quotation marks.

**,STORAGE=**

specifies whether any dynamically obtained storage is returned when the REXIT macro is issued to return control to the caller.

**TEMP**

Returns dynamically obtained storage. This is the default.

**PERM**

Does not return dynamically obtained storage.

**,WRKREG=number**

specifies an internal work register, destroying the register's original contents. The register may be specified symbolically or by an absolute expression. Do not specify registers 0, 1, 12, 13, 14, or 15 as a work register. This register is used only if SAVE=YES and SAVAREA=NONE are also specified. If you do not specify a register, R2 is the default.

**,LOC=**

specifies the location of the requested block of storage.

**ANY**

The requested storage can be located anywhere. This is the default.

**BELOW**

The requested storage is to be allocated entirely below 16 MB.

**RES**

The location of the virtual storage requested is to be allocated based on the location of the requesting routine. If the requesting routine resides below 16 MB, the virtual storage will be allocated below 16 MB. If the requesting routine resides above 16 MB, virtual storage may be located anywhere.

## Usage Notes

1. The machine-readable eye catcher contains the entry point name, as specified on the *label* of the RENTRY macro.
2. RENT=YES generates a re-entrant entry point. If you specify SAVAREA=YES, RSCS issues a GETMAIN macro to obtain any necessary register saveareas and your exit routine must manage the savearea chain. If you specify SAVAREA=PREALLOC, RSCS uses a savearea that was preallocated by the calling routine and RSCS will manage the save area chain for you.  
  
RENT=NO generates nonreentrant entry point code. The register savearea is generated in-line in the macro expansion.
3. You cannot specify the same register for the WRKBASE and ARGBASE parameters.
4. Issuing supervisor macros or calls to other programs will destroy the contents of R1; use caution if specifying this register for WRKBASE or ARGBASE.
5. If a dynamic work area is specified, it is allocated contiguous with any register saveareas. If register 13 is specified for WRKBASE, it points to the first register savearea. If you specify another register for WRKBASE, that register points to the actual work area; register 13 always points to the register savearea. To make data in a dynamic work area available to the called routine, you must not specify register 13 on WRKBASE.
6. A NUMSAVE value greater than 1 preallocates saveareas used by the called routines, which must specify SAVAREA=PREALLOC to use the previously allocated saveareas. The saveareas are allocated

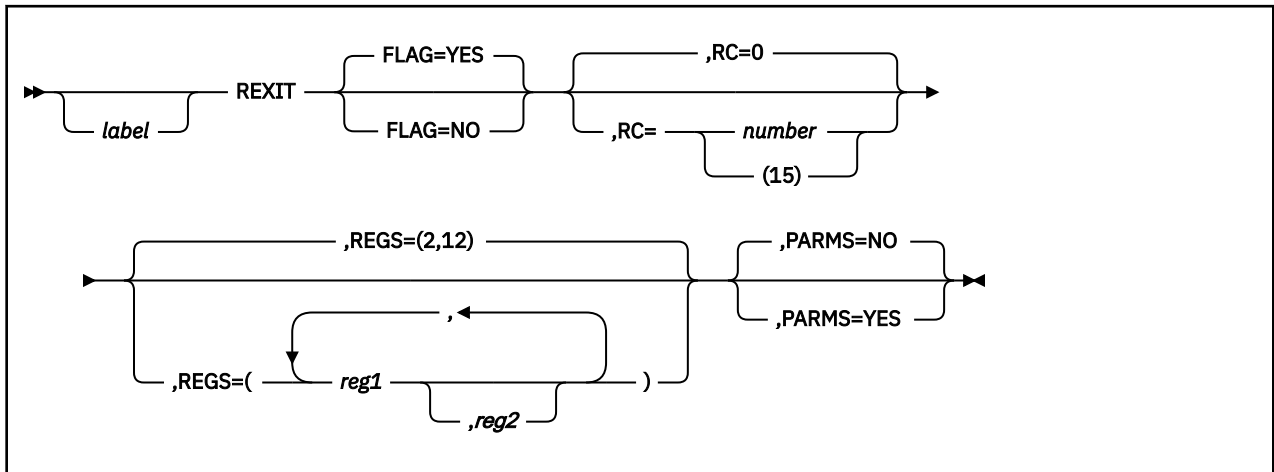
only by a module at the subtask level (for example, link driver or spool manager tasks). You must ensure that there are enough saveareas allocated so that they are available for calls to lower level modules.

7. Word 1 of the first savearea contains the length of the register saveareas and dynamic work area. In RSCS, only the RENTRY and REXIT macros use this reserved word in OS register saving conventions. This allows the REXIT macro to issue a FREEMAIN macro to release this area when the routine returns control to the caller.
8. Each generated savearea is 104 bytes (26 words) long. The first 72 bytes are the OS register savearea used by the RENTRY and REXIT macros. The 32-byte area after the register savearea is a general purpose work area that is available to the called routine. The address of the work area can be found by adding 72 to the contents of register 13 (this is the same as the SAVEWRK1 field of the SAVEAREA DSECT).
9. If you specify SAVE=NO, the caller's registers are not saved. This option is usually used in a VTAM exit routine (with RENT=NO) because the caller does not care if its registers are destroyed.  
  
Care must be taken if the called exit routine calls another routine. Because the RENTRY code has not saved register 14 (the return register), linkage back to the caller would be destroyed. Here, the exit routine must save and restore register 14.
10. Using internal entry points for subroutines, you can avoid exceeding addressability restrictions in a module; only one base register (12) is required. Each subroutine can be a maximum of 4096 bytes in length. Subroutines issue the RCALL macro to call each other. For this type of module, a register must be reserved (and initialized) to point to the static DC area, if all entry points are to be able to address it; another register must be reserved to point to any dynamic area. Register 13 can be used to point to the savearea and the dynamic area.
11. The machine-readable IBM copyright notification has the following format (where *number* is the PGMNUM value and *copyright* is the COYR value):  
  

```
number (C) COPYRIGHT IBM CORP copyright LICENSED MATERIAL - PROGRAM  
PROPERTY OF IBM
```
12. You can specify the WORK=YES parameter only once within the module. If more than one entry point calls a work area, additional coding is needed to obtain addressability to the work area.
13. If you specify SAVAREA=YES, RSCS links the acquired saveareas into a doubly linked list. Entry points that specify SAVAREA=PREALLOC can then detect when they find the last allocated savearea. Here, a protection exception occurs and R1 points to the last savearea in the list.
14. If you specify SAVE=YES and SAVAREA=NONE, RSCS acquires storage for the specified number of saveareas and places them in a doubly linked list. However, RSCS does not assume that the calling routine has provided a savearea. Here, a work register is required; unless specified on the WRKREG parameter, register 2 is the default. This special condition is used only when VTAM does not provide a savearea for VTAM exits.
15. When RSCS acquires storage for saveareas, it places the subpool of the storage in the first byte of the first savearea; the second and third bytes of this savearea contain the length of the storage.

The fourth byte of all saveareas specifies the number of available saveareas in the list. You can use this area for debugging purposes or to determine if RSCS has provided enough saveareas for an exit routine.

## REXIT – Defining a Module Return Point



### Purpose

The REXIT macro defines a return point from a module. REXIT generates all necessary instructions to restore the caller's registers, release a dynamically obtained register savearea, and return to the caller.

### Parameters

#### *label*

is any valid assembler statement label.

#### **FLAG=**

specifies whether the caller's savearea is flagged when returning control.

##### **YES**

Flags the savearea. This is the default.

##### **NO**

Does not flag the savearea.

#### **,RC=number**

specifies the return code passed in register 15 to the calling program. The return code *number* has a maximum value of 4095.

If you specify RC= (15) , it shows that the return code has been previously loaded into register 15. If this parameter is not specified, REXIT issues a return code of 0.

#### **,REGS=(reg1,reg2)**

specifies the registers that are restored when returning control. The registers are specified as one or more pairs of numbers between 0 and 12. To specify one register, omit the second value in a pair. If you do not specify a value, registers 2 - 12 are restored. Registers 0 and 1 must be restored if they are used as temporary or work registers and are not passed back to the caller by the PARMS parameter. See usage note [“3”](#) on page 296 for examples of specifying register pairs.

#### **,PARMS=**

specifies whether this entry point passes values back to the caller in registers 0 and 1.

##### **NO**

Does not pass back any values. This is the default.

##### **YES**

Passes parameters in R0 and R1.

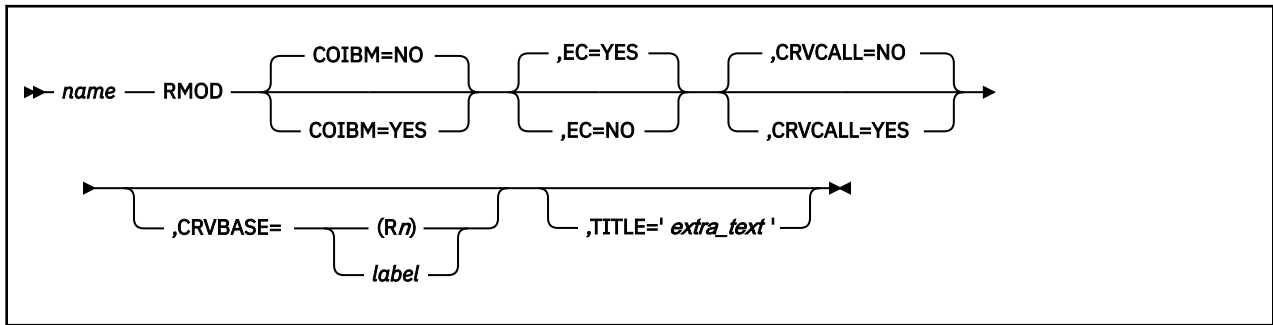
## Usage Notes

1. If you specify FLAG=YES, an NI (AND immediate) instruction of X'7F' is performed on the high-order byte of word 4 of the savearea after the registers are restored. If the caller has not provided a savearea address in register 13, the routine must specify FLAG=NO when returning control.
2. The parameters you specify and the assembler global variables set by the immediately preceding RENTRY macro determine the code generated by the REXIT macro.

If used in pairs, you can specify many RENTRY and REXIT macros in a module. However, if you use one REXIT macro, you must specify the same values for the RENT, SAVE, and SAVAREA parameters on all RENTRY macros. Otherwise, errors might occur on exit from the routine.

3. Examples of REGS parameter usage:
  - To restore registers 4 to 12 and return the values in registers 2 and 3 to the caller, specify REGS=(4,12).
  - To return the value in register 3 to the caller and restore register 2 and registers 4 - 12, specify REGS=(2, ,4,12).
4. Specify PARMS=YES only if RENT=YES and SAVAREA=YES were specified on the RENTRY macro for the module. This saves the values of registers 0 and 1 on return from the routine. For other cases, REXIT always passes the value of registers 0 and 1 to the calling routine.

## RMOD – Defining a Module



### Purpose

The RMOD macro generates the TITLE statement, CSECT statement, machine-readable eye catcher, and standard RSCS equate symbols (defined by RSSEQU COPY). It can also generate the machine-readable IBM copyright notification. You can specify RMOD only once in a module; it must appear before any RENTRY macros. (See the RENTRY macro for more information about generating internal and external entry point names.)

### Parameters

#### *name*

is any valid assembler statement that identifies the name of the module. This parameter must contain 6 characters.

#### **COIBM=**

specifies whether the machine-readable IBM copyright notification is to be generated.

##### **NO**

Does not generate the copyright. This is the default.

##### **YES**

Generates the copyright for the first module in the RSCS load module. COIBM=YES is intended for IBM use only.

#### **,EC=**

specifies whether the machine-readable eye catcher is generated.

##### **YES**

Generates the eye catcher. This is the default.

##### **NO**

Does not generate the eye catcher.

#### **,CRVCALL=**

specifies whether the CRV should be used to call a routine whose name begins with DMT. This is done if the first 3 characters of *name* are not DMT.

##### **NO**

The address of the routine is not taken from the CRV table. This is the default. Specify this option if the routine can be accessed from the RSCS LOADLIB by a standard V-constant linkage.

##### **YES**

The address of the routine is taken from the CRV table. Specify this option if you are accessing routines that are not in the RSCS LOADLIB.

#### **,CRVBASE=**

specifies how the CRV table is accessed if you specified CRVCALL=YES to find the address of a routine.

**(Rn)**

Reserves register *n* to point to the CRV table; a USING statement is automatically generated.

**label**

Specifies the location where the CRV address is stored. Each time RCALL is issued, RSCS finds the CRV table, using the address stored at the *label*, to find the address of the requested routine.

**,TITLE='extra\_text'**

specifies the extra text added to the TITLE statement. The *extra\_text* value must be enclosed in single quotation marks. The default text is '*program function*', where *program* and *function* are the current RSCS program number and function level.

## Usage Notes

1. Equate symbols are not printed in the assembler listing, unless you specify the EXP option of the VMFHLASM command when you assemble the module.
2. The machine-readable eye catcher contains the specified module name and the date and time it was assembled:

```
name mm/dd/yy hh.mm
```

3. The machine-readable IBM copyright notification has the following form:

```
program (C) Copyright IBM Corp. - copyright
```

4. You must specify EC=NO if the module *name* is also used as its entry point name. If you specify EC=YES, the eye catcher appears at the entry point defined by the module name, rather than executable code.
5. RMOD generates the following TITLE statement (where *xxx* are characters 4 - 6 of the module name and *extra\_text* is the value on the TITLE parameter):

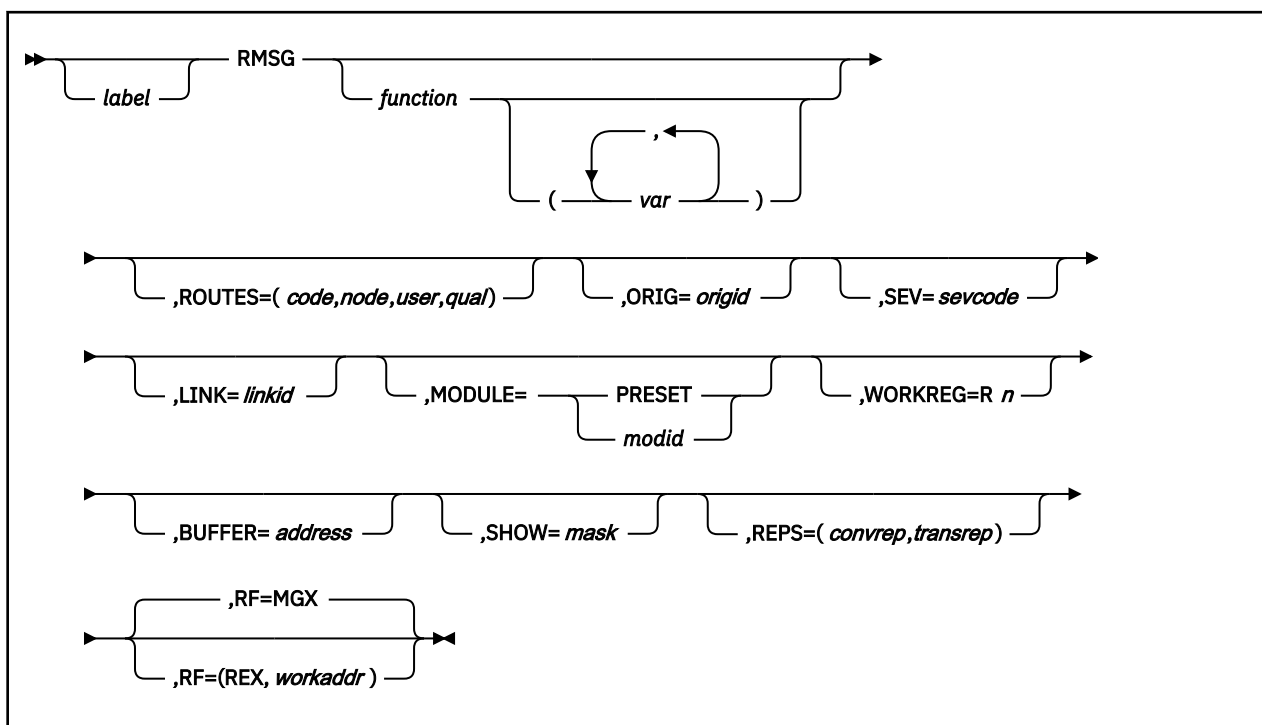
```
xxx name - VM RSCS NETWORKING extra_text
```

6. If you specify CRVCALL=YES, you should include the following statement at the end of the module:

```
CRV DSECT=YES
```



## RMSG – Issuing a Message



### Purpose

The RMSG macro updates MSGBLOK fields to describe a message. RSCS then begins the process of formatting and issuing the specified message.

### Parameters

#### *label*

is any valid assembler statement label.

#### *function*

specifies one of the following functions for this invocation of RMSG.

#### *msgnum*

Specifies the decimal number of the message to be issued.

#### **(Rn)**

Specifies a register that contains the number of the message to be issued.

#### **ALLOCATE**

Allocates a MSGBLOK, which will remain allocated after the message is issued. Its address is stored in the location specified on the BUFFER parameter; any values specified on other parameters are stored in the MSGBLOK, except for the REPS parameter. For more information, see the REPS parameter.

#### **FREE**

Deallocates a MSGBLOK that was previously allocated by the ALLOCATE parameter.

#### **SET**

Sets various information, specified by other RMSG parameters, in the MSGBLOK, but does not issue the message.

#### *var*

is a value that is placed in an MSGBVARS field in the MSGBLOK. The *var* values can be specified as RX-type addresses. Eight bytes of data from the address are moved to the appropriate MSGBVARS

field. They may also be specified as register numbers, where the number is stored in the first fullword in a MSGBVARS field.

**,ROUTES=(code,node,user,qual)**

specifies routing information for the message.

**code**

is the routing code of the message.

**node**

is the destination node.

**user**

is the destination user ID.

**qual**

is the qualifier for the destination node.

**,ORIG=origid**

identifies the origin of the command; the response message is then sent to that location. The *origid* points to an area, mapped by the CMORIG DSECT, that contains subfields that describe the destination node, user ID, and qualifier. Each subfield corresponds to the ROUTES parameter options. This area also contains three subfields that describe CRI specifications. The *origid* is found in data areas, such as the LINKTABL (LSTORIG) and CMNDAREA.

**,SEV=sevcode**

specifies the override severity code of the message.

**,LINK=linkid**

identifies the link that is associated with this message.

**,MODULE=**

identifies the module from which the message is issued.

**PRESET**

Specifies that the MSGBLOK already contains the module name.

**modid**

Identifies the routine that issues the message. The *modid* defaults to bytes 4 - 6 of the module *name* specified on the RMOD macro.

**,WORKREG=Rn**

specifies a work register when ALLOCATE, FREE, or SET is specified as the *function* parameter; R1 is the default work register.

**,BUFFER=address**

specifies a buffer that contains a MSGBLOK. The *address* can be specified as an RX-type address that refers to a fullword that contains the actual buffer address; you can also specify it as the number of a register that contains the buffer address.

**,SHOW=mask**

specifies the name of a *mask* that identifies the columns to be displayed in a columnar message. If this parameter is omitted, RMSG issues the message from the RSCS message repository.

**,REPS=(convrep,transrep)**

identifies repositories that contain information about the message.

**convrep**

is the address of a conversion repository.

**transrep**

is the address of a translation repository.

**Note**

The REPS parameter applies only to an individual RMSG macro invocation. After exiting the code generated by the RMSG macro, RSCS clears the fields. This lets you write an application that issues messages from the RSCS message repository and any other user-defined message repositories.

**,RF=**

specifies how the message should be issued.

**MGX**

Specifies that the message builder processes the MSGBLOK directly.

**(REX,*workaddr*)**

Specifies that the routine first passes the MSGBLOK and a work area address, *workaddr*, to the communications task, which then calls the message builder to issue the message.

## RWORK – Defining the Start of a Module Work Area

---

➤ RWORK ➤

### Purpose

The RWORK macro defines the start of a dynamic work area; RWORD starts a DSECT that maps all of the storage that follows. This storage is automatically obtained if WORK=YES is specified on the RENTRY macro (see [“RENTRY – Defining a Module Entry Point”](#) on page 290).

### Usage Notes

1. You can specify RWORK only once within a module.
2. DSECT or CSECT statements must not appear within the code delimited by the RWORK and RWORKEND macros.
3. You can specify only one set of RWORK and RWORKEND macros in a module.
4. The register savearea is generated immediately after the RWORK macro and appears at displacement zero within the dynamic work area DSECT. All defined areas appear at a displacement equal to the total length of all the register saveareas (104 multiplied by the value of the NUMSAVE parameter on the RENTRY macro).
5. The total length of the dynamic work area (including the register saveareas) cannot exceed 4095 bytes. If you attempt to define an area larger than 4095 bytes, an assembler error occurs from the expansion of the RENTRY macro.

## RWORKEND – Defining the End of a Module Work Area

---

➤ RWORKEND ➤

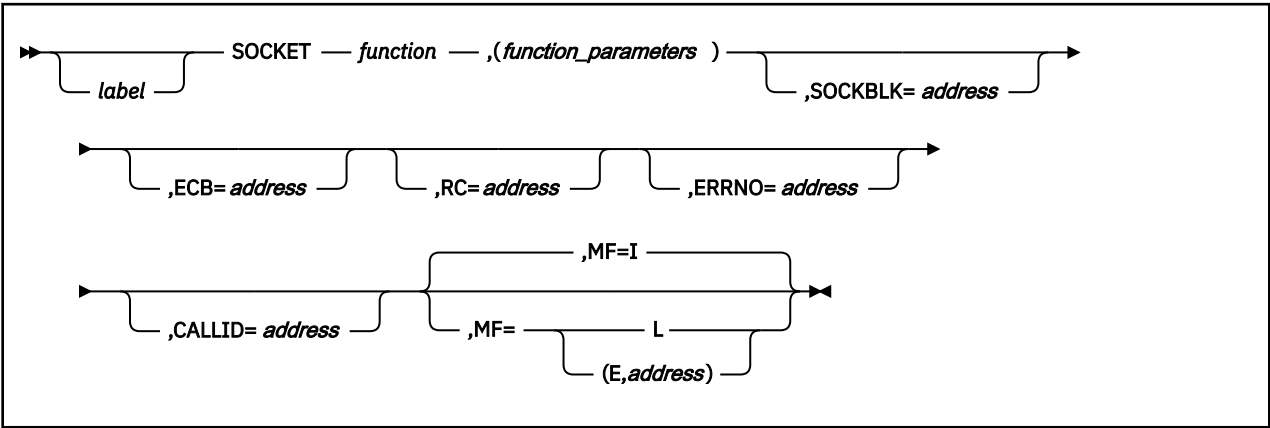
### Purpose

The RWORKEND macro defines the end of a dynamic work area that was started by the RWORK macro.

### Usage Notes

You can specify RWORKEND only once in a module.

# SOCKET – Using the TCP/IP Socket Interface



**Purpose**

Use the SOCKET macro to simplify the use of the RSCS socket interface to TCP/IP.

**Parameters**

**label**  
is any valid assembler statement label.

**function**  
is one of the following socket functions. (For descriptions of these functions, see “SOCKET Function Descriptions” on page 307.) This parameter is required for the standard form of this macro; it may be omitted on the list and execute formats.

ACCEPT	BIND	CANCEL	CLOSE
CONNECT	DSECT	FCNTL	GETCLIENTID
GETHOSTID	GETHOSTNAME	GETPEERNAME	GETSOCKNAME
GETSOCKOPT	GIVESOCKET	INITIALIZE	IOCTL
LISTEN	READ	RECV	RECVFROM
SELECT	SEND	SENDTO	SETSOCKOPT
SHUTDOWN	SOCKET	TAKESOCKET	TERMINATE
WRITE			

**Note:** The DSECT function does not generate a socket call; it generates the equates and DSECTs that are useful when using sockets.

**function\_parameters**  
specifies the parameter list for the socket function. Parameters are required for all function calls except DSECT, GETHOSTID, and TERMINATE. For descriptions of the parameters for each of the socket functions, see “SOCKET Function Descriptions” on page 307.

**,SOCKBLK=address**  
specifies a pointer to the socket block for all functions, except INITIALIZE. For the INITIALIZE function, this value is a pointer that will be filled in with the address of the socket block allocated by the call.

This parameter is required for the standard form of this macro; it may be omitted on the list and execute formats.

**,ECB=address**

specifies the ECB to be posted when the socket call completes.

This parameter is required for the standard form of this macro; it may be omitted on the list and execute formats.

**,RC=address**

specifies a fullword to receive the return code from the socket function. If you do not want to receive this return code, specify the *address* value as a null (0) pointer.

**Note:** This return code is not the return code issued from the SOCKET macro. Rather, it is the TCP/IP return code value from the actual socket call; see usage note “2” on page 305 for more information.

**,ERRNO=address**

specifies a fullword to receive the error number from the socket function; see usage note “2” on page 305. If you do not want the return code, specify the *address* value as a null (0) pointer.

**,CALLID=address**

specifies an 8-character string to receive the unique call ID from the socket function. If this value is not wanted, you can specify *address* as a null (0) pointer. See usage note “3” on page 306 for more information.

**,MF=**

specifies the format of the macro.

**I**

Generates an inline parameter list. This is the default.

**L**

Generates only a parameter list.

**(E,address)**

Specifies the execute form of the macro. The storage at the specified *address* is used as a parameter list. The address value is any valid RX-type address or a register, 1 - 12, that was previously loaded with the specified address. You can specify registers symbolically or with an absolute expression; they must be coded within parentheses. That is, register 1 must be specified as (1) or (R1).

For example, if (R2) is specified as the *address*, register 2 is pointing to the start of the parameter list. If MYLIST is specified, MYLIST is treated as the label on a parameter list. Only the parameters specified on the SOCKET macro will be altered in the parameter list; parameters need to be specified only if their value changes.

## Usage Notes

1. On return from the SOCKET macro, register 15 may contain the following return code values (this value is not from TCP/IP):

Return Code	Results
0	Socket function call has completed.
4	Socket function call has started; when the TCP/IP socket function completes, your ECB will be posted.
8	Temporary error occurred; retry the call later. Return code 8 occurs only if a recoverable error has been detected on the IUCV CONNECT request. For example, this may occur if the virtual machine that is the target of the connection request is not logged on or if it has not issued an IUCV declare buffer.

2. Each socket function call produces a TCP/IP return code. This return code value is returned in the RC=*address* field of the SOCKET macro. This value is not the return code produced by the SOCKET macro itself.

When the socket call completes successfully, the RC=*address* field will be set to 0, or it will contain the *rc* output value indicated in some function call descriptions. (The *rc* output value is not listed in the following sections for function calls that produce a 0 return code.)

If a TCP/IP error occurs and the function call is not successful, the RC=*address* field will be set to -1. If an IUCV error occurs (EIBMIUCV (1002)), the TCP/IP socket return code will have one of the following meanings, depending on the type of error that occurred:

- If the TCP/IP return code complement is less than 1256, the return code is the same as the one issued from the GCS IUCVINI and IUCVCOM macros.
  - If the TCP/IP return code complement is greater than 1256, the return code is the address of the IUCV interrupt IPARML
3. The CALLID=*address* field is filled in only after the IUCV SEND function has successfully started. The INITIALIZE socket call fills in this value only after it has sent the initial message to the TCP/IP virtual machine. The field will contain blanks if the IUCV SEND function has not yet occurred or if the IUCV path has quiesced and RSCS is waiting for an IUCV RESUME request. Do not attempt to cancel an outstanding socket call when the CALLID=*address* field contains blanks; wait until it contains a non-blank value.



## SOCKET Function Descriptions

This section shows the format of the individual socket calls that are supported as *function* parameters on the SOCKET macro. Each call and its input parameters are described. Output parameters (if any) are also described. These input parameters are the *function\_parameters* of the SOCKET macro. All parameters shown must be coded and separated by a comma; they must also be enclosed in parentheses. For more information about each socket function (except DSECT, GETHOSTBYNAME, INITIALIZE, and TERMINATE, which are RSCS-only IUCV socket calls), see the section on IUCV sockets in [z/VM: TCP/IP Programmer's Reference](#).

### Invoking the SOCKET Macro

In the following descriptions, the other parameters of the SOCKET macro are omitted. Typically, however, you would specify additional parameters. For example, [Figure 32 on page 307](#) shows how an INITIALIZE call can be coded on the SOCKET macro.

```
SOCKET INITIALIZE, (AXSLINK, MAXDESC, TCPID, MAXACALL),      X
                SOCKBLK=ASOCKBLK, ECB=UTLECB, RC=URETCODE, ERRNO=UERRNO,  X
                CALLID=UCALLID, MF=(E, UTLSP)
```

Figure 32. Sample of a SOCKET Macro Invocation

In this INITIALIZE call, AXSLINK is a label within the assemble file. The address of AXSLINK could also be loaded in a register. That register can then be specified on the macro in the form 0(Rx), where x is the register number.

### ACCEPT

The ACCEPT call is used by a server to accept a connection request from a client.

```
SOCKET ACCEPT, (socket, from, fromlen)
```

#### **socket**

is the address of a fullword integer socket descriptor.

#### **from**

is the address of a string to receive information about the connecting client (remote address and port). The format is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

#### **fromlen**

is the address of a fullword containing the length of the *from* parameter. The *fromlen* value must be 16.

#### **rc**

(Output) is a value returned when the socket call completes successfully, which contains the socket number that was allocated.

### BIND

The BIND call binds a unique local name to the socket that has the specified integer socket descriptor.

```
SOCKET BIND, (socket, name, namelen)
```

#### **socket**

is the address of a fullword integer socket descriptor.

#### **name**

is the address of the local address and port to which the socket is to be bound. The format of this structure is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

### ***namelen***

is the address of a fullword containing the length of the *name* parameter. The *namelen* value must be 16.

## CANCEL

The CANCEL call ends an outstanding socket call. Only the following socket calls may be canceled: read-type, write-type, ACCEPT, and SELECT.

SOCKET CANCEL,( <i>callid</i> )
---------------------------------

### ***callid***

is the 8-byte call ID string from the previous socket call.

## CLOSE

The CLOSE call is issued to shut down the socket associated with the specified socket descriptor and free the resources allocated for the socket.

SOCKET CLOSE,( <i>socket</i> )
--------------------------------

### ***socket***

is the address of a fullword integer socket descriptor.

## CONNECT

The CONNECT call attempts to establish a connection between two sockets.

SOCKET CONNECT,( <i>socket,name,namelen</i> )
---

### ***socket***

is the address of a fullword integer socket descriptor.

### ***name***

is the address of a string containing the remote address and port to which the socket is to be connected. The format of this structure is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

### ***namelen***

is the address of a fullword containing the length of the *name* parameter. The *namelen* value must be 16.

## DSECT

The DSECT call generates the equates and DSECTs that are used when making socket calls. This is an RSCS-only socket call.

SOCKET DSECT
--------------

## FCNTL

The FCNTL call controls the operating characteristics of a specified socket.

SOCKET FCNTL,( <i>socket,cmd,arg</i> )
--

### ***socket***

is the address of a fullword integer socket descriptor.

### ***cmd***

is the address of a fullword containing the command to perform:

**F\_SETFL**

Sets socket flags. FNDELAY, which is the only valid flag, sets the socket in nonblocking mode.

**F\_GETFL**

Queries the status of socket flags.

***arg***

is the address of a fullword containing the address of the data argument associated with the command. Only 0 and FNDELAY are supported.

***rc***

(Output) is a value returned when the socket call completes successfully for the F\_GETFL command, which contains the flags set as a bit mask.

**GETCLIENTID**

The GETCLIENTID call returns the identifier by which the calling program is known to the TCP/IP virtual machine.

SOCKET GETCLIENTID,( <i>domain,clientid</i> )
---

***domain***

is the address of a fullword containing the domain. Only AF\_INET (2), which defines addressing in the internet domain, is supported.

***clientid***

is the address of a string to receive the client ID. The format of this structure is defined by the CLIENTID DSECT generated by the SOCKET DSECT macro call.

**GETHOSTBYNAME**

The GETHOSTBYNAME call returns the unique 32-bit identifier for the host name being queried. This is an RSCS-only socket call. It results in a GetAddrInfo C call by the GETHOSTC user ID.

SOCKET GETHOSTBYNAME,( <i>name,namelen,dnsport</i> )
--

***name***

is the address of a string containing the host name.

***namelen***

is the address of a fullword containing the length of the *name* parameter. The *namelen* value must be 1 - 256.

***dnsport***

is the address of a halfword containing the port number of the RSCS domain name server on the local system.

**GETHOSTID**

The GETHOSTID call returns the unique 32-bit identifier for the current host.

SOCKET GETHOSTID
------------------

***rc***

(Output) is a value returned when the socket call completes successfully, which contains the 32-bit host ID.

**GETHOSTNAME**

The GETHOSTNAME call returns the name of the host processor on which the program is running.

SOCKET GETHOSTNAME,( <i>name,namelen</i> )
--

***name***

is the address of a string to receive the host name.

***namelen***

is the address of a fullword containing the length of the *name* parameter.

## GETPEERNAME

The GETPEERNAME call returns the name of the peer connected to the socket that is associated with the specified fullword socket descriptor.

SOCKET GETPEERNAME,( <i>socket,name,namelen</i> )
---

***socket***

is the address of a fullword integer socket descriptor.

***name***

is the address of a string to receive the remote address and port to which the socket is connected. The format of this structure is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

***namelen***

is the address of a fullword containing the length of the *name* parameter. The *namelen* value must be 16.

## GETSOCKNAME

The GETSOCKNAME call stores the current name for the socket that is associated with the specified fullword socket descriptor into the structure pointed to by the *name* parameter.

SOCKET GETSOCKNAME,( <i>socket,name,namelen</i> )
---

***socket***

is the address of a fullword integer socket descriptor.

***name***

is the address of a string to receive the local address and port to which the socket is bound. The format of this structure is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

***namelen***

is the address of a fullword containing the length of the *name* parameter. The *namelen* value must be 16.

## GETSOCKOPT

the GETSOCKOPT call returns the options associated with a specified socket.

SOCKET GETSOCKOPT,( <i>socket,level,optname,optval,optlen</i> )
---

***socket***

is the address of a fullword integer socket descriptor.

***level***

is the address of a fullword containing the option level. SOL\_SOCKET (X'FFFF') is the only supported level. It refers to the socket protocol level, as opposed to the TCP or IP level.

***optname***

is the address of a fullword containing an option name.

***optval***

is the address of a string to receive the option value.

***optlen***

is the address of a fullword containing the length of the *optval* parameter.

**GIVESOCKET**

The GIVESOCKET call tells TCP/IP to make the specified socket available to a TAKESOCKET call that is issued by another application running on the same host.

SOCKET GIVESOCKET,( <i>socket,clientid</i> )
--

***socket***

is the address of a fullword integer socket descriptor.

***clientid***

is the address of a string containing the client ID. The length is assumed to be 40 bytes. The format of this structure is defined by the CLIENTID DSECT generated by the SOCKET DSECT macro call.

**INITIALIZE**

The INITIALIZE call sets up the IUCV connection to the TCP/IP virtual machine and allocates a SOCKBLOK that must be passed on all other calls. It also sends the initial parameters to the TCP/IP virtual machine and allocates a socket set. This is an RSCS-only socket call.

SOCKET INITIALIZE,( <i>taskid,maxdesc,tcpid,maxcall</i> )
---

***taskid***

is the address of an 8-character string that contains the task ID, which is usually the link ID or the name of the RSCS task. This value and the user ID of the RSCS machine uniquely identify this socket set.

***maxdesc***

is the address of a fullword integer (0 - 2000) that indicates the maximum number of sockets that will be used in this socket set. If 0 is specified, the default value 50 is assumed.

***tcpid***

is the address of an 8-character string that contains the user ID of the TCP/IP virtual machine. If specified as a null (0) parameter, the user ID is assumed to be TCPIP.

***maxcall***

is the address of a fullword integer that indicates the maximum number of simultaneous socket calls that can be requested. If specified as a null (0) parameter, the default of 2 calls is used. On return, this will be set to the maximum number that TCP/IP allows.

**IOCTL**

The IOCTL call controls the operating characteristics of the specified sockets.

SOCKET IOCTL,( <i>socket,request,data</i> )
---

***socket***

is the address of a fullword integer socket descriptor.

***request***

is the address of a fullword containing the name of a request. These are listed in equates generated by the SOCKET DSECT macro call.

***data***

is the address of a string containing any *request* data.

## LISTEN

The LISTEN call completes the binding necessary for the socket associated with the specified fullword socket descriptor, if BIND has not been called. LISTEN also creates a queue of incoming connect requests.

SOCKET LISTEN,( <i>socket</i> , <i>backlog</i> )
--

### *socket*

is the address of a fullword integer socket descriptor.

### *backlog*

is the address of a fullword integer (0 - 2000) that specifies the length for the pending queue of connect requests.

## READ

The READ call reads data on the socket associated with the specified fullword socket descriptor and stores it in a buffer. This call is applicable only for connected sockets.

SOCKET READ,( <i>socket</i> , <i>buffer</i> , <i>buflen</i> )
---

### *socket*

is the address of a fullword integer socket descriptor.

### *buffer*

is the address of the storage area to which the data will be received.

### *buflen*

is the address of a fullword containing the length of the *buffer* parameter.

### *rc*

(Output) is a value returned when the socket call completes successfully, which contains the number of bytes read.

## RECV

The RECV call receives data on the socket that is associated with the specified fullword socket descriptor and stores it in a buffer. This call is applicable only for connected sockets.

SOCKET RECV,( <i>socket</i> , <i>buffer</i> , <i>buflen</i> , <i>flags</i> )
--

### *socket*

is the address of a fullword integer socket descriptor.

### *buffer*

is the address of the storage area to which the data will be received.

### *buflen*

is the address of a fullword containing the length of the *buffer* parameter.

### *flags*

is the address of a fullword containing one of the following supported receive flags:

#### MSG\_OOB (1)

Reads any out-of-band data on the socket.

#### MSG\_PEEK (2)

Peeks at the data present at the socket. The data is returned but is not consumed; the same data will be peeked by a subsequent receive operation.

### *rc*

(Output) is a value returned when the socket call completes successfully, which contains the number of bytes read.

## RECVFROM

The RECVFROM call receives data on the socket associated with the specified fullword socket descriptor and stores it in a buffer. This call applies to any datagram socket that is connected or unconnected.

SOCKET RECVFROM, (*socket*, *buffer*, *buflen*, *flags*, *from*, *fromlen*)

***socket***

is the address of a fullword integer socket descriptor.

***buffer***

is the address of the storage area to which the data will be received.

***buflen***

is the address of a fullword containing the length of the *buffer* parameter.

***flags***

is the address of a fullword containing one of the following supported receive flags:

**MSG\_OOB (1)**

Reads any out-of-band data on the socket.

**MSG\_PEEK (2)**

Peeks at the data present at the socket. The data is returned but is not consumed; the same data will be peeked by a subsequent receive operation.

***from***

is the address of a string to contain the remote address and port that sent the data. Specify this value with a null (0) pointer if you do not want this data. The format of this structure is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

***fromlen***

is the address of a fullword containing the length of the *from* parameter. If a null value is specified for *from*, this value may also contain a null value. If a *from* parameter is specified, the *fromlen* value must be 16.

***rc***

(Output) is a value returned when the socket call completes successfully, which contains the number of bytes read.

## SELECT

The SELECT call monitors activity on a set of sockets to determine if any sockets are ready for reading, writing, or have an exceptional condition pending.

SOCKET SELECT, (*nsds*, *rmask*, *wmask*, *emask*, *timeout*)

***nsds***

is the address of a fullword integer that defines the maximum number of socket descriptors to be checked. This value should not be larger than 1 plus the largest descriptor number actually in use.

***rmask***

is the address of a bit mask string that contains the read descriptors to be checked for data. If this value is null, no descriptors will be checked.

The offset of the word containing the bit for a socket descriptor is calculated as follows:

```
offset=(descriptor_number/32)*4
```

The bit for a descriptor is masked as follows (<< is the left shift operator):

```
bitmask=X'00000001'<<(descriptor_number modulo 32)
```

### ***wmask***

is the address of a fullword bit mask string containing the write descriptors that will be checked to determine if TCP/IP has buffer space available to write data to the socket. If specified as a null pointer, no descriptors are to be checked.

### ***emask***

is the address of a bit mask string containing the exception descriptors that will be checked for exceptional pending conditions. If specified as a null pointer, no descriptors are to be checked.

### ***timeout***

is the address of an 8-byte field that contains a value that determines how long the task will wait for the SELECT call to complete. The format of this structure is defined by the TIMEVAL DSECT that is generated by the SOCKET DSECT macro call. The first 4-byte field contains the seconds value; the second 4-byte field contains the microseconds value. This value determines how the SELECT call completes:

#### **Null**

Completes only when one of the mask values is satisfied.

#### **0**

Completes immediately.

#### **Non-zero**

Completes when the *timeout* period expires or when one of the masks is satisfied.

### ***rc***

(Output) is a value returned when the socket call completes successfully, which contains the total number of ready sockets in all mask sets.

## SEND

The SEND call sends packets on the socket that is associated with the specified fullword integer socket descriptor. This call applies only to connected sockets.

SOCKET SEND,( <i>socket</i> , <i>buffer</i> , <i>buflen</i> , <i>flags</i> )
--

### ***socket***

is the address of a fullword integer socket descriptor.

### ***buffer***

is the address of the storage area to which the data will be received.

### ***buflen***

is the address of a fullword containing the length of the *buffer* parameter.

### ***flags***

is the address of a fullword containing one of the following supported send flags:

#### **MSG\_OOB (1)**

Sends any out-of-band data on sockets for which it is supported.

### ***rc***

(Output) is a value returned when the socket call completes successfully, which contains the number of bytes sent. This does not indicate, however, that the data was actually received or delivered to the other side of the socket. An *rc* value of -1 indicates a locally detected error.

## SENDTO

The SENDTO call sends packets on the socket that is associated with the specified fullword integer socket descriptor. This call applies to any datagram socket, whether connected or unconnected.

SOCKET SENDTO,( <i>socket</i> , <i>buffer</i> , <i>buflen</i> , <i>flags</i> , <i>to</i> , <i>tolen</i> )
---

### ***socket***

is the address of a fullword integer socket descriptor.



***buffer***

is the address of the storage area to which the data will be received.

***buflen***

is the address of a fullword containing the length of the *buffer* parameter.

***flags***

is the address of a fullword containing one of the following supported send flags:

**MSG\_OOB (1)**

Sends any out-of-band data on sockets for which it is supported.

***to***

is the address of a string to receive the remote address and port that will receive the data. The format of this structure is defined by the SOCKADDR DSECT generated by the SOCKET DSECT macro call.

***tolen***

is the address of a fullword containing the length of the *to* parameter. The *tolen* value must be 16.

***rc***

(Output) is a value returned when the socket call completes successfully, which contains the number of bytes sent. This does not indicate, however, that the data was actually received or delivered to the other side of the socket. An *rc* value of -1 indicates a locally detected error.

## SETSOCKOPT

The SETSOCKOPT call sets options associated with the specified socket.

SOCKET SETSOCKOPT,( <i>socket,level,optname,optval,optlen</i> )
---

***socket***

is the address of a fullword integer socket descriptor.

***level***

is the address of a fullword integer containing the option level. SOL\_SOCKET (X'FFFF') is the only supported level. It refers to the socket protocol level, as opposed to the TCP or IP levels.

***optname***

is the address of a fullword containing an option name.

***optval***

is the address of a string containing the option value. This is a fullword integer except for the SO\_LINGER option, which is a doubleword.

***optlen***

is the address of a fullword containing the length of the *optval* parameter.

## SHUTDOWN

The SHUTDOWN call shuts down all or part of a duplex connection.

SOCKET SHUTDOWN,( <i>socket,how</i> )
---------------------------------------

***socket***

is the address of a fullword integer socket descriptor.

***how***

is the address of a fullword integer that describes how the socket is to be shutdown:

**0**

End communication from the socket.

**1**

End communication to the socket.

**2**

End communication to and from the socket.

## SOCKET

The SOCKET call creates an endpoint for communication and returns a socket descriptor that represents the endpoint.

SOCKET SOCKET, (*domain, type, protocol*)

***domain***

is the address of a fullword integer containing the domain. Only AF\_INET (2) is supported, which defines addressing in the internet domain.

***type***

is the address of a fullword containing one of the following supported socket types:

**SOCK\_STREAM (1)**

Provides the sequence of 2-way byte streams that are reliable and connection-oriented.

**SOCK\_DGRAM (2)**

Provides the datagrams that are connectionless messages with a fixed maximum length whose reliability is not guaranteed.

**SOCK\_RAW (3)**

Provides an interface to internal protocols, such as IP and ICMP.

***protocol***

is the address of a fullword containing the protocol. Only PF\_INET (2) is supported. If set to 0, the default protocol number for the requested domain and socket type is used.

***rc***

(Output) is a value returned when the socket call completes successfully, which contains the socket number allocated.

## TAKESOCKET

The TAKESOCKET call acquires a socket from another program.

SOCKET TAKESOCKET, (*clientid, socket*)

***clientid***

is the address of a string containing the client ID. The length is assumed to be 40 bytes. The format of this structure is defined by the CLIENTID DSECT that is generated by the SOCKET DSECT macro call.

***socket***

is the address of a fullword integer socket descriptor.

***rc***

(Output) is a value returned when the socket call completes successfully, which contains the socket number allocated.

## TERMINATE

The TERMINATE call ends IUCV communication with the TCP/IP virtual machine and deallocates the SOCKBLOK and all SOCKCBLKs. Because storage that is allocated by INITIALIZE is deallocated only by TERMINATE, this function should be called even if the INITIALIZE call ended in error. This is an RSCS-only socket call.

SOCKET TERMINATE

## WRITE

The WRITE call writes data on the socket associated with the specified socket descriptor. This call applies only to connected sockets.

SOCKET WRITE, ( <i>socket</i> , <i>buffer</i> , <i>buflen</i> )
---

***socket***

is the address of a fullword integer socket descriptor.

***buffer***

is the address of the storage area that contains the data to write.

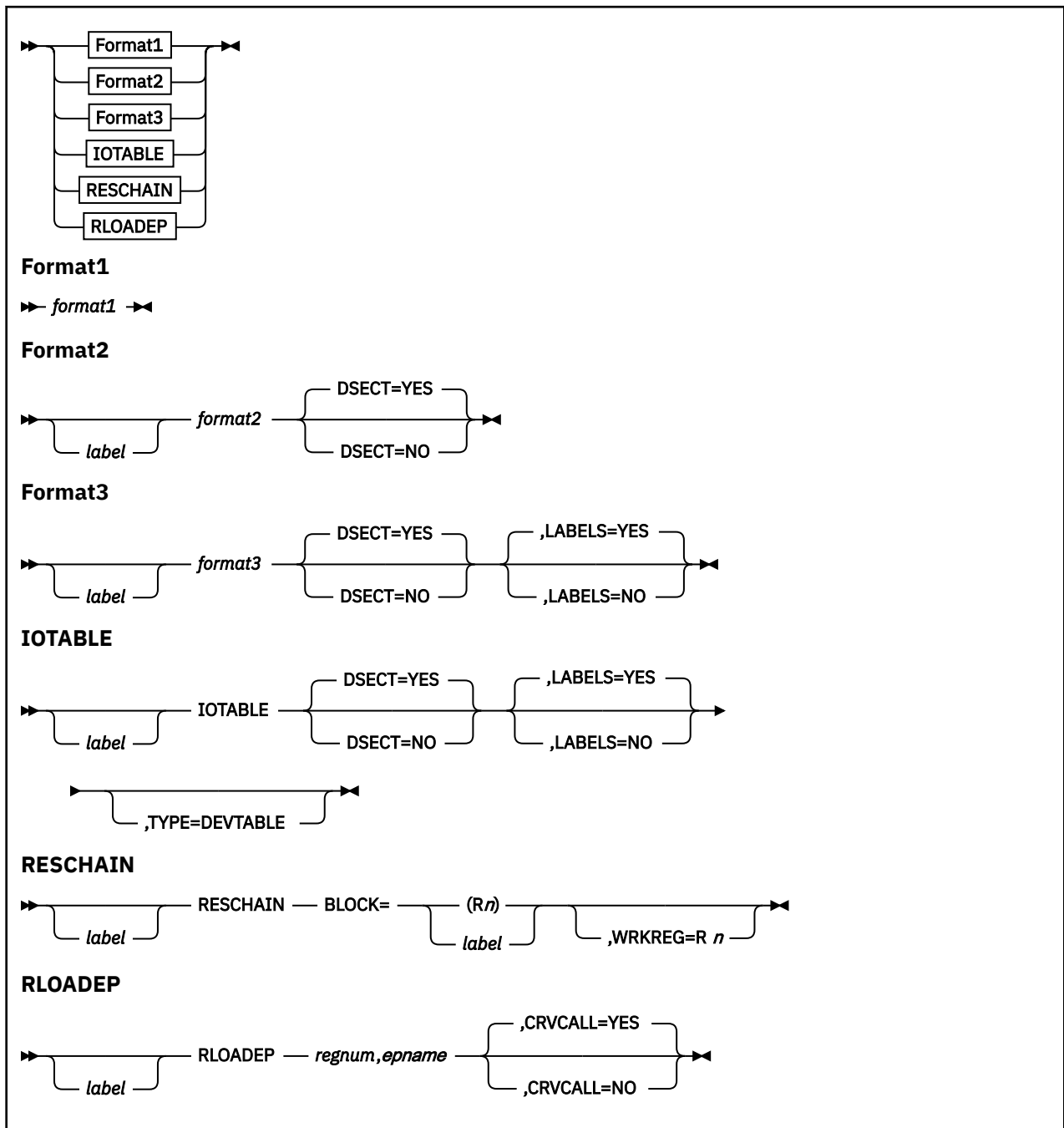
***buflen***

is the address of a fullword containing the length of the *buffer* parameter.

***rc***

(Output) is a value returned when the socket call completes successfully, which contains the number of bytes written.

## Control Block Macros



### Purpose

The RSCS control block macros define the format of RSCS control blocks or mapping DSECTs. Most RSCS control block macros conform to one of the formats shown in this syntax diagram. Some control block macro functions are local to some modules and are not included in this syntax diagram.

## Parameters

### *label*

is any valid assembler statement label. If the label is omitted and LABELS=YES is specified (or allowed to default), RSCS generates a label with the name of the macro, except for the following macros:

Macro	Generated Label
RDR	RDRPARMS
IOTABLE	DEVTABLE (if TYPE=DEVTABLE is specified)

### *format1*

specifies one of the following macro names. RSCS generates a label that matches the name of the macro.

NHDTR	NJEEQU	NMR	RIB	TIB
-------	--------	-----	-----	-----

### *format2*

specifies one of the following macro names:

ACNTBUFF	AUTHBLOK	CMNDAREA	CRV	CVT
ECXBLOK	FILREQ	ITRACREC	LINKTABL	MSGLINE
MSGWA	PORT	PRDBLOK	ROUTEGRP	SAFTAG
SAVEAREA	SOCKBLOK	SOCKCBLK	SYSIDENT	TAG
TASHADOW	TASKBLOK	XABHDR		

### *format3*

specifies one of the following macro names:

MSGBLOK	NOTEBLOK	RDEVBLOK	RDR	REROUTE
RESBLOK	SEPBLOK			

### **IOTABLE**

specifies the IOTABLE macro, which maps the IOTABLE control block that RSCS uses when performing I/O operations to a device.

### **DSECT=**

specifies whether a DSECT statement is generated.

#### **YES**

Generates a DSECT statement. This is the default.

#### **NO**

Does not generate a DSECT statement. The macro expansion is generated as a continuation of the current CSECT or DSECT.

### **,LABELS=**

specifies whether assembler labels are generated for each statement in the macro expansion. The generated labels are defined for the appropriate control blocks.

#### **YES**

Generates labels. This is the default. You can specify YES only once for each control block macro in the module.

#### **NO**

Does not generate labels.

### **,TYPE=DEVTABLE**

generates a device block (DEVxxxxx) for use with a device other than an input or output unit record device.

### **RESCHAIN**

specifies the RESCHAIN macro, which adds a RESBLOK to the RSCS global resource chain.

### **BLOCK=**

identifies the RESBLOK to be enqueued. It can be identified by a register (*Rn*) that contains its address, or by an associated *label*.

### **,WRKREG=Rn**

specifies an internal work register, *n*.

### **RLOADEP *regnum,epname***

specifies the RLOADEP macro, which loads a register with an entry point address. The entry point is identified by register *regnum*, which contains the entry point address, and by its name (*epname*).

### **,CRVCALL=**

specifies whether the CRV should be used to find the address of the entry point whose *epname* begins with DMT.

#### **YES**

The entry point address is taken from the CRV. This is the default.

#### **NO**

The address of the entry point is not taken from the CRV. The entry point must be externalized and accessible by standard V-constant linkage.

## **Usage Notes**

1. If you specify DSECT=YES, the specified *label* (or the default label) is generated as the label of the DSECT statement. If you specify DSECT=NO, the label is generated as a DS OD statement.
2. The NHDTR macro maps the NJE job headers, data set headers, and job trailers. It generates the following DSECT labels: NJH, NDH, and NJT.

## Chapter 12. Supported Routines in the CRV

The RSCS common routines vector table (CRV) contains a list of pointers to RSCS routines that are supported for use by exit routines. The following sections identify the executable entry points and the nonexecutable entry points.

### Executable Entry Points

This section describes the executable entry points listed in the CRV that are supported for customer use. The routines are listed here in alphabetical order, and the following information is provided for each routine:

- Intended function
- Significant input and output parameters
- Return codes and their meanings

#### DMTAXMRQ

The DMTAXMRQ routine processes requests to open or close an input or output spool file.

Parameter	Reg	Function
Input Parameter	R1	Address of an RDEVBLK
Output Parameter	R15	Return code:
		<b>0</b> Requested processing completed.
		<b>4</b> File not found.
		<b>8</b> File already opened.
		<b>12</b> Unavailable resources (storage, TAG elements, or devices).
		<b>16</b> CP spool error.

#### DMTBPLLX

The DMTBPLLX routine loads exit modules into storage for RSCS.

Parameter	Reg	Function
Input Parameter	R1	Address of 8-character module name to be loaded
Output Parameter	R15	Return code:
		<b>0</b> Processing completed; R0 contains the address of the loaded module.
		<b>4</b> Module not loaded.

## DMTCOMDG

The DMTCOMDG routine validates EBCDIC decimal values that are sent by the calling routine and converts them into binary values.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:
		<b>Word 1</b> Address of the decimal value to validate
		<b>Word 2</b> Length of the value (up to 10 bytes)
		<b>Word 3</b> Address of the valid range (low value in first fullword, high value in second fullword)
Output Parameters	R0	Converted value
	R15	Return code 0

## DMTCOMDQ

The DMTCOMDQ routine dequeues command and message elements from a queue, in FIFO order, and places them in a buffer supplied by the calling routine.

Parameter	Reg	Function
Input Parameter	R1	Address of parameter list, containing:
		<b>Word 1</b> Address of queue anchor (4 bytes)
		<b>Word 2</b> Address of buffer (1 - 256 bytes), or 0 if the element is to be purged
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed.
		<b>4</b> End of queue.

## DMTCOMFI

The DMTCOMFI routine accesses records from files on any disk that GCS has accessed in the RSCS virtual machine.

Parameter	Reg	Function
Input Parameter	R1	Address of FILREQ request block



Parameter	Reg	Function
Output Parameters	R0	Length of retrieved record
	R1	Pointer to retrieved record
	R15	Return code:
	<b>0</b>	Normal processing completed.
	<b>4</b>	End of file.
	<b>8</b>	No storage available.
	<b>12</b>	Bad return code from FILEDEF or OPEN failed.
	<b>16</b>	File format not valid.
	<b>20</b>	IMBED record not valid.
	<b>24</b>	Maximum number of open files (1000) exceeded.
	<b>28</b>	Maximum IMBED depth (10) reached.

## DMTCOMGG

The DMTCOMGG routine scans the ROUTEGRP table to find a specified routing group name.

Parameter	Reg	Function
Input Parameter	R1	Pointer to 8-byte group field
Output Parameters	R1	Pointer to the ROUTEGRP entry
	R15	Return code:
	<b>0</b>	Normal processing completed.
	<b>4</b>	Entry not found.

## DMTCOMGN

The DMTCOMGN routine searches the ROUTEGRP table to find the route information for a node.

Parameter	Reg	Function
Input Parameter	R1	Pointer to 8-byte node name
Output Parameters	R0	Pointer to NODE entry
	R1	Pointer to ROUTEGRP
	R15	Return code:
	<b>0</b>	Normal processing completed.
	<b>4</b>	NODE entry not found.

## DMTCOMHG

The DMTCOMHG routine validates EBCDIC hexadecimal values sent by the calling program and converts them into binary values.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:
		<b>Word 1</b> Address of hexadecimal value to validate
		<b>Word 2</b> Length of the value (up to 10 bytes)
		<b>Word 3</b> Address of the valid range (low value in first fullword, high value in second fullword)
Output Parameters	R0	Contains converted value
	R15	Return code:
		<b>0</b> Normal processing completed.
		<b>4</b> Value is out of range.
		<b>8</b> Characters in parameter are not valid.

## DMTCOMLK

The DMTCOMLK routine locates entries in the link table (LINKTABL).

Parameter	Reg	Function
Input Parameter	R1	Pointer to 8-byte link ID field
Output Parameters	R1	Pointer to the LINKTABL entry
	R15	Return code:
		<b>0</b> Normal processing completed.
		<b>4</b> Link not found.

## DMTCOMNQ

The DMTCOMNQ routine places command and message elements, which are supplied by the calling routine, in FIFO order in a task's queue.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:
		<b>Word 1</b> Address of queue anchor (4 bytes)
		<b>Word 2</b> Address of element (1 - 256 bytes); byte 0 must contain the length of the element - 1

Parameter	Reg	Function
Output Parameter	R15	Return code:  <b>0</b> Normal processing completed.  <b>4</b> Insufficient storage; element not queued.

## DMTCOMSM

The DMTCOMSM routine sends L3 type command or message elements on a link.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:  <b>Word 1</b> Pointer to LINKTABL entry  <b>Word 2</b> Pointer to Type L3 element
Output Parameter	R15	Return code:  <b>0</b> Normal processing completed.  <b>4</b> Message cannot be sent on the link.

## DMTCOMTE

The DMTCOMTE routine converts TOD clock values into EBCDIC time and date format.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:  <b>Word 1</b> Address of input time in TOD format (8 bytes)  <b>Word 2</b> Address of time zone number (0, non-0, 1 byte)  <b>Word 3</b> Address of area in which to place the converted value  <b>Word 4</b> Address of a work area that is aligned on a doubleword
Output Parameter	R15	Return code 0; the date, time, and time zone are placed in the area specified by Word 3.

The work area pointed to by Word 3 should have the following format;

DC	AL1()	Length of edit area to follow (not including 6-byte time zone specification). On return it will be overwritten.
DC	XL	Edit area to place converted value into.
DC	XL6	Time zone specification.

The work area pointed to by Word 4 should have the following format. It should not be modified after the first call to DMTCOMTE.

DC	3D'0'	For date and time decimal conversion
DC	A(0)	Reserved
DC	F'-1'	To hold last calculation elapsed hours
DC	A(0)	Reserved
DC	A(0)	Reserved

## DMTCOMTS

The DMTCOMTS routine converts EBCDIC time and date values into TOD clock values.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Address of EBCDIC time value ( <i>mm/dd/yyhh:mm:ss</i> ) <b>Word 2</b> Address of time zone number (0-24, 1 byte) <b>Word 3</b> Address of a field to hold the converted time value (8 bytes) <b>Word 4</b> Address of a doubleword aligned work area
Output Parameter	R15	Return code 0; the TOD clock value is placed in the area specified by Word 3.

The work area pointed to by Word 4 has the following format:

DC	D'0'	Utility work area
DC	F'0'	Save area for decimal date
DC	F'0'	Save area for decimal time

## DMTDDLEP

The DMTDDLEP routine punches data in NETDATA NOTE format to a unit record device.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing: <b>Word 1</b> Address of a work area (contains 0 on first call) <b>Word 2</b> Address of a record buffer in the following format (contains 0 on last call): <div>AL1(nn), CLnn'data'</div> <b>Word 3</b> Pointer to IOTABLE <b>Word 4</b> Pointer to LINKTABL entry <b>Word 5</b> Logical record length (LRECL) override value (default is 80)

Parameter	Reg	Function
Output Parameters	R0	Work area address (contains 0 for last call)
	R15	Return code:
	0	Normal processing completed.
	4	Problem with storage.
	8	Error in unit record device output.

## DMTHASHA

The DMTHASHA routine adds an element to a hash table. It does not allocate storage or check for duplicate keys to the hash table.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:
		<b>Word 1</b> Address of HASHBLOK
		<b>Word 2</b> Address of the element
Output Parameter	R15	Return code 0

## DMTHASHB

The DMTHASHB routine builds a hash table for various types of data areas.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:
		<b>Word 1</b> Address of HASHBLOK
		<b>Word 2</b> Address of a control block chain anchor
Output Parameter	R15	Return code:
		0
		Normal processing completed.
		4
		Insufficient storage for hash index table.

## DMTHASHC

The DMTHASHC routine clears a hash table and its associated storage when RSCS ends.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing:
		<b>Word 1</b> Address of HASHBLOK
Output Parameter	R15	Return code 0

## DMTHASHD

The DMTHASHD routine deletes a data area from a hash table. The calling routine, however, must first locate the data area before it can be deleted.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Address of HASHBLOK <b>Word 2</b> Address of element to be deleted
Output Parameter	R15	Return code 0

## DMTHASHF

The DMTHASHF routine finds a data area in the hash table. It returns the address of the data area to the calling routine.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Address of HASHBLOK <b>Word 2</b> Address of the requested key
Output Parameter	R15	Return code: <b>0</b> Element was found; R1 contains its address. <b>4</b> Element was not found; R1 contents are destroyed.

## DMTHASHG

The DMTHASHG routine finds generic entries in hash tables. If the hash table does not support generics (see [“HASHBLOK – Defining a Hash Table”](#) on page 270), DMTHASHG willabend.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Address of HASHBLOK <b>Word 2</b> Address of requested key
Output Parameter	R15	Return code: <b>0</b> Element was found; R1 contains its address. <b>4</b> Element was not found; R1 contents are destroyed.

## DMTHASHS

The DMTHASHS routine determines the number of hash chain anchors in use and the length of the longest chain.

Parameter	Reg	Function
Input Parameter	R1	Address of HASHBLOK
Output Parameter	R15	Return code 0

## DMTIOTHD

The DMTIOTHD routine translates general I/O halt processing requests into GCS I/O requests.

Parameter	Reg	Function
Input Parameter	R1	Address of IOTABLE
Output Parameter	R15	Return code 0; IODSIOCC field is set accordingly: <b>X'00'</b> Normal processing completed. <b>X'03'</b> GENIO error occurred.

## DMTIOTST

The DMTIOTST routine translates general I/O start requests into GCS I/O requests. If the number of consecutive I/O requests is exceeded or a programming error occurs, abend X'010' will occur.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing: <b>Word 1</b> Address of IOTABLE <b>Word 2</b> Address of LINKTABL entry
Output Parameter	R15	Return code 0; the IODSIOCC field is also set accordingly: <b>X'00'</b> Normal processing completed. <b>X'01'</b> CSW stored. <b>X'03'</b> GENIO error occurred.

## DMTLOGCL

The DMTLOGCL routine flushes output to a log spool device and closes the device.

Parameter	Reg	Function
Input Parameter	R1	Address of LOGPARMS control block

Parameter	Reg	Function
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed.
		<b>4</b> An error specified by DMTAXMRQ.
		<b>n</b> The return code passed from DMTUROFL.

## DMTLOGEP

The DMTLOGEP routine opens an output device and places records, passed by the calling routine, in that device.

Parameter	Reg	Function
Input Parameter	R1	Address of LOGPARMS area
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed.
		<b>4</b> Output device could not be opened.
		<b>8</b> Irrecoverable I/O error on output device.

## DMTMANDE

DMTMANDE is the common ESTAE processing routine for RSCS. The requirements for the input parameters vary, depending on the availability of SDWA storage.

Parameter	Reg	Function
Input Parameter (Storage available)	R0	I/O processing return code (X'10')
	R1	Address of the ESTAE diagnostic area (SDWA)
Input Parameter (Storage unavailable)	R0	Indicator X'0C' (no SDWA available)
	R1	Abend completion code
	R2	Address of the LINKTABL entry for failed link driver task
Output Parameter	R15	Return code 0

## DMTMGFFM

The DMTMGFFM routine formats a message described by the specified MSGBLOK.

Parameter	Reg	Function
Input Parameters	R8	Pointer to the MSGBLOK that describes the requested message
	R9	Pointer to a message work area (MSGWA)



Parameter	Reg	Function
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed; the message is formatted.
		<b>4</b> Message is not formatted; insufficient storage or message was not found in the specified repository.

## DMTMGXEP

The DMTMGXEP routine issues a message in a specified language.

Parameter	Reg	Function
Input Parameter	R1	Pointer to the MSGBLOK that describes the requested message
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed; the message is issued
		<b>12</b> Message is undefined.

## DMTMPTBP

The DMTMPTBP routine 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, DMTMPTBP returns control to the calling routine.

Parameter	Reg	Function
Input Parameters	R0	Points to keyword table
	R1	Address of a parameter list, containing:
		<b>Word 1</b> Pointer to 16-byte parameter area
		<b>Word 2</b> Pointer to start of previous token
		<b>Word 3</b> Length of previous token
		<b>Word 4</b> Pointer to end of the text to be parsed
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed. If a matching keyword is not found, control returns to the calling routine. Otherwise, control passes to the processing routine associated with the keyword.
		<b>4</b> Parameter missing.
		<b>8</b> Parameter too long.

## DMTMPTCK

The DMTMPTCK routine compares a parsed parameter to the keyword table supplied by the calling routine. If it finds a match, DMTMPTCK issues return code 0 and passes the corresponding fullword value in R0.

Parameter	Reg	Function
Input Parameters	R0	Pointer to keyword table
	R1	Address of a parameter list, containing: <b>Word 1</b> Pointer to 16-byte keyword area
Output Parameter	R15	Return code:
		<b>0</b> A match was found in the keyword table; R0 contains the address of the keyword.
		<b>4</b> No match was found; R0 contains zero.

## DMTMPTGD

The DMTMPTGD routine converts input parameter string values from decimal EBCDIC to a signed binary fullword. The parameter, which must be an integer value, may be preceded by a plus (+) or minus (-) sign.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Pointer to 16-byte parameter area
		<b>Word 2</b> Pointer to start of previous token
		<b>Word 3</b> Length of previous token
		<b>Word 4</b> Pointer to end of the text to be parsed
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed; R0 contains the converted decimal number.
		<b>4</b> Parameter missing.
		<b>8</b> Parameter too long.
		<b>12</b> Parameter is not a valid decimal or is too large.

## DMTMPTGP

The DMTMPTGP routine parses a parameter from an input string. To identify a keyword, it specifies the 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 uppercase.

Parameter	Reg	Function
Input Parameter	R1	<p>Address of a parameter list, containing:</p> <p><b>Word 1</b> Pointer to 16-byte parameter area</p> <p><b>Word 2</b> Pointer to start of previous token</p> <p><b>Word 3</b> Length of previous token</p> <p><b>Word 4</b> Pointer to end of the text to be parsed</p>
Output Parameter	R15	<p>Return code:</p> <p><b>0</b> A parameter has been parsed successfully and stored in the area pointed to by Word 1. The addresses in Words 2 and 3 are updated to refer to this parameter.</p> <p><b>4</b> No parameter found; the area pointed to by Word 1 contains blanks.</p> <p><b>8</b> Parameter is longer than 16 bytes; it is truncated and stored in the area pointed to by Word 1.</p>

## DMTPAREP

The DMTPAREP routine verifies the syntax of a parameter string. Keywords and options are verified against a keyword table, which the calling routine must supply.

Parameter	Reg	Function
Input Parameter	R1	<p>Address of a parameter list containing:</p> <p><b>Word 1</b> Address of first byte in the parameter</p> <p><b>Word 2</b> Length of the parameter</p> <p><b>Word 3</b> Address of the keyword table</p> <p><b>Word 4</b> Address of a 16-byte field to contain the feedback</p>

Parameter	Reg	Function
Output Parameter	R15	Return code:
		<b>0</b> The field pointed to by the target address in the keyword table is updated.
		<b>4</b> Error found in a keyword. The first 8 bytes of the area pointed to by Word 4 contain the keyword.
		<b>8</b> Duplicate or conflicting keyword found. The first 8 bytes of the area pointed to by Word 4 contain the keyword.
		<b>12</b> Error found in a keyword option. The first 8 bytes of the area pointed to by Word 4 contain the keyword. The second 8 bytes contain the option, which may be truncated.

## DMTPRDDQ

The DMTPRDDQ routine receives information that has been posted by the port redirector task.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing: <b>Word 1</b> Address of the task name <b>Word 2</b> Address of a PRDBLOK to receive the response
Output Parameters		None

## DMTPRDNQ

The DMTPRDNQ routine posts the port redirector task to start or cancel a LISTEN request.

Parameter	Reg	Function
Input Parameter	R1	Address of the PRDBLOK containing the post request
Output Parameters		None

## DMTQSAAB

The DMTQSAAB routine allocates a storage buffer for the calling routine. The characteristics of the storage are defined by the QSABLOK macro (see [“QSABLOK – Defining a Storage Request”](#) on page 286).

Parameter	Reg	Function
Input Parameter	R0	Pointer to the area mapped by QSABLOK
Output Parameter	R15	Return code:
		<b>0</b> Storage buffer acquired. R0 contains the length of the buffer and R1 points to the buffer.
		<b>4</b> Buffer not allocated because of a GETMAIN problem.

## DMTQSAFA

The DMTQSAFA routine frees all buffers associated with a QSABLOK and releases the storage.

Parameter	Reg	Function
Input Parameter	R0	Pointer to the area mapped by QSABLOK
Output Parameter	R15	Return code 0

## DMTQSAUB

The DMTQSAUB routine frees one storage buffer that is no longer needed by the calling routine. The buffer is chained to the existing queue anchor.

Parameter	Reg	Function
Input Parameters	R0	Pointer to the area mapped by QSABLOK
	R1	Pointer to the buffer to deallocate
Output Parameter	R15	Return code 0

## DMTRDREP

The DMTRDREP routine reads records from input spool files.

Parameter	Reg	Function
Input Parameter	R1	Address of RDR area, containing the following fields:
		<b>RDRTAG</b> Address of file's TAG element
		<b>RDRFIOA</b> Address of the input file I/O area
		<b>RDRLINK</b> LINKTABL address
		<b>RDRDREA</b> Address of area to contain input record
		<b>RDRMAX</b> Maximum logical record length (non-NJE links)
		<b>RDRIFLG</b> One of the following values:
		<b>X'04'</b> Work station input
		<b>X'02'</b> 3211-type FCB input

Parameter	Reg	Function
Output Parameter	R15	Return code:
		<b>0</b> Normal processing completed; fields set as follows: <b>RDRAREA</b> Address of returned logical record <b>RDRLRECL</b> Logical record length <b>RDRCCWOP</b> CCW opcode for this record <b>RDRIFLG</b> One of the following values: <b>X'02'</b> Data found. <b>X'01'</b> CCW is an immediate command. <b>X'04'</b> Alternate input area in use.
		<b>4</b> End of file reached.

## DMTRDROP

The DMTRDROP routine initializes processing of an input spool file.

Parameter	Reg	Function
Input Parameter	R1	Address of RDR area, containing the following fields: <b>RDRTAG</b> Address of file's TAG element <b>RDRFIOA</b> Address of the input file I/O area <b>RDRLINK</b> LINKTABL address (non-NJE links only) <b>RDRDREA</b> Address of area to contain input record <b>RDRMAX</b> Maximum logical record length (non-NJE links only) <b>RDRIFLG</b> One of the following values: <b>X'04'</b> Work station input <b>X'02'</b> 3211-type FCB input <b>X'10'</b> CCW optimization requested

Parameter	Reg	Function
Output Parameter	R15	Return code:  <b>0</b> Noraml processing completed; fields set as follows: <b>RDRAREA</b> Address of returned logical record <b>RDRLRECL</b> Logical record length  <b>4</b> No storage available for alternate input area (for NJE requests only).

## DMTRERSC

The DMTRERSC routine determines if a file, command, or message should be rerouted based on its node, user ID, and request type.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Address of FOR node (ignored for NOTRCVG messages) <b>Word 2</b> Address of FOR user (ignored for CMDS) <b>Word 3</b> Address of flag bytes <b>Word 4</b> Address of TO node <b>Word 5</b> Address of FOR user (ignored for CMDS) <b>Word 6</b> Address of MSGBLOK, or 0 for forced QUIET processing
Output Parameter	R15	Return code:  <b>0</b> Data should be rerouted to the specified TO node and user ID.  <b>4</b> The data should not be rerouted.

## DMTRESLO

The DMTRESLO routine lets a routine get exclusive use of RSCS spool or file resources or a resource defined by an exit routine.

Parameter	Reg	Function
Input Parameter	R1	Pointer to the RESBLOK that describes the resource
Output Parameters		None

## DMTRESUN

The DMTRESUN routine removes a lock held on a resource by a routine.

Parameter	Reg	Function
Input Parameter	R1	Pointer to the RESBLOK that describes the resource
Output Parameters		None

## DMTSEPBL

The DMTSEPBL routine formats block letters for printed output. Each letter is 10 characters high by 9 characters wide.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list, containing: <b>Word 1</b> Address of a 1-byte field that contains the current line number, which is updated on return <b>Word 2</b> Address of the character string to be printed <b>Word 3</b> Address of a 1-byte field containing the length of the string <b>Word 4</b> Address of the output area for processed string
Output Parameters		None

## DMTSOKET

The DMTSOKET routine provides access to the RSCS TCP/IP socket interface.



Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing:
		<b>Word 1</b> An integer that identifies the socket function.
		<b>Word 2</b> Address of SOCKBLOK created by the INITIALIZE function.
		<b>Word 3</b> Pointer to the ECB to post when the socket call completes.
		<b>Word 4</b> Address of a word to receive the TCP/IP return code; if not needed, this may be a null (0) pointer.
		<b>Word 5</b> Address of a word to receive the TCP/IP error number; if not needed, this may be a null (0) pointer.
		<b>Word 6</b> Address of an 8-character string to receive the unique call ID associated with the socket call. This ID is needed if the CANCEL request is used to terminate the call; if not needed, this may be a null (0) pointer.
		<b>Word 7</b> Address of parameter 1, if applicable.
		<b>Word 8</b> Address of parameter 2, if applicable.
		<b>Word 9</b> Address of parameter 3, if applicable.
		<b>Word 10</b> Address of parameter 4, if applicable.
		<b>Word 11</b> Address of parameter 5, if applicable.
		<b>Word 12</b> Address of parameter 6, if applicable.
Output Parameters	R1	Return code
		<b>0</b> Socket function call completed.
		<b>4</b> Socket function call has started.
		<b>8</b> Temporary error has occurred; retry the function call later.

## DMTTASKA

The DMTTASKA routine adds a task entry to the RSCS task table or to a chain of entries anchored at the task table.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing: <b>Word 1</b> TASKFLAG value to identify the task: <b>1</b> System task <b>2</b> Link driver task <b>3</b> Auto-answer task <b>Word 2</b> Pointer to the main control block associated with the identified task (SYSIDENT, LINKTABL, or PORT)
Output Parameter	R15	Return code: <b>0</b> Normal processing completed; R1 points to the added entry. <b>4</b> An error occurred; entry not added.

## DMTTASKD

The DMTTASKD routine deletes entries from any task table.

Parameter	Reg	Function
Input Parameter	R1	Pointer to task ID
Output Parameter	R15	Return code 0

## DMTTASKF

The DMTTASKF routine finds entries in the task table, using their task ID.

Parameter	Reg	Function
Input Parameter	R1	Pointer to task ID
Output Parameter	R15	Return code: <b>0</b> Normal processing completed; R1 contains a pointer to the TASKBLOK. <b>4</b> Error occurred; TASKBLOK not found.

## DMTTASKG

The DMTTASKG routine finds entries in the task table, using the task ID of the current task.

Parameter	Reg	Function
Input Parameters		None

Parameter	Reg	Function
Output Parameters	R15	Return code:  <b>0</b> Normal processing completed; R1 contains a pointer to the TASKBLOK.  <b>4</b> Error occurred; TASKBLOK not found.

## DMTUROEP

The DMTUROEP routine builds a chain of CCWs and data for a unit record output device. It then schedules the I/O operation.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing:  <b>Word 1</b> Address of output file I/O area  <b>Word 2</b> Address of LINKTABL entry
Output Parameter	R15	Return code:  <b>0</b> Normal processing completed.  <b>4</b> Irrecoverable I/O error occurred.

## DMTUROFL

The DMTUROFL routine completes the CCW chain started by DMTUROEP and schedules an I/O operation. It also processes recoverable error conditions, such as reaching the end of page on an FCB-type printer.

Parameter	Reg	Function
Input Parameter	R1	Address of a parameter list containing:  <b>Word 1</b> Address of output file I/O area of the RDEVBLK  <b>Word 2</b> Address of LINKTABL entry
Output Parameter	R15	Return code:  <b>0</b> Normal processing completed.  <b>4</b> Fatal I/O error occurred.

## Nonexecutable Entry Points

This section describes the nonexecutable entry points listed in the CRV that are supported for use by exit routines. The entry points are listed alphabetically.

### DMTASTCM

The internal command ECB for the RSCS autostart task.

**DMTASTCQ**

The anchor associated with the internal command ECB for the RSCS autostart task.

**DMTAXMCM**

The command ECB for the RSCS spool manager task. It can be posted to tell the spool manager task to process a command.

**DMTAXMCQ**

The anchor for the command queue associated with the spool manager task's command ECB.

**DMTBOXPR**

Contains the logo image used for separator page output.

**DMTCOMTN**

Contains the local time zone abbreviation.

**DMTCOMTO**

Contains the address of the time zone offset.

**DMTEVECM**

The command ECB for the RSCS event scheduler task. It can be posted to tell the event scheduler task to process a command.

**DMTEVECQ**

The anchor for the command queue associated with the event scheduler task's command ECB.

**DMTIRWLK**

Contains an empty link table entry, which can be used to start a new entry in the LINKTABL.

**DMTIRWTA**

Contains an empty tag entry, which can be used to start a new TAG table entry.

**DMTIRXHL**

A predefined HASHBLOK, used to add entries to the RSCS hashing tables for links.

**DMTIRXHN**

A predefined HASHBLOK, used to add entries to the RSCS hashing tables for nodes.

**DMTIRXHR**

A predefined HASHBLOK, used to add entries to the RSCS hashing tables for routing groups.

**DMTQSAAU**

A QSABLOK that defines the storage used for AUTHBLOKs.

**DMTQSAEC**

A predefined QSABLOK, which can be used to acquire 256 bytes of storage from a conditional GETMAIN macro invocation.

**DMTQSAEU**

A predefined QSABLOK, which can be used to acquire 256 bytes of storage from an unconditional GETMAIN macro invocation.

**DMTQSAEV**

The QSABLOK that defines the storage used for event blocks.

**DMTQSAMB**

The QSABLOK that defines storage used for message blocks.

**DMTQSAML**

The QSABLOK that defines the storage used for message line areas.

**DMTQSAMW**

The QSABLOK that defines the storage used for message work areas.

**DMTREXCM**

The internal command ECB for the RSCS communications task.

**DMTREXCQ**

The anchor associated with the internal command ECB for the communications task.

**DMTREXME**

The message ECB for the RSCS communications task, it can be posted to tell the communications task to issue a message.

**DMTREXMQ**

The anchor for the message element queues associated with the RSCS communications task.

**DMTREXTE**

The terminate ECB for the RSCS communications task, it can be posted to tell RSCS to end immediately.

**DMTSCTAC**

The access control block for the RSCS/VTAM interface, which communicates with VTAM when running a RSCS session driver.



## Chapter 13. Message Repositories

This section describes the format of the RSCS message repositories. You should follow this format if you modify the repositories supplied with RSCS or create alternate repositories for your installation. This section also describes the MCOMP and MCONV execs, which are used to compile the repositories.

### Conversion Repository

The conversion repository contains all parts of a message that cannot be translated. For example, a conversion repository entry contains the following information about a message:

- Routing code
- Type of data to be displayed
- How to find the data
- How to convert and justify the data in the message text

### Naming Convention

A conversion repository file should have a 6-character file name in the form xxxyyy. When you compile the repository with the MCONV exec (see [“MCOMP and MCONV – Compiling Message Repositories”](#) on page 360), the resulting file is named xxxyyy TEXT. The entry point xxxyyyNX identifies the main index into the repository. The first 3 characters of the EMSG header created for any message issued from this repository are xxx.

If your exit routine issues messages from another repository, it should set the MSGBCONV field in the MSGBLOK to point to that conversion repository. To do so, you can use the RMSG macro or set the field directly.

When supplying a conversion repository for exit packages, you should use a 6-character file name. The first 3 characters should be the same as the first 3 characters of the module names of the exit routines in the package. To identify the conversion repository, use MGC as the next 3 characters.

### Repository Structure

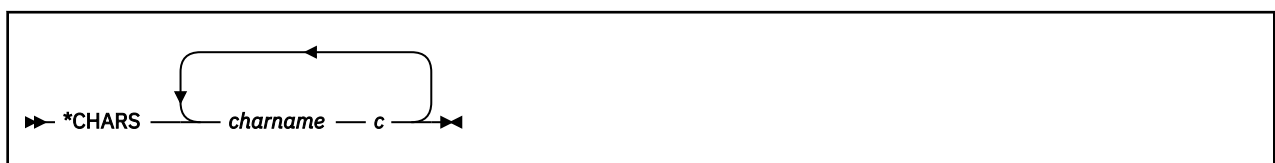
The conversion repository must be a fixed-format, 80-character file with standard sequence numbers. Updates to the file are applied by the CMS UPDATE facility.

The repository contains statements that describe the characteristics of the file itself and the messages it defines. Each statement usually consists of one line in the repository. As the conversion repository is compiled, individual lines, statements, and tokens are identified. Each token represents a keyword, punctuation character, or other element in the message text. If an error occurs as the repository is compiled, MCONV issues a message and a pointer to the line or token that caused the error.

### Control Statements

The conversion repository can contain any of the following control statements.

#### \*CHARS Statement



The \*CHARS statement identifies characters that have a special meaning within the conversion repository. The valid character names, default characters, and their functions are described as follows:

<i>charname</i>	<i>c</i>	Function
SEPCH	\	Separates each line in a message and the items in a dictionary.
INDCH	!	Shows that indirection is needed to access a field in a message.
CONTCH	#	Continues a statement to another line if the last non-blank character (before column 73 or the first @) is a # character. The statement continues to the first non-blank character of the following line. If blanks are needed in the following line, the # character specifies where the line should continue.
IGNORECH	@	When specified, all characters following the first occurrence of the @ are ignored.
SUBCH	\$	Identifies a substitution within the message text.

The compiler inspects columns 1 - 72 of each physical line in the repository. All lines that begin with an asterisk (\*) are considered comment lines and are not processed.

INCLUDE Statement



The INCLUDE statement identifies macros that should be called to generate symbolic references. For example, if you specify the following statement, you could use symbolic references to TAG fields in the conversion repository.

```
INCLUDE TAG
```

Some macros contain more than one DSECT. Only macros that generate DSECTs by default should be specified on an INCLUDE statement; you cannot specify any operands of a macro.

COPY Statement



The COPY statement identifies files that should be copied into the assembler source. This allows any DSECTs within the files to be used for symbolic references. For example, by specifying the following statement, the conversion repository can use the SPOOL and DEVTYPES COPY files to resolve symbolic references.

```
COPY SPOOL DEVTYPES
```

BASE Statement





## Notes:

<sup>1</sup> You can specify up to 12 base names.

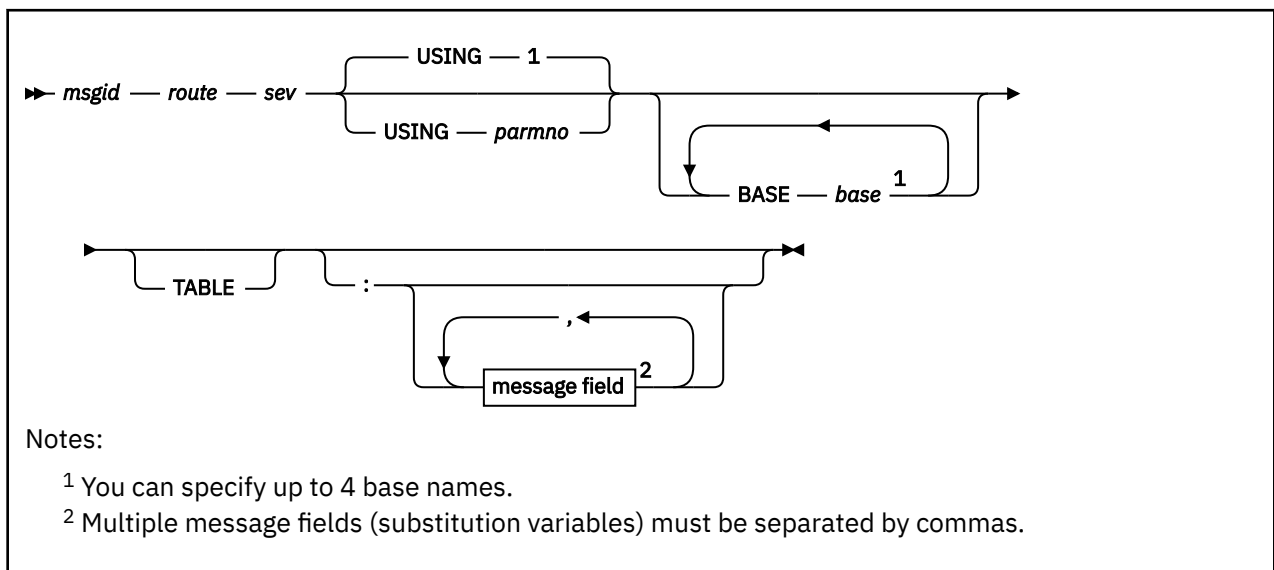
The BASE statement identifies a global base address that converts symbolic references into absolute values. The base address should be the name of a DSECT. Specify one base for each DSECT that requires symbolic references. However, you can define only 12 global bases. (Use the BASE option on the message definition statement to define local bases for a message.)

For example, the following statement identifies the symbol LINKTABL as a global base.

```
BASE LINKTABL
```

Symbolic references to the LINKID can be resolved and converted into the absolute value LINKID-LINKTABL.

## Message Definition Statement



The message definition statement identifies the characteristics of each message. Each message is described by a message ID, routing code, severity code, and optional additional information.

### **msgid**

is a number, 0 - 999, that uniquely identifies the message.

### **route**

specifies the default routing code for the message and shows if it is a private message. The routine that requests the message can override the default routing code by specifying a nonzero MSGBRCD field in the MSGBLOK.

### **R**

Message is sent to RSCS console. If the RSCS DISCONNECT command has been issued with the NOLOG option, the message is not written to the RSCS console. If a user ID was specified on the DISCONNECT command, the user ID also receives the messages.

### **O**

Message is sent to any user ID/node ID designated in the MSGBLOK.

### **V**

Message is sent to any user ID on the local node only.

### **C**

Message is sent to the OPERATOR user ID (local operator)

### **P**

Message is private; it cannot be part of SET or SETMSG subscriptions.

The routing code can be any combination of the preceding characters, with no intervening blanks. For example, the code RC means the message is sent to the local RSCS console and the CP operator. The code VP means that the message is sent to a local user ID and cannot be part of a message subscription.

**sev** is the default severity code of the message. The default can be overridden if the message issuer specifies a nonzero MSGBSCOD field in the MSGBLOK.

- I** Informational message
- W** Warning message
- E** Error message
- S** Severe error message
- T** Terminating error message

**USING parmno** specifies the parameter in a data area that RSCS should use to provide message information if none are explicitly specified.

The *parmno* is a decimal number equal to or greater than 1. It refers to the MSGBLOK variable that contains a pointer to another data area where all non-explicitly coded fields reside. If the USING option is not specified, USING 1 is assumed.

For example, if you do not specify the USING option, the reference LINKID is interpreted by the compiler as 1! (LINKID-LINKTABL). This tells the compiler that the first MSGBLOK variable contains a pointer to a LINKTABL entry. The specified field resides at LINKID-LINKTABL into the field.

However, if you specify USING 3 in the message definition, the compiler would interpret this statement as 3! (LINKID-LINKTABL). This tells the compiler that MSGBLOK parameter 3 contains the pointer to a LINKTABL data area.

**BASE base** specifies DSECT names that are referred to only occasionally by a specific message. You can specify up to four BASE options for a message, but you must repeat the BASE keyword each time. (Use the BASE statement to define up to 12 global data areas. See [“BASE Statement” on page 346.](#))

For example, the following statement identifies the DSECTs AREA1 and AREA2 as local bases for message 200 only.

```
200 V    I BASE AREA1 BASE AREA2 ...
```

**TABLE** specifies that the message is a columnar message (table display). These messages must conform to a special columnar format. The task or routine requesting the message must specify the columns that should be displayed in a message.

Message Fields

Each message definition statement also defines how specific information should be converted and displayed in each field in the message text. To convert each message field, the RSCS message builder must have the following information about the field:

- Source of the data to be converted
- Type of conversion needed
- Required output characteristics of the converted data

A message field is defined by the following structures:

**message field**  

Notes:

<sup>1</sup> This value is required for some data types.

See [“Source Definition”](#) on page 349, [“Data Type Definition”](#) on page 350, and [“Output Definition”](#) on page 355. Some data type definitions require additional information (*typeinfo*). If needed, the *typeinfo* is included in the detailed description of the data type.

Source Definition

The source definition of a message field describes the address and length of the binary data to be converted in the message.

**source**

addr

Tells the compiler where it can find the data to be converted in the message field. The address value can be specified in the following ways:

Method	Explanation
MSGBLOCK parameter  For example: <i>n</i>	Parameters in the MSGBLOK can contain the values to be converted. Each parameter has a number, starting from 1. Parameter 1 is at address MSGBVARS+0, parameter 2 at MSGBVARS+8, and parameter <i>n</i> is at MSGBVARS+( <i>n</i> -1)*8. You can use this number to refer to a MSGBLOK parameter. For example, 1.8 specifies that the first MSGBLOK parameter (MSGBVARS+0) contains the field data, which is 8 bytes long.
Offset from a parameter  For example: <i>n+offset</i>	A parameter that is not aligned at the start of a data area can also point to the address of the field data. Here, you can use the + character to identify the appropriate parameter. For example, 2+4.4 specifies that the source starts 4 bytes after parameter 2 (at MSGBVARS+12) and is 4 bytes long.

Method	Explanation
<p>Data area pointed to by a parameter</p> <p>For example: <i>n!offset</i></p>	<p>A MSGBLOK parameter can also contain a pointer to a data area that is the source of a message field. The ! character identifies when this type of indirection is needed to find the source. For example, 2!8.2 identifies the following information about the source:</p> <p><b>2!</b> The first 4 bytes of MSGBLOK parameter 2 are a pointer.</p> <p><b>8</b> Add 8 to the value of the pointer to find the address of the source.</p> <p><b>.2</b> The length of the source is 2 bytes.</p> <p>You can specify the ! and + symbols several times in a statement to create many levels of indirection.</p>
<p>Symbolic reference</p> <p>For example: <i>n!(label-offset)</i></p>	<p>You can use symbolic references to find information within a data structure. For example, to find a LINKID in a LINKTABL entry, you can specify 1!LINKID. This produces the same results as specifying 1! (LINKID-LINKTABL) . L ' LINKID. The compiler calculates the LINKID-LINKTABL offset by treating LINKID as a relative address, based on the LINKTABL DSECT. The length defaults to the length of the LINKID field.</p> <p>You can specify several symbolic references at the same time. However, when many symbols are used, the last one in the chain is used as the default length. For example, 1!PORTLINK!LINKID is the same as specifying 1! (PORTLINK-PORT) ! (LINKID-LINKTABL) . L ' LINKID. In both cases, the first parameter in the MSGBLOK is treated as a pointer to the PORT entry. The PORTLINK field of this PORT entry is taken as a pointer to a LINKTABL entry. The value of LINKID field of this LINKTABL entry is then used as the message field.</p>
<p>Parameter default</p> <p>For example: <i>n!label</i></p>	<p>If you use the first parameter of a data area to locate the source of a message field, you do not need to specify the initial parameter. For example, LINKID is the same as 1!LINKID. However, if the needed parameter is not parameter 1, you must specify the parameter (for example, LINKID ..., 2!TAGORGID). You can specify the USING option on a message definition to override the default parameter number where none is explicitly specified.</p>

**len**

Length of the value to be converted in the message field. If you specify the *len* as a decimal integer in the message definition, that value is used. If you specify a symbolic reference for the address, the length of the source field is used. For example, if the address is LINKID, the length will be L'LINKID, which is 8. If you do not specify a *len* value, the default length of the selected data type is assumed (see “Data Type Definition” on page 350).

**Data Type Definition**

The data type definition (*typedef*) of a message field determines how the input binary source value should be converted into an EBCDIC string. The output EBCDIC string is also represented by an address and a length. The following table summarizes each data type recognized by RSCS. It also describes the default source length, output length, and justification within the message field. The data types are described in more detail following the table.

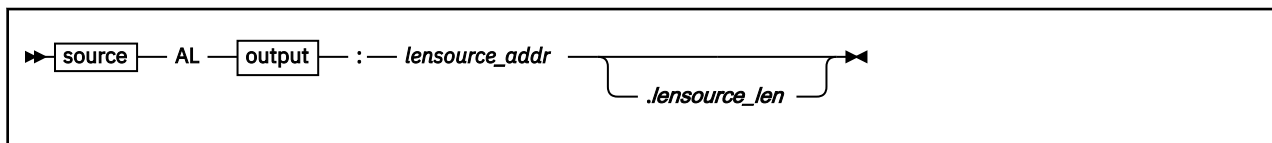
Some data type definitions require additional information (*typeinfo*). If needed, the *typeinfo* is included in the detailed description of the data type.

<b>typedef</b>	<b>Explanation</b>	<b>Source Length</b>	<b>Output Length</b>	<b>Justification</b>
AL	Adlen string	4	As converted	Left
ALH	Adlen string (hidden characters)	4	As converted	Left
ALZ	Adlen string (no ellipsis for null)	4	As converted	Left
C	Character	8	As input	Left
D	Decimal	4	As converted	Right
DB	Decimal (leading zeros)	4	10	Right
DZ	Decimal (leading zeros)	4	10	Right
E	Enumerator	1	As converted	Left
S	Selector	1	As converted	Left
T	TOD clock	8	As converted	Left
W	Word	8	As input	None
X	Hex	4	As converted	Right
XZ	Hex (leading zeros)	4	8	Right

### **AL – Adlen String (Ellipsis)**

Use this data type for converting general variable-length strings. The source *addr* is the address of the start of string text; the source *len* is ignored. If the source address is zero, the string is converted into an ellipsis (...).

To find the area that contains the length of the source, you must specify a *typeinfo* value in the following format:



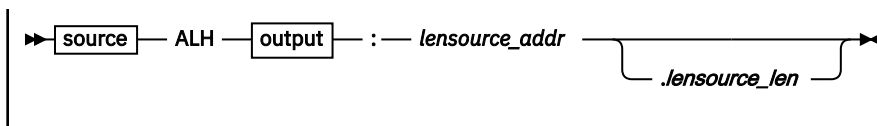
Like the source *addr*, the *len* source address and length must be defined, using indirection and offset values. The *len* source field is treated as an unsigned binary integer that is 1 to 4 bytes long (if longer, it is truncated). The integer is padded with binary zeros to create a fullword value that is used as the length of the string. If the length is zero, an ellipsis is returned. Otherwise, the converted data is the string text, starting at the source address.

For example, to define an Adlen string, you can place the address of the text string and its length in a MSGBLOK parameter. To show that the source pointer for the text is at MSGBVARS+4 and its length is in the fullword at MSGBVARS+0, you can specify the value:

```
1+4!0 AL: 1.4
```

### **ALH – Adlen String (Hidden Characters)**

Use this data type to convert strings that contain information that you do not want to display (for example, passwords). The source *addr* is the address of the start of string text; the length is ignored. If the source address is zero, the string is converted to an ellipsis. Otherwise, you must specify a *typeinfo* value in the following format to find its length:



Like the source *addr*, the *lensource* address and length must be defined, using indirection and offset values. The *lensource* field is treated as an unsigned binary integer that is 1 to 4 bytes long (if longer, it is truncated). The integer is padded with binary zeros to create a fullword value that is used as the length of the string. If the length is zero, an ellipsis is returned. Otherwise, the converted data is the string text, starting at the source address.

For example, to define an Adlen string, you can place the address of the text string and its length in a MSGBLOK parameter. You can then specify the value

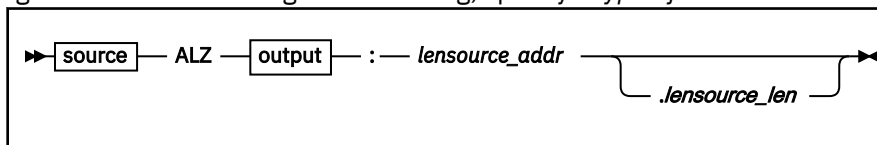
```
1+4!0 AL: 1.4
```

to indicate that the source pointer for the text is at MSGBVAR+4; its length is in the fullword at MSGBVAR+0.

Now, however, all characters that are placed within two occurrences of the hide character or after one hide character will appear as the string XXXX. The default hide character (\) can be changed by the HIDECHARACTER statement, which is described in [z/VM: RSCS Networking Planning and Configuration](#).

### ALZ – Adlen String (No Ellipse)

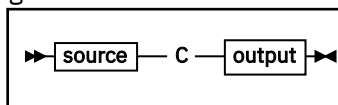
Use this data type to convert general variable-length strings if you do not want to generate an ellipsis when the string is zero. The source address is the address of the start of string text; source length is ignored. To find the length of the string, specify a *typeinfo* value in the following format:



Like the source *addr*, the *lensource* address and length must be defined, using indirection and offset values. The *lensource* field is treated as an unsigned binary integer that is 1 to 4 bytes long (if longer, it is truncated). The integer is padded with binary zeros to create a fullword value that is used as the length of the string. However, an ellipsis is not produced if the length is zero.

### C – Character

Use this data type for fixed-length character data. The source address defines the beginning of the string. The explicitly coded length defines the (fixed) length of the string. The converted output data is the data given as the source.

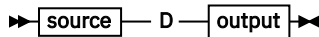


For example, the following entry denotes a field that has a fixed length of one character:

```
TAGCLASS C 1
```

### D – Decimal

Use this data type for general signed decimal numbers. The source address is expected to point at a 1- to 4-byte binary integer. If the source length is greater than 4 bytes, it is truncated. The source value must then be converted into a fullword value. If its original length is 2 bytes, it is interpreted as a signed binary integer and sign-extended. If the length is 1 or 3 bytes, it is treated as an unsigned binary integer and no sign extension takes place.



If the resulting binary fullword is X'80000000', it is converted to an ellipsis. Use this value to represent an area that has not been set or is not applicable. Otherwise, the binary value specified by the source information is converted to an EBCDIC value. The EBCDIC value can contain up to 10 digits and a minus sign (-), if the source value is negative. Leading zeros are removed.

### ***DB – Decimal (Leading Zeros)***

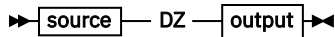
Use this data type to display spool IDs that may contain more than 4 digits. You should specify the output length for the number of digits needed. The source address points to a 1- to 4-byte unsigned binary integer. If the source length exceeds 4 bytes, it is truncated. The resulting number is converted into a 10-digit unsigned decimal number that includes leading zeros.



As the number is being converted to a displayable format, RSCS determines if the 5-digit spool ID support is enabled. This support is enabled by the `OPTION` statement, which is described in [z/VM: RSCS Networking Planning and Configuration](#). If this support is not enabled, which is the default, a 4-digit number will be returned. If the number being converted is greater than 9999 and 5-digit support is not in effect, zeros will be returned. If the 5-digit support is enabled, a number for the specified output length will be returned.

### ***DZ – Decimal (Leading Zeros)***

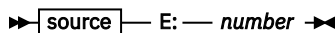
Use this data type to convert decimal numbers that need leading zeros (for example, spool IDs). You should also specify an output length for the number of digits needed. The source address points to the beginning of a 1- to 4-byte unsigned binary integer. If the source length is not 1 - 4 bytes, it is truncated. The resulting number is converted into a 10-digit unsigned decimal number, including leading zeros.



### ***E – Enumerator***

Use this data type to explicitly identify a dictionary item to be used in a message field. The source address points to the beginning of a 1- to 4-byte unsigned binary integer. If the source length is greater than 4 bytes, it is truncated. The source value is then converted to a fullword and padded on the left with binary zeros.

This resulting value is used as an index into a dictionary supplied in the translation repository. The output data and length are taken from this dictionary definition. To prevent RSCS from referencing past the index value, you must identify the number of terms in the dictionary. To do so, specify a *typeinfo* value in the form:



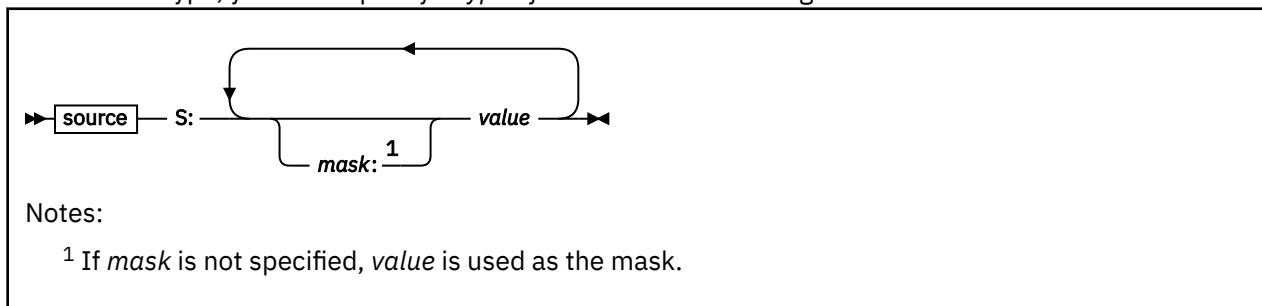
The fullword index into the dictionary must be in the range 0 - *number*-1, where *number* is the number of terms defined. If it is not in this range, the data is converted into the string ???.

When the language-independent form of the CRI is used, dictionary items in the translation repository are not used. Rather, the index is converted into the character string *Dnn*. The index is incremented by one so that *nn* is in the range 01 - *number*. If the index is wrong, the characters D?? are used.

## S – Selector

Use this data type to analyze flag bytes in a data area and dynamically determine the dictionary item to be used in a message text. The source address points to the beginning of a 1- to 4-byte bitmap. If the source length is greater than 4 bytes, it is truncated. The source is then converted to a fullword and padded to the left with binary zeros. The selector value determines the index to use to find a dictionary value. Incorrect indexes are identified by ??? values for regular messages, and D?? values for messages issued with the language independent form of the CRI.

For this data type, you must specify a *typeinfo* value in the following format:



The *mask* and *value* pair tests for certain conditions; if a condition is true, the appropriate item can be selected from a dictionary. The number of *mask* and *value* pairs specified defines the number of dictionary terms that should be defined in the translation repository. If omitted, the *mask* defaults to the specified *value*. A *mask* and *value* can be specified as:

- A hex number, starting with a decimal digit (for example, X'FF' must be specified as 0FF)
- A symbol (for example, LACTIVE)
- A combination of numbers and symbols (for example, LACTIVE+LCONNECT)

For example in the following statement, RSCS chooses the first dictionary term if the LACTIVE flag is set and the LCONNECT flag is not set. RSCS selects the second dictionary term if the LCONNECT flag is set. Finally, if neither flag is set, RSCS selects the third dictionary item.

```
LFLAG S: LACTIVE+LCONNECT:LACTIVE LCONNECT 0
```

## T – TOD Clock

Use this data type to convert TOD clock values into readable formats. The source address points to the beginning of an 8-byte TOD clock produced by an STCK instruction; the source length is ignored.



The TOD clock is converted into the local date and time in the form:

```
yyyymmddhhmmssuuuuuuzzzzzz
```

**yyyy**

Year

**mm**

Month (01 - 12)

**dd**

Day (01 - 31)

**hh**

Hour (00 - 23)

**mm**

Minutes (00 - 59)



**ss**

Seconds (00 - 59)

**uuuuuuu**

Microseconds (000000 - 999999)

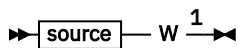
**zzzzzzz**

Time zone, which can contain trailing blanks.

This value is then converted into the TOD format described in the translation repository. When the language-independent form of the CRI is used, however, this TOD format specified in the translation repository is not used.

**W – Word**

Use this data type for identifiers and fields, such as link IDs, user IDs, and node IDs. The source address points to the beginning of a string. The source length is the implied or implicitly stated length of the string. The converted output data is the same as the input data, without leading and trailing blanks. If the input data is all blanks, the output is an ellipsis.

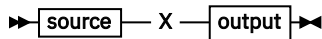


Notes:

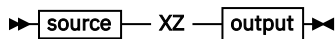
<sup>1</sup> There are no output parameters. The output length defaults to the input length.

**X – Hexadecimal**

Use this data type where you want a hex number with leading zeros suppressed. The source address points to the beginning of a 1- to 4-byte unsigned binary integer. If the source length is greater than 4 bytes, it is truncated. The source is converted to a fullword and padded on the left with binary zeros. The resulting number is converted to an unsigned hexadecimal number of up to 8 digits.

**XZ – Hexadecimal (Leading Zeros)**

Use this data type to display binary fields in hexadecimal form. Usually, you will specify an output length that is twice the input length. The source address points to the beginning of a 1- to 4-byte unsigned binary integer. If the source length is greater than 4 bytes, it is truncated. The source is converted to a fullword and padded on the left with binary zeros. The resulting number is converted to an 8-digit unsigned hexadecimal number, including leading zeros.

**Output Definition**

The output definition of a message field identifies the address and length of the string of EBCDIC characters that is appended to the message text.



If the output *len* specified in the conversion repository is 0, or if no output value is specified, the data is used as it is entered. If the *len* is not 0 and the length of the converted data is greater than *len*, the data is truncated according to the specified *justification* value:

- L** Left
- R** Right
- C** Centered
- N** None

If the length of the converted data is less than *len* and justification is specified, the text is padded with blanks, as appropriate, to the required length. For the language-independent form of the CRI, however, the data is not padded with blanks.

When using the DB, DZ, and XZ data types in text messages, you should specify an appropriate length. For columnar messages, you should specify the output width of each column in the message. You should also ensure that the length of a column header is appropriate for the data to be displayed in that column.

## Translation Repository

---

A translation repository contains all the elements of a message that can be translated into a different language.

Like the conversion repository, a translation repository must be a fixed-format, 80-character file that is compatible with the CMS UPDATE facility. Only columns 1 - 72 of each physical line are inspected. Any @ characters on the line, and any following characters, are ignored. You can also identify special characters in the translation repository with the \*CHARS statement. See [“\\*CHARS Statement” on page 345](#) for more information.

### Naming Convention

A translation repository should have a file name in the form xxxyyyyy and a file type of MSGS. When the repository is compiled by the MCOMP exec, the resulting file is called xxxyyyyy TEXT. The main index to the repository, entry point xxxyyyyy, will then have the alias xxxMSGNX.

The alias name ensures that the translation repository can be linked into the RSCS load module and used as the default message language. A translation repository can also be linked into a separate load module by specifying the following link-editor control statements:

```
INCLUDE xxxyyyyy
ENTRY   xxxyyyyy
NAME    xxxyyyyy
```

When the translation repository is specified on the LANGUAGE configuration statement, it can then be used to issue messages. All RSCS messages implicitly refer to the specified local and network languages. If an exit routine issues messages from another translation repository, it should set a pointer to this repository in the MSGBLOK. To do so, you can use the RMSG macro or set the MSGBTRAN field directly.

If supplying an alternate translation repository with an exit package, the first 3 characters of the repository name should match the first 3 characters of each exit module's name. The remaining characters in the file name should be used to identify the language of the repository.

### Repository Structure

The translation repository contains four types of statements:

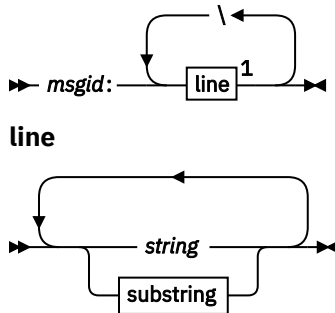
- Text messages
- Columnar, or table-display, messages

- Dictionary items, which contain text strings that describe states or conditions
- TOD-clock formats, which determine how date and time are to be displayed

Each statement is identified by a message ID (*msgid*). For text and columnar messages, the range is 0 - 999, and corresponds to the *msgid* on the message definition statement in the conversion repository. For TOD and DICT statements, the range is 1000 - 9999.

## Text Message Statement

A text message can contain up to nine lines. Each line of the message is separated by the SEPCH character (\). Each line consists of text strings with optional substitutions. Nulls strings contain two substitutions without separating text.

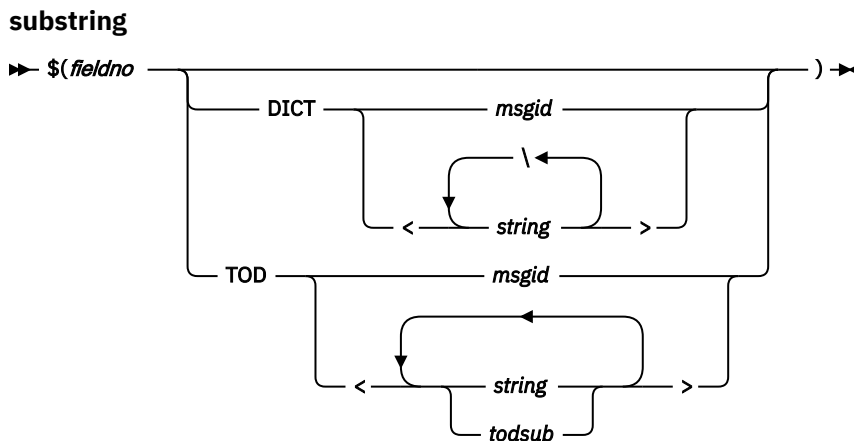


Notes:

<sup>1</sup> You can specify up to 9 lines.

## Substitutions

Each substitution string within a message statement contains a field number (*fieldno*) that refers to the field in the definition statement for that message in the conversion repository. The field specification describes how the data for the substitution can be found and converted.



For enumerator and selector data types, the translation repository contains a dictionary. The dictionary definition can be literal:

```
$(1 DICT <yes\no>)
```

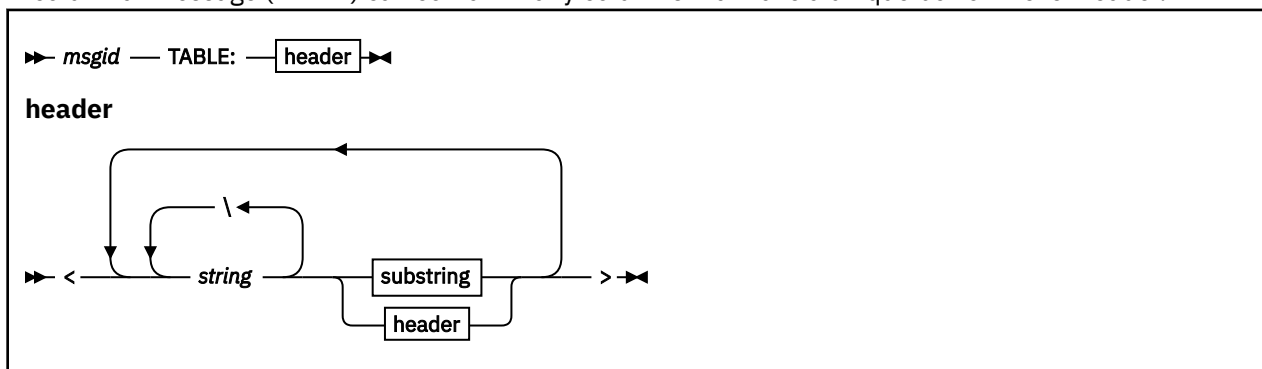
or it can refer to the *msgid* of a dictionary statement:

```
$(3 DICT 1049)
```

For the T data type, the translation repository must supply a TOD clock format. Like dictionaries, TOD clock format definitions can be literal or referenced. For other data types, no other information is needed in the translation repository.

## Columnar Message Statement

A columnar message (TABLE) can contain many columns that have a unique bottom-level header.



You can use the SEPCH (\) to show multi-line header text. You should ensure that the individual lines have the same length. When creating bottom-level headers, you should consider the output width of the fields they represent (specified in the conversion repository).

The order in which the bottom-level headers are entered in the translation repository determines the order in which they are displayed in the columnar message. If the language-independent form of the CRI is used, the columns are presented in the order their corresponding *fields* are defined in the conversion repository. To avoid confusion, you should ensure that the columns and fields are in the same order in each repository.

## Common Headers

Often, when two adjacent bottom-level column headers are selected, one common header can be displayed. Here, you can specify a common (high-level) header for two or more columns. Multiple-column headers can also apply to columns that are already grouped by lower level multicolumn headers.

For example, the following statement can be coded in the translation repository:

```
#      Origin      \Node      Userid  #
<#
  Origin  \Node    $(3)#
  Origin  \Userid  $(4)#
>
```

**Note:** The # symbol is the default line continuation character.

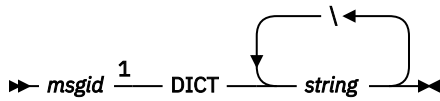
When both columns are selected, the following common header is produced:

```
      Origin
Node      Userid
```

You should ensure that the total width of each line in the multicolumn header text is the same as total width of all bottom-level headers. You should also provide one space or character between each column. Leading blanks may often be needed for multiple-column headers. You should use the continuation character at the beginning to identify needed blanks.

## Dictionary Statement

A dictionary definition item (inline or literal) will translate enumerator or selector data types in the conversion repository. The dictionary is made of a list of strings separated by a SEPCH (\). A string can also contain blanks.

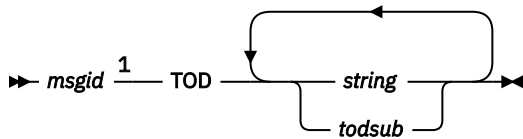


Notes:

<sup>1</sup> Do not specify a : separator following *msgid*.

## TOD Clock Statement

A TOD clock format is required for translating the T data type. Because one TOD format is generally used for all messages, you can define it with a TOD statement.



Notes:

<sup>1</sup> Do not specify a : separator following *msgid*.

You can specify the following substitution (*todsub*) values in a TOD definition:

### **\$YEAR**

The tens-and-units format of the year *yy*

### **\$FULLYEAR**

The full year *yyyy*

### **\$MONTH**

Month (01 - 12)

### **\$DAY**

Day (01 - 31)

### **\$HOUR**

Hour (00 - 23)

### **\$MIN**

Minutes (00 - 59)

### **\$SECOND**

Seconds (00 - 59)

### **\$MILLI**

Milliseconds (000 - 999)

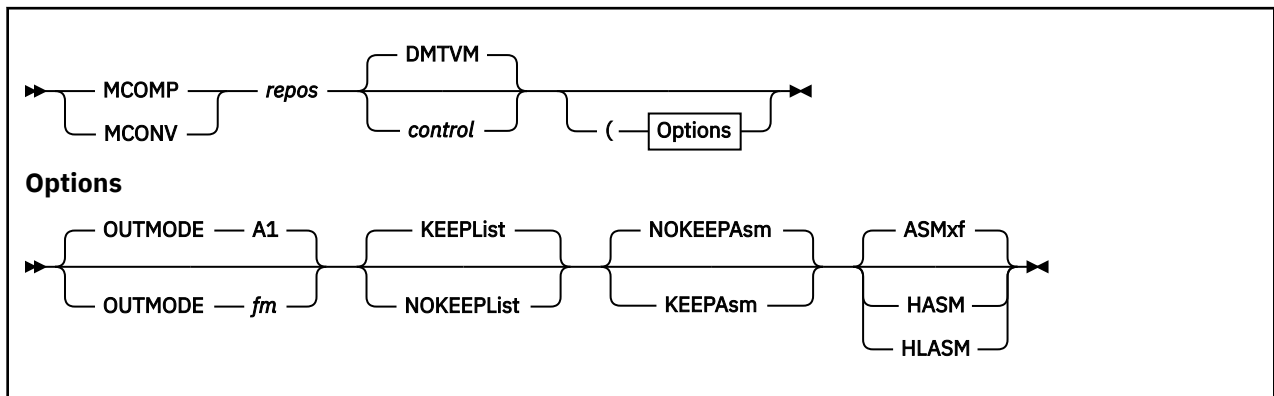
### **\$ZONE**

Time zone indicator

The following is an example of a TOD string containing *todsub* substitution values:

```
$month/$day/$year $hour:$minute:$second $zone
```

## MCOMP and MCONV – Compiling Message Repositories



### Purpose

RSCS supplies two execs to compile the message repositories. The MCONV exec compiles the conversion repository (see [“Conversion Repository”](#) on page 345). The MCOMP exec compiles the translation repository (see [“Translation Repository”](#) on page 356).

### Parameters

#### *repos*

is the file name of the repository to be compiled. The file type differs for each repository; the standard search order determines the file mode.

For translation repositories, the file type must be MSGS. The translation repository supplied with RSCS is DMTAMENG MSGS.

For conversion repositories, the file type must be MCONV. The conversion repository supplied with RSCS is DMTMGC MCONV.

#### *control*

is the file name of the control file used to apply updates to the repository. The default is DMTVM. The file type of the control file is CNTRL. The standard search order determines the file mode.

#### **OUTMODE *fm***

specifies the file mode where the resulting TEXT deck is placed. The default value is A1.

#### **KEEPLIST**

#### **NOKEEPLIST**

specifies whether the compiler listing is kept. The default is KEEPLIST.

#### **NOKEEPAasm**

#### **KEEPAasm**

specifies whether the intermediate assembler source code is kept. The default is NOKEEPAASM. If an error occurs while assembling, however, the source is kept on the disk. You can then reassemble the source code for diagnostic purposes.

#### **ASMxf**

#### **HASM**

#### **HLASM**

specifies the assembler used to assemble the source code. The default is the ASMXF assembler.

### Usage Notes

1. If you modify a message repository supplied with RSCS (DMTAMENG MSGS or DMTMGC MCONV) or create other repositories, take these steps:

- a. Issue MCOMP and MCONV to compile the source file for the appropriate repository into a TEXT deck.
  - b. Rebuild the RSCS load library to include the new TEXT deck.
  - c. Re-IPL RSCS to access the message repository.
2. Each compiler exec must have access to a read/write file mode A. This file mode holds intermediate work files, including:
  - Work files used by the CMS UPDATE command
  - Updated source
  - Compiler listing
  - Intermediate assembler code
  - Text deck, before it is copied to the output file mode

3. RSCS provides the MCOMP and MCONV execs in source and compiled form. The source form of the execs have the file type EXEC; the compiled form have the file type CEXEC.

The execs are installed on the 400 minidisk, which is owned by the RSCS installation user ID. If you want to use the compiled form of the execs, copy and rename the CEXEC files to EXEC files on the 400 disk.

If you have REXX compiler support at your installation, IBM recommends you use the compiled form of the execs.

4. For every 100 lines processed in a message repository, each exec issues a progress message.
5. [z/VM: RSCS Networking Messages and Codes](#) contains the messages that are issued by the RSCS message compilers.





## Chapter 14. Customizing the RSCS Data Interchange Manager

This section describes how you can customize the RSCS Data Interchange Manager (RSCS Interchange) to fit the individual needs of your installation.

### Creating Exit Routines

There are four areas where you can code exit routines for processing of RSCS Interchange:

- Accounting (ACCT)
- Commands (CMD)
- Format Recognition of Mail (FMT)
- Security (SEC)

Each exit routine is passed a set of parameters when it receives control. In return, each exit routine must issue a return code, depending on the action taken by the exit routine.

When an exit area has a group of exit routines to be called, each is passed the same parameters. Unless a nonzero return code is returned from an exit, the next exit in the group is called. If all exits run successfully (return code is 0), processing continues as usual.

Exit routines can be execs (EXEC), modules (MODULE), or compiled REXX execs (CEXECs). Because they run under CMS, CMS services and command interfaces can be used.

After you create an exit routine, you must specify it on an EXIT statement in the RSCS Interchange configuration file. For more information, see [\*z/VM: RSCS Networking Planning and Configuration\*](#).

### Using Accounting Exits

You can code accounting exits to audit files processed. This exit is called after a mail file has been handled (for example, delivered or rejected). If delivered, the original incoming mail file (MAIL MAIL A) as read in by the RSCS Interchange server and outgoing converted note (MAIL NOTE A) still exist on the disk for possible analysis.

**Parameters Passed** (in order shown, separated by blanks):

- RSCSNAME from configuration file
- SMTPNAME from configuration file
- RSCSLINK from configuration file
- DOMAIN name from configuration file
- ADMIN user ID from configuration file
- File Origin
  - If NJE, *userid@nodeid* with no blanks
  - If SMTP, *SMTPaddr* with no blanks
- File Destination
  - If NJE, *userid@nodeid* with no blanks
  - If SMTP, *SMTPaddr* with no blanks
- Disposition of File

**DELIVERED**

File sent to destination.

**FORWARDED**

File transferred to ADMIN.

**RETURNED**

File sent back to originator.

**IGNORED**

Server does not process the file.

**Return Codes:**

Return Code	Results
0	Call next exit; no action taken by this exit.
4	Do not call next exit; process continues normally.

If the z/VM user directory entry for the RSCS Interchange virtual machine includes an OPTION ACCT statement (see *z/VM: RSCS Networking Planning and Configuration*), you can issue DIAGNOSE code X'4C' from your exit routines to create accounting records and pass them to CP to include in the accounting log. For more information about the OPTION ACCT statement, see *z/VM: CP Planning and Administration*. For more information about DIAGNOSE code X'4C', see *z/VM: CP Programming Services*.

## Using Command Exits

You can code command exits, which are called before any command processing, to perform the following tasks:

- Process special commands not recognized by the RSCS Interchange server
- Authorize or restrict specific users
- Handle an existing command

**Parameters Passed** (in order shown, separated by blanks):

- RSCSNAME from configuration file
- SMTPNAME from configuration file
- RSCSLINK from configuration file
- DOMAIN name from configuration file
- ADMIN user ID from configuration file
- Command Origin (*userid@nodeid* with no blanks)
- Command and command text as sent

**Return Codes:**

Return Code	Results
0	Call next exit; no action taken by this exit.
4	Do not call next exit; process continues normally.
8	Reject command because of unauthorized user.
12	Accept command by authorizing user.
16	Exit has handled command; server will ignore.

## Using Format Recognition Exits

You can code format recognition exits to enable RSCS Interchange to recognize mail file types other than CMS, PROFS™, or OfficeVision® notes. These exits can:

- Have the server process and deliver the note.

- Have the exit process the note and have the server deliver it.
- Have the exit process and deliver the note.

This exit is called after the server has read an incoming mail file from spool and before any conversion processing. The spool file is still present and available for analysis. If the mail file is a CMS, PROFS, or OfficeVision note, the file will be written to disk as MAIL MAIL A and is available for analysis.

If the mail file is not a recognized type supported by RSCS Interchange, it must be processed manually from the spool file by the format recognition exit. In addition, if the exit returns with a return code of 0 or 4, the file will be rejected with no further processing. For a format recognition exit to properly handle a file type not recognized, it must return with a return code of 8 (file converted to MAIL NOTE A) or 12 (exit handled).

**Parameters Passed** (in order shown, separated by blanks):

- RSCSNAME from configuration file
- SMTPNAME from configuration file
- RSCSLINK from configuration file
- DOMAIN name from configuration file
- ADMIN user ID from configuration file
- File Origin (*userid@nodeid* with no blanks)
- File Destination (*SMTPaddr* with no blanks)
- Spool ID of the original mail file

**Return Codes:**

Return Code	Results
0	Call next exit; no action taken by this exit.
4	Do not call next exit; process continues normally.
8	Exit has recognized the file and converted it to MAIL NOTE A; the server delivers the file to its destination.
12	Exit has handled the mail file; the server ignores the incoming mail file and purges the spool file.

## Using Security Exits

You can code security exits to restrict certain users or groups of users from sending mail through RSCS Interchange. These exits can tell the server to:

- Process the file itself.
- Transfer the file to the system administrator user ID.
- Allow the exit to handle the file completely.

This exit is called *prior* to reading the file from the spool and writing it to disk as MAIL MAIL and converting it to MAIL NOTE. The spool file is still present and available for analysis.

**Parameters Passed** (in order shown, separated by blanks):

- RSCSNAME from configuration file
- SMTPNAME from configuration file
- RSCSLINK from configuration file
- DOMAIN name from configuration file
- ADMIN user ID from configuration file
- File Origin

- If NJE, *userid@nodeid* with no blanks
- If SMTP, *SMTPaddr* with no blanks
- File Destination
  - If NJE, *userid@nodeid* with no blanks
  - If SMTP, *SMTPaddr* with no blanks
- Spool ID of the original mail file

### Return Codes:

Return Code	Results
0	Call next exit; no action taken by this exit.
4	Do not call next exit; process continues normally.
8	The file is transferred to the system administrator user ID.
12	<p>Exit has handled the mail file and disposed of the spool file accordingly; the server ignores the file.</p> <p>If the spool file is not purged or transferred out of the server's reader by the exit for this return code, the file will be available for processing again by the server. The exit must dispose of the spool file appropriately.</p>

## Appendix A. DSECTs Generated by Mapping Macros

Table 5 on page 367 lists mapping macros (contained in DMTMAC MACLIB) that generate DSECTs for RSCS data areas supported as programming interfaces. For macro invocation formats, see “Control Block Macros” on page 318.

For information about the contents of the data areas, see *z/VM: RSCS Networking Diagnosis*.

<b>Attention</b>
Only the data areas mapped by the macros listed in Table 5 on page 367 are programming interfaces. All other data areas described in <i>z/VM: RSCS Networking Diagnosis</i> are not supported as programming interfaces. Also, some DSECTs generated by the mapping macros might contain fields that are not supported as programming interfaces.

Table 5. Macros That Map RSCS Data Areas	
Macro (Data Area)	Function
ACNTBUFF	Accounting buffer – contains the format of the standard RSCS accounting record.
AUTHBLOK	Authorization table – lists users who are authorized to act as RSCS alternate operators or link operators.
CMNDAREA	Command area – contains information about a specific command or message request.
CRV	Common routines vector table – contains pointers to various RSCS routines that can be used by exit routines.
CVT	Communications vector table – contains information about RSCS data areas, counters, and flags that are available for exit routines to use.
DEST	Destination table – contains a list of PSF destination names.
ECXBLOK	Exit call extension block – stores exit routine return codes.
EVEBLOK	Event block – represents a scheduled RSCS event.
EXITBLOK	Exit block – contains information about an entry point specified for an IBM-defined exit point.
FILREQ	File request block – contains information about a file.
FORM	Form table – describes the characteristics of a print form.
IOTABLE	I/O table – defines a request to write output, either to a line or to the spool.
ITRACREC	Internal trace table record – defines the prefix for each ITRACE record in the internal trace table.
LINKTABL	Link table – describes the characteristics of a specific link in the network.
MSGBLOK	Message request parameter list – contains information about an individual message request, including its number, routing and severity codes, and repository information.
MSGLINE	Message line element – contains one line of the text when building a message.

<i>Table 5. Macros That Map RSCS Data Areas (continued)</i>	
<b>Macro (Data Area)</b>	<b>Function</b>
MSGWA	Message work area – used when building the text of a message.  <b>Note:</b> The CMDAREA macro is a prerequisite for MSGWA because MSGWA contains references to symbols in CMDAREA. When using MSGWA in your exit routine, you must ensure that CMNDAREA precedes MSGWA.
NHDTR	NJE block – defines the formats for NJE headers, trailers, and job set headers.
NJEEQU	NJE equates – contains the networking equates used by all networking link drivers.
NMR	Nodal message record – used by networking link drivers to transmit messages and commands to remote nodes.
NOTEBLOK	NOTIFY link driver control block – contains information important to the NOTIFY link driver exit points.
PORT	Port table – contains the addresses of switched telecommunications lines for auto-answer and auto-dial links.
PRDBLOK	TCP/IP port redirector block – contains information about a port redirector task request.
RDEVBLOK	File request element block – contains information about a file request.
RDR	RDRPARMS control block – contains a parameter list for processing an input spool file.
REROUTE	Reroute control block – each entry describes a reroute definition in the network.
RESBLOK	Resource block – describes an RSCS resource.
RFCBTAB	FCB table – contains printer form information specified on FCB statements.
RIB	Receiver information block – contains information about a message or file being received over one stream of a networking link.  <b>Note:</b> The TAG macro is a prerequisite for RIB because RIB contains references to symbols in TAG. When using RIB in your exit routine, you must ensure that TAG precedes RIB.
ROUTEGRP	Route group table – each entry describes a group of nodes or collection of groups in the RSCS network.
SAFTAG	Store-and-forward TAG – describes the TAG element for store-and-forward files.
SAVEAREA	Register save area block – contains the RSCS register save area and extension.
SEPBLOK	Separator page control block – contains information used by the separator page exit points.
SOCKBLOK	Socket set descriptor block – contains information about a socket set.
SOCKCBLK	Active socket call block – contains information about an active socket call.
SYSIDENT	System ID table – contains information about each RSCS system task.
TAG	TAG element (TAG slot) – contains information about a file enqueued for processing by RSCS, including its origin, destination, network origin time, and record count.
TANBLOK	Task ID number allocation block – contains event task ID numbers.

<i>Table 5. Macros That Map RSCS Data Areas (continued)</i>	
<b>Macro (Data Area)</b>	<b>Function</b>
TANK	TANK block – used by networking link drivers as an intermediate buffer to hold a deblocked output record. There are several forms of the TANK.
TASHADOW	TAG shadow element - represents an inactive file on each link that can send the file.
TASKBLOK	Task block – describes a type of active RSCS task (system, link driver, or auto-answer).
TIB	<p>Transmitter information block – contains information about a message or file being transmitted on one stream of a networking link.</p> <p><b>Note:</b> The NJEEQU and TANK macros are prerequisites for TIB because TIB contains references to symbols in NJEEQU and TANK. When using TIB in your exit routine, you must ensure that NJEEQU and TANK precede TIB.</p>
XABHDR	External attribute buffer – contains the format of the header for an external attribute buffer (XAB) for files destined for an all-points-addressable printer.





## 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 publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/VM.

## **Trademarks**

---

IBM, the IBM logo, and [ibm.com](https://www.ibm.com)<sup>®</sup> 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](https://www.ibm.com/legal/copytrade) (<https://www.ibm.com/legal/copytrade>).

Adobe and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or 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) ([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](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](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](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](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](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](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

:OPTIONS. statement [14](#), [34](#)  
\*CHARS repository statement [345](#)  
\*CHARS statement [356](#)  
\*USER\* node [84](#)  
&-symbols [86](#)  
%LEPARMS statement [33](#)

## Numerics

31-bit addressing [34](#)  
31-bit enablement [13](#), [36](#)  
3270P-type links  
    output verification (exit 46) [130](#)  
5-digit spool IDs [353](#)

## A

abend  
    dumps, suppressing (exit 35) [110](#)  
accept request processing  
    external transmission algorithms [139](#)  
    internal transmission algorithms  
        transmission algorithm 0 [143](#)  
        transmission algorithm 1 [144](#)  
        transmission algorithms 2-F [144](#)  
ACCEPT socket call [307](#)  
accounting  
    accepting or rejecting a spool file (exit 21) [83](#)  
    accepting spool files (exit 2) [45](#)  
    adjusting information (exit 48) [134](#)  
    driver initialization (exit 47) [132](#)  
    link state changes (exit 26) [94](#)  
    output page (exit 45) [128](#)  
    purging files (exit 4) [49](#)  
    receiving files (exit 5) [50](#)  
    records  
        ACNTBUFF macro [319](#)  
        creating [40](#)  
        RSCS Interchange [363](#)  
    sending files (exit 3) [47](#)  
    sign-on attempts (exit 9) [57](#)  
    sign-on rejection (exit 10) [59](#)  
    sign-on time outs (exit 7) [54](#)  
    terminating a link (exit 44) [127](#)  
    unrecognizable data (exit 8) [55](#)  
    verification of pages (exit 46) [130](#)  
ACNTBUFF data area [40](#)  
ACNTBUFF macro [40](#), [319](#)  
adding format table records [272](#)  
Adlen string data type format [351](#), [352](#)  
AL data type [351](#)  
ALH data type [351](#)  
ALIAS statements [28](#)  
allocation routines, storage [25](#)  
altering command authorization levels [82](#)

ALZ data type [352](#)  
ASCII printer and plotter exits  
    ASCXONE sample module  
        configuration file [165](#)  
        EPARM parameters [163](#)  
    ASCPSE sample module  
        configuration file [160](#)  
        CP SPOOL FORM, using [161](#)  
        EPARM parameters [158](#)  
    attention interrupt processing routine [154](#)  
    calling conventions [14](#)  
    device reset routine [152](#)  
    initialization routine [148](#)  
    language requirements [13](#)  
    link-editing considerations [33](#)  
    message processing routine [153](#)  
    print record vector [148](#)  
    programming considerations [147](#)  
    record processing routine [151](#)  
    sample modules  
        ASCX749E [165](#)  
        ASCXDSOE [157](#)  
        ASCXDWRE [157](#)  
        ASCXONE [163](#)  
        ASCPROP [157](#)  
        ASCPSE [158](#)  
        ASCXSPWE [157](#)  
        ASCXZETE [166](#)  
        sending files with sample exit routines [166](#)  
    tag processing routine [150](#)  
    termination routine [155](#)  
ASCII printers and plotters  
    exit processing [147](#)  
ASCX749E exit routine [165](#)  
ASCXDSOE exit routine [157](#)  
ASCXDWRE exit routine [157](#)  
ASCXONE configuration file [165](#)  
ASCXONE sample exit routine module [163](#)  
ASCPROP exit routine [157](#)  
ASCPSE configuration file [160](#)  
ASCPSE sample exit routine module [158](#)  
ASCXSPWE exit routine [157](#)  
ASCXZETE exit routine [166](#)  
assembling routines  
    ASCII printer and plotter [5](#)  
    gateway programs [5](#)  
    LPD exits [6](#)  
    LPR exits [6](#)  
    RSCS exit facility [3](#)  
    UFT exits [7](#)  
    UFTD exits [8](#)  
AUTHBLOK macro [319](#)  
auto-answer  
    sign-on rejection (exit 10) [59](#)  
    sign-on time out (exit 7) [54](#)  
    sign-on validation (exit 9) [57](#)  
    unrecognizable data (exit 8) [55](#)

## B

- BASE option on message definition statement [348](#)
- BASE repository statement [346](#)
- billing information, obtaining [130](#)
- BIND socket call [307](#)
- branch on return code [266](#)
- branch table, generating [266](#)
- BRC macro [266](#)
- buffer received ECB, gateway program [172](#)
- building format tables [273](#)

## C

- C data type [352](#)
- calling a routine [288](#)
- calling exit routines using VMFHLASM and VMFLKED [18](#)
- CANCEL socket call [308](#)
- changing sort priority [103](#)
- channel-command opcodes [79](#)
- character data type format [352](#)
- choosing an exit [9](#)
- CLOSE socket call [308](#)
- CMNDAREA (command execution request buffer)
  - mapping macro [319](#)
- CMNDAREA macro [319](#)
- coding considerations
  - calling conventions [14](#)
  - distribution considerations [34](#)
  - issuing messages [17](#)
  - link-editing considerations [33](#)
  - linkage conventions [16](#)
  - problem solving [33](#)
  - restoring registers [16](#)
  - samples
    - creating a new command [22](#)
    - defining printing shifts [18](#)
    - exit routine communication [19](#)
    - mapping a work area [26](#)
- standard
  - entry conditions [39](#)
  - exit conditions [39](#)
  - return codes [39](#)
- storage considerations [25](#)
- using
  - data areas [17](#)
  - RSCS routines [17](#)
  - storage allocation routines [25](#)
- warning [1](#)
- command ECB, gateway program [172](#)
- command processing
  - post-CP command screening (exit 25) [91](#)
  - screening (exit 19) [81](#)
  - screening CP spooling commands (exit 24) [88](#)
  - spool manager commands [109](#)
  - unknown commands (exit 29) [99](#)
- command response interface [22](#)
- common problems and solutions [33](#)
- communication between exit routines
  - example [19](#)
  - user fields [31](#)
- compiling message repositories [360](#)
- composite stream identifiers [139](#)
- configuration file

- configuration file (*continued*)
  - ASCXONE [165](#)
  - ASCXPSE [160](#)
  - LPDXMANY [226](#)
  - LPRXONE [203](#)
  - LPRXPSE [206](#)
  - statements, RSCS
    - DUMP [110](#)
    - EXIT 2, [34](#)
    - ITRACE [35](#)
    - LINKTYPE 8, [34](#)
    - PARM [137](#)
    - REROUTE [101](#)
  - UFTXIN [259](#)
  - UFTXOUT [242](#)
- CONNECT socket call [308](#)
- continuation characters [346](#)
- control block
  - macros [318](#)
- control statements, repository [345](#)
- conventions
  - linkage [16](#)
  - message repository, naming [345](#), [356](#)
- conversion repositories
  - compiling [360](#)
  - control statements [345](#)
  - data type definition [350](#)
  - message definition statement [347](#)
  - message field [348](#)
  - naming conventions [345](#)
  - output justification [355](#)
  - special characters [345](#)
- COPY repository statement [346](#)
- copyright notification, generating [292](#), [293](#)
- CP commands
  - DUMP [110](#)
  - extending command functions [88](#)
  - SPOOL [161](#), [208](#)
  - VMDUMP [110](#)
- creating exit points [267](#)
- CRI responses [22](#)
- CRV (common routines vector table)
  - description [17](#)
  - executable entry points
    - DMTAXMRQ [321](#)
    - DMTBPLLX [321](#)
    - DMTCOMDG [322](#)
    - DMTCOMDQ [322](#)
    - DMTCOMFI [322](#)
    - DMTCOMGG [323](#)
    - DMTCOMGN [323](#)
    - DMTCOMHG [324](#)
    - DMTCOMLK [324](#)
    - DMTCOMNQ [324](#)
    - DMTCOMSM [325](#)
    - DMTCOMTE [325](#)
    - DMTCOMTS [326](#)
    - DMTDDLEP [326](#)
    - DMTHASHA [327](#)
    - DMTHASHB [327](#)
    - DMTHASHC [327](#)
    - DMTHASHD [328](#)
    - DMTHASHF [328](#)
    - DMTHASHG [328](#)

CRV (common routines vector table) (*continued*)

executable entry points (*continued*)

DMTHASHS [329](#)  
DMTIOTHD [329](#)  
DMTIOTST [329](#)  
DMTLOGCL [329](#)  
DMTLOGEP [330](#)  
DMTMANDE [330](#)  
DMTMGFFM [330](#)  
DMTMGXEP [331](#)  
DMTMPTBP [331](#)  
DMTMPTCK [332](#)  
DMTMPTGD [332](#)  
DMTMPTGP [332](#)  
DMTPAREP [333](#)  
DMTPRDDQ [334](#)  
DMTPRDNQ [334](#)  
DMTQSAAB [334](#)  
DMTQSAFA [335](#)  
DMTQSAUB [335](#)  
DMTRDREP [335](#)  
DMTRDROP [336](#)  
DMTRERSC [337](#)  
DMTRESLO [337](#)  
DMTRESUN [337](#)  
DMTSEPBL [338](#)  
DMTSOKET [338](#)  
DMTTASKA [339](#)  
DMTTASKD [340](#)  
DMTTASKF [340](#)  
DMTTASKG [340](#)  
DMTUROEP [341](#)  
DMTUROFL [341](#)

mapping macro [319](#)

nonexecutable entry points [341](#)

CRV macro [319](#)

CVT (communications vector table)

description [17](#)

mapping macro [319](#)

user field [31](#), [42](#)

CVT macro [319](#)

## D

### D data type [352](#)

#### data areas

list of [367](#)

using with exit routines [40](#)

data buffer processing [128](#)

data record vector [232](#), [246](#)

#### data set headers, NJE

creation (exit 12) [64](#)

post-processing (exit 42) [123](#)

reception (exit 15) [72](#)

tracing [278](#)

transmission (exit 38) [115](#)

data type definition, message repository [350](#)

DB data type [353](#)

decimal data type format [352](#), [353](#)

DEFINE command [107](#)

#### defining

end of keyword table [281](#)

entry points [290](#)

exit points [267](#)

#### defining (*continued*)

hash tables [270](#)

keyword [282](#)

keyword options [284](#)

keyword table [279](#)

message characteristics [347](#)

module work area [302](#), [303](#)

modules [297](#)

printing shifts, example [18](#)

return points, module [295](#)

storage requests [286](#)

#### Diagnose codes

08

exit 24 [90](#)

exit 25 [91](#)

4C

exit 10 [60](#)

exit 2 [46](#)

exit 21 [84](#)

exit 26 [95](#)

exit 3 [48](#)

exit 4 [49](#)

exit 44 [127](#)

exit 45 [129](#)

exit 46 [131](#)

exit 47 [133](#)

exit 48 [135](#)

exit 5 [51](#)

exit 7 [54](#)

exit 8 [56](#)

exit 9 [58](#)

tracing [275](#)

dictionary items [358](#)

distributing exit routines [34](#)

distribution lists, modifying [86](#)

DMTASTCM entry point [341](#)

DMTASTCQ entry point [342](#)

DMTAXMCM entry point [342](#)

DMTAXMCQ entry point [342](#)

DMTAXMRQ routine [321](#)

DMTBOXPR entry point [342](#)

DMTBPLLX routine [321](#)

DMTCOMDG routine [322](#)

DMTCOMDQ routine [322](#)

DMTCOMFI routine [322](#)

DMTCOMGG routine [323](#)

DMTCOMGN routine [323](#)

DMTCOMHG routine [324](#)

DMTCOMLK routine [324](#)

DMTCOMNQ routine [324](#)

DMTCOMSM routine [325](#)

DMTCOMTE routine [325](#)

DMTCOMTN entry point [342](#)

DMTCOMTO entry point [342](#)

DMTCOMTS routine [326](#)

DMTDDLEP routine [326](#)

DMTEVECM entry point [342](#)

DMTEVECQ entry point [342](#)

DMTHASHA routine [327](#)

DMTHASHB routine [327](#)

DMTHASHC routine [327](#)

DMTHASHD routine [328](#)

DMTHASHF routine [328](#)

DMTHASHG routine [328](#)

- DMTHASHS routine [329](#)
- DMTIOTHD routine [329](#)
- DMTIOTST routine [329](#)
- DMTIRWLK entry point [342](#)
- DMTIRWTA entry point [342](#)
- DMTIRXHL entry point [342](#)
- DMTIRXHN entry point [342](#)
- DMTIRXHR entry point [342](#)
- DMTLOGCL routine [329](#)
- DMTLOGEP routine [330](#)
- DMTMACEX MACLIB [36](#)
- DMTMANDE routine [330](#)
- DMTMGFFM routine [330](#)
- DMTMGXEP routine [331](#)
- DMTMPTBP routine [331](#)
- DMTMPTCK routine [332](#)
- DMTMPTGD routine [332](#)
- DMTMPTGP routine [332](#)
- DMTPAREP routine [333](#)
- DMTPRDDQ routine [334](#)
- DMTPRDNQ routine [334](#)
- DMTQSAAB routine [334](#)
- DMTQSAAU entry point [342](#)
- DMTQSAEC entry point [342](#)
- DMTQSAEC QSABLOK [25](#)
- DMTQSAEU entry point [342](#)
- DMTQSAEU QSABLOK [26](#)
- DMTQSAEV entry point [342](#)
- DMTQSAFA routine [335](#)
- DMTQSAMB entry point [342](#)
- DMTQSAML entry point [342](#)
- DMTQSAMW entry point [342](#)
- DMTQSAUB routine [335](#)
- DMTRDREP routine [335](#)
- DMTRDROP routine [336](#)
- DMTRERSC routine [337](#)
- DMTRESLO routine [337](#)
- DMTRESUN routine [337](#)
- DMTREXCM entry point [342](#)
- DMTREXCQ entry point [342](#)
- DMTREXME entry point [343](#)
- DMTREXMQ entry point [343](#)
- DMTREXTE entry point [343](#)
- DMTSTACT entry point [343](#)
- DMTSEPBL routine [338](#)
- DMTSOKET routine [338](#)
- DMTTASKA routine [339](#)
- DMTTASKD routine [340](#)
- DMTTASKF routine [340](#)
- DMTTASKG routine [340](#)
- DMTUROEP routine [341](#)
- DMTUROFL routine [341](#)
- DMTMEX CNTRL [36](#)
- driver initialization, SNA3270P [132](#)
- DSECT socket call [308](#)
- DUMP command [110](#)
- dump processing (exit 35) [110](#)
- DZ data type [353](#)

## E

- E data type [353](#)
- ECBs, monitoring [171](#)
- ECXBLOK macro [319](#)

- enabling sample exit routines [36](#)
- entry points
  - defining [290](#)
  - identifying [28](#)
- enumerator data type format [353](#)
- ESA mode [13](#), [36](#)
- ESTAE exit processing [110](#), [172](#)
- event control blocks, monitoring [171](#)
- EVENTS CONFIG file [18](#)
- events, tracing [274](#)
- example
  - creating a new command [22](#)
  - defining printing shifts [18](#)
  - mapping a work area [26](#)
  - SOCKET macro specification [307](#)
  - using two exit routines [19](#)
- execs
  - MCOMP [360](#)
  - MCONV [360](#)
  - VMFHLASM [18](#)
  - VMFLKED [18](#)
- exit 0 (initialization)
  - description [41](#)
  - sample exit routine [19](#), [32](#)
  - use with exit 33 [108](#)
- exit 1 (termination)
  - description [43](#)
  - sample routine [26](#)
  - specifying on EXIT statement [35](#)
- exit 10 (auto-answer sign-on reject) [59](#)
- exit 11 (NJE job header creation)
  - description [61](#)
  - sample exit routine [20](#)
- exit 12 (NJE data set header creation) [64](#)
- exit 13 (NJE job trailer creation) [67](#)
- exit 14 (NJE job header reception)
  - description [69](#)
  - sample routine [26](#)
- exit 15 (NJE data set header reception) [72](#)
- exit 16 (NJE job trailer reception) [74](#)
- exit 17 (separator page selection) [76](#)
- exit 18 (separator page generation) [78](#)
- exit 19 (command screening) [81](#)
- exit 2 (spool file accept accounting) [45](#)
- exit 20 replacement [10](#)
- exit 21 (spool file accept/reject) [83](#)
- exit 22 (NOTIFY driver note selection) [85](#)
- exit 23 (NOTIFY driver note editing) [86](#)
- exit 24 (spooling CP command screening) [88](#)
- exit 25 (post-CP command screening) [91](#)
- exit 26 (link state accounting) [94](#)
- exit 27 (message request screening) [96](#)
- exit 28 (message language selection) [98](#)
- exit 29 (unknown command)
  - description [99](#)
  - sample exit routine [22](#)
- exit 3 (spool file send accounting) [47](#)
- exit 30 (reroute interception) [101](#)
- exit 31 (sort priority change)
  - description [103](#)
  - sample exit routine [18](#)
- exit 32 (NMR reception) [105](#)
- exit 33 (user parm processing) [107](#)
- exit 34 (spool manager command) [109](#)

- exit 35 (dump processing) [110](#)
- exit 36 (NOTIFY driver purge) [112](#)
- exit 37 (NJE job header transmission) [113](#)
- exit 38 (NJE data set header transmission) [115](#)
- exit 39 (NJE job trailer transmission) [117](#)
- exit 4 (spool file purge accounting) [49](#)
- exit 40 (NJE record reception) [119](#)
- exit 41 (NJE job header post-processing) [121](#)
- exit 42 (NJE data set header post-processing) [123](#)
- exit 43 (NJE job trailer post-processing) [125](#)
- exit 44 (link termination) [127](#)
- exit 45 (output page accounting) [128](#)
- exit 46 (verification of page accounting) [130](#)
- exit 47 (driver initialization) [132](#)
- exit 48 (verification of output page error) [134](#)
- exit 5 (spool file receive accounting) [50](#)
- exit 6 (TAG priority change) [52](#)
- exit 7 (auto-answer sign-on time out) [54](#)
- exit 8 (auto-answer unrecognizable data) [55](#)
- exit 9 (auto-answer sign-on validation) [57](#)
- exit conditions, exit facility [39](#)
- exit facility summary table [9](#)
- exit packages
  - packaging considerations [27](#)
  - sample [36](#)
- exit points
  - defining [2](#), [290](#)
  - definition [1](#)
  - IBM-defined [3](#), [39](#)
  - installation-defined [3](#)
  - invoking, illustration [3](#)
  - providing [267](#)
  - writing routines [14](#)
- exit routines
  - ASCII-type links [147](#)
  - calling [18](#)
  - definition [1](#)
  - LPD-type links [211](#)
  - LPR-type links [187](#)
  - RSCS exit facility [39](#)
  - TCPASCII-type links [147](#)
  - transmission algorithms [137](#)
  - UFT-type links [231](#)
  - UFTD-type links [245](#)
- EXIT statement
  - defining exit points [2](#)
  - FIRST parameter [35](#)
- EXITCALL macro
  - format [267](#)
  - tracing invocations [277](#)
- extending command functions [88](#)
- external transmission algorithms
  - accept request [139](#)
  - installing [145](#)
  - open requests [138](#)
  - programming considerations [137](#)
  - select requests [141](#)
- external writer name [157](#), [163](#), [201](#)

## F

- FCNTL socket call [308](#)
- file arrival ECB, gateway program [171](#)
- FILREQ macro [319](#)

- finding RSCS routines [17](#)
- flag fields, printer [188](#)
- format table, building [273](#)
- format, save area [15](#)
- FREEMAIN macro [14](#), [25](#)
- function byte values [138](#)
- function parameters, SOCKET macro [307](#)

## G

- gateway programming interface
  - calling conventions [14](#)
  - ECBs, monitoring [171](#)
  - entry conditions [167](#)
  - exit 37 [113](#)
  - exit 38 [115](#)
  - exit 40 [119](#)
  - exit conditions [168](#)
  - gateway service macros
    - NJEABORT [175](#)
    - NJECLOSE [176](#)
    - NJECONCT [177](#)
    - NJEDSCON [178](#)
    - NJEGET [179](#)
    - NJEOPEN [180](#)
    - NJEPUT [182](#)
    - NJERJECT [183](#)
  - language requirements [13](#)
  - link-editing [169](#)
  - link-editing considerations [33](#)
  - NJE file control block fields [184](#)
  - NJE file control block macros
    - NJEFILE [185](#)
    - NJEFILED [186](#)
  - program structure [170](#)
  - reason codes [173](#)
  - return codes [169](#)
  - WORK parameter, specifying [169](#)
  - work, types of [170](#)
- GCS
  - GLOBAL statement [27](#)
  - LOADCMD [28](#), [29](#)
  - macros
    - FREEMAIN [14](#), [25](#)
    - GETMAIN [14](#), [25](#)
    - IDENTIFY [28](#), [29](#)
  - subpools [25](#)
  - GETCLIENTID socket call [309](#)
  - GETHOSTBTNAME socket call [309](#)
  - GETHOSTID socket call [309](#)
  - GETHOSTNAME socket call [309](#)
  - GETMAIN macro [14](#), [25](#)
  - GETPEERNAME socket call [310](#)
  - GETSOCKNAME socket call [310](#)
  - GETSOCKOPT socket call [310](#)
  - GIVESOCKET socket call [311](#)
  - GLOBAL statement, GCS [27](#)

## H

- HASHBLOK macro [270](#)
- hexadecimal data type format [355](#)
- hidden characters [351](#)

## I

### IBM-defined exits

- accounting [40](#)
- data areas [40](#)
- definition [3](#)
- entry conditions [39](#)
- exit 0 (initialization processing) [41](#)
- exit 1 (termination processing) [43](#)
- exit 10 (auto-answer sign-on reject) [59](#)
- exit 11 (NJE job header creation) [61](#)
- exit 12 (NJE data set header creation) [64](#)
- exit 13 (NJE job trailer creation) [67](#)
- exit 14 (NJE job header reception) [69](#)
- exit 15 (NJE data set header reception) [72](#)
- exit 16 (NJE job trailer reception) [74](#)
- exit 17 (separator page selection) [76](#)
- exit 18 (separator page generation) [78](#)
- exit 19 (command screening) [81](#)
- exit 2 (spool file accept accounting) [45](#)
- exit 21 (spool file accept/reject) [83](#)
- exit 22 (NOTIFY driver note selection) [85](#)
- exit 23 (NOTIFY driver note editing) [86](#)
- exit 24 (spooling CP command screening) [88](#)
- exit 25 (post-CP command screening) [91](#)
- exit 26 (link state change accounting) [94](#)
- exit 27 (message request screening) [96](#)
- exit 28 (message language selection) [98](#)
- exit 29 (unknown command) [99](#)
- exit 3 (spool file send accounting) [47](#)
- exit 30 (reroute interception) [101](#)
- exit 31 (sort priority change) [103](#)
- exit 32 (NMR reception) [105](#)
- exit 33 (user parm processing) [107](#)
- exit 34 (spool manager command) [109](#)
- exit 35 (dump processing) [110](#)
- exit 36 (NOTIFY driver purge) [112](#)
- exit 37 (NJE job header transmission) [113](#)
- exit 38 (NJE data set header transmission) [115](#)
- exit 39 (NJE job trailer transmission) [117](#)
- exit 4 (spool file purge accounting) [49](#)
- exit 40 (NJE record reception) [119](#)
- exit 41 (NJE job header post-processing) [121](#)
- exit 42 (NJE data set header post-processing) [123](#)
- exit 43 (NJE job trailer post-processing) [125](#)
- exit 44 (link termination) [127](#)
- exit 45 (output page accounting) [128](#)
- exit 46 (verification of page accounting) [130](#)
- exit 47 (driver initialization) [132](#)
- exit 48 (verification of output page error) [134](#)
- exit 5 (spool file receive accounting) [50](#)
- exit 6 (TAG priority change) [52](#)
- exit 7 (auto-answer sign-on time out) [54](#)
- exit 8 (auto-answer unrecognizable data) [55](#)
- exit 9 (auto-answer sign-on validation) [57](#)
- exit conditions [39](#)
- return codes [39](#)
- summary table [9](#)
- identifiers, stream [139](#), [142](#)
- identifying exit routines to RSCS
  - ASCII printer and plotter exits [5](#)
  - entry points [28](#), [29](#)
  - gateway programs [5](#)
  - load libraries [27](#)

### identifying exit routines to RSCS (*continued*)

- LPD exits [6](#)
- LPR exits [6](#)
- transmission algorithms [4](#), [137](#)
- UFT exits [7](#)
- UFTD exits [8](#)
- INCLUDE repository statement [346](#)
- initialization processing (exit 0) [41](#)
- INITIALIZE socket call [311](#)
- INSTALIT macro [272](#)
- installation-defined exits [3](#)
- intercepting reroutes [101](#)
- internal transmission algorithms
  - installing [145](#)
  - programming considerations [142](#)
  - summary [11](#)
  - transmission algorithm 0
    - accept requests [143](#)
    - open request [143](#)
    - select requests [143](#)
  - transmission algorithm 1
    - accept requests [144](#)
    - open request [143](#)
    - select request [144](#)
  - transmission algorithms 2-F [144](#)
- IOCTL socket call [311](#)
- IOTABLE macro [318](#)
- IRREMSG, routing code [97](#)
- issuing messages [17](#), [299](#)
- ITFORMAT macro [273](#)
- ITRACE macro [274](#)
- ITRACREC macro [319](#)

## J

### job headers, NJE

- creation (exit 11) [61](#)
- post-processing (exit 41) [121](#)
- reception (exit 14) [69](#)
- tracing [278](#)
- transmission (exit 37) [113](#)

### job trailers, NJE

- creation (exit 13) [67](#)
- post-processing (exit 43) [125](#)
- reception (exit 16) [74](#)
- tracing [278](#)
- transmission (exit 39) [117](#)

### justification, repository output [355](#)

## K

### keyword

- defining [282](#)
- defining end of table [281](#)
- defining options [284](#)
- defining table [279](#)

## L

- language requirements [13](#)
- language selection, message [98](#)
- leading zeros, data type format [353](#)
- limiting file transmission [104](#)



- link state accounting (exit 26) [94](#)
- link termination processing (exit 44) [127](#)
- link-editing
  - %LEPARMS statement options [33](#)
  - considerations [33](#)
  - external transmission algorithms [145](#)
  - gateway program [169](#)
  - problem solving [33](#)
  - using ALIAS statements [28](#)
- linkage conventions [16](#)
- LINKTABL (link table)
  - description [367](#)
  - mapping macro [319](#)
  - tracing [276](#)
  - user field [31, 42](#)
- LINKTABL macro [319](#)
- LINKTYPE statement [8](#)
- LISTEN socket call [312](#)
- lists, distribution [86](#)
- load libraries
  - characteristics [33](#)
  - identifying [27–29](#)
  - separating [28](#)
- loadable link drivers
  - calling conventions [14](#)
  - language requirements [13](#)
  - link-editing considerations [33](#)
  - summary [8, 12](#)
- loading exit routines using VMFHLASM and VMFLKED [18](#)
- logging messages [96](#)
- LPD exits
  - control file routine [222](#)
  - data processing routine [218](#)
  - end of file routine [220](#)
  - initialization routine [213](#)
  - LPDXMANY sample module
    - configuration file [226](#)
    - control file commands [225](#)
    - EPARM parameters [226](#)
  - print command processing routine [214](#)
  - print job command processing routine [216](#)
  - print record vector [212](#)
  - print server, using LPD-type link as [230](#)
  - programming considerations [211](#)
  - sample module [224](#)
  - termination routine [223](#)
- LPDXMANY configuration file [226](#)
- LPDXMANY sample exit routine module [224](#)
- LPR exits
  - control file routine [197](#)
  - end of file routine [195](#)
  - initialization routine [189](#)
  - LPRXONE sample module
    - configuration file [203](#)
    - EPARM parameters [201](#)
  - LPRXPSE sample module
    - configuration file [206](#)
    - CP SPOOL FORM, using [208](#)
    - EPARM parameters [204](#)
  - print record vector [189](#)
  - printer flag fields [188](#)
  - programming considerations [187](#)
  - record processing routine [193](#)
  - sample modules

- LPR exits (*continued*)
  - sample modules (*continued*)
    - LPRXONE [201](#)
    - LPRXPSE [204](#)
  - TAG processing routine [191](#)
  - termination routine [199](#)
- LPRXONE configuration file [203](#)
- LPRXONE sample exit routine module [201](#)
- LPRXPSE configuration file [206](#)
- LPRXPSE sample exit routine module [204](#)

## M

- macros
  - control block [318](#)
  - gateway service
    - NJEABORT [175](#)
    - NJECLOSE [176](#)
    - NJECONCT [177](#)
    - NJEDSCON [178](#)
    - NJEGET [179](#)
    - NJEOPEN [180](#)
    - NJEPUT [182](#)
    - NJERJECT [183](#)
- GCS
  - FREEMAIN [14, 25](#)
  - GETMAIN [14, 25](#)
  - IDENTIFY [28, 29](#)
- NJE file control block
  - NJEFILE [185](#)
  - NJEFILED [186](#)
- program structure
  - BRC [266](#)
  - EXITCALL [267](#)
  - HASHBLOK [270](#)
  - INSTALIT [272](#)
  - ITFORMAT [273](#)
  - ITRACE [274](#)
  - PARDSECT [279](#)
  - PAREND [281](#)
  - PARKEY [282](#)
  - PAROPT [284](#)
  - QSABLOK [286](#)
  - RCALL [288](#)
  - RENTY [290](#)
  - REXIT [295](#)
  - RMOD [297](#)
  - RMSG [299](#)
  - RWORK [302](#)
  - RWORKEND [303](#)
  - SOCKET [304](#)
- specifying parameters [265](#)
- managing auto-answer links
  - exit 10 (auto-answer sign-on reject) [59](#)
  - exit 7 (auto-answer sign-on time out) [54](#)
  - exit 8 (auto-answer unrecognizable data) [55](#)
  - exit 9 (auto-answer sign-on validation) [57](#)
- mapping a work area, example [26](#)
- message
  - calling conventions [24](#)
  - issuing [17, 299](#)
  - language selection (exit 28) [98](#)
  - repositories
    - compiling [360](#)

- message (*continued*)
  - repositories (*continued*)
    - conversion [345](#)
    - sample entries [24](#)
    - structure [345](#), [356](#)
    - translation [356](#)
  - rerouting (exit 30) [101](#)
  - screening requests (exit 27) [96](#)
  - using RMSG macro [17](#), [299](#)
- message examples, notation used in [xviii](#)
- migration considerations [13](#)
- modifying
  - distribution lists [86](#)
  - messages [96](#), [98](#)
  - NOTIFY driver notes [85](#), [86](#)
- module
  - entry points, defining [290](#)
  - name, defining [297](#)
  - return point, defining [295](#)
  - work area, defining [302](#), [303](#)
- MSGBLOK (message request parameter list)
  - command processing [100](#)
  - description [367](#)
  - example [20](#)
  - mapping macro [319](#)
  - message processing [96](#)
  - parameters [349](#)
  - tracing [276](#)
  - user field [31](#)
- MSGBLOK macro [319](#)
- MSGLINE macro [319](#)
- MSGWA macro [319](#)
- multistreaming [137](#)

## N

- NHDTR macro [319](#)
- NJE reason code responses [173](#)
- NJE record processing
  - creation
    - data set headers (exit 12) [64](#)
    - job headers (exit 11) [61](#)
    - job trailers (exit 13) [67](#)
  - post-processing
    - data set headers (exit 42) [123](#)
    - job header (exit 41) [121](#)
    - job trailers (exit 43) [125](#)
  - reception
    - data set headers (exit 15) [72](#)
    - job header (exit 14) [69](#)
    - job trailers (exit 16) [74](#)
    - nonheader records (exit 40) [119](#)
  - transmission
    - data set headers (exit 38) [115](#)
    - job headers (exit 37) [113](#)
    - job trailers (exit 39) [117](#)
- NJE sub record control byte values [172](#)
- NJEABORT macro [175](#)
- NJECLOSE macro [176](#)
- NJECONCT macro [177](#)
- NJEDSCON macro [178](#)
- NJEEQU macro [319](#)
- NJEFILE macro [185](#)
- NJEFILED macro [186](#)

- NJGET macro [179](#)
- NJEOPEN macro [180](#)
- NJEPUT macro [182](#)
- NJERJECT macro [183](#)
- NMR (nodal message record)
  - closing streams, restriction [176](#)
  - generating a file control block [185](#)
  - obtaining [179](#)
  - opening transmission streams [180](#)
  - reception (exit 32) [105](#)
  - SCRB representation [171](#)
- NMR macro [319](#)
- nonpersistent subpools [25](#), [27](#)
- notation used in message and response examples [xviii](#)
- NOTEBLOK (NOTIFY link driver control block)
  - description [368](#)
  - macro format [319](#)
  - mapping macro [319](#)
  - user field [31](#), [85](#), [87](#)
- NOTEBLOK macro [319](#)
- notification, copyright [292](#), [293](#)
- NOTIFY-type links
  - file purging (exit 36) [112](#)
  - note editing (exit 23) [86](#)
  - note selection (exit 22) [85](#)
  - queuing [84](#)
  - queuing files [46](#)

## O

- obtaining storage [25](#)
- opcodes, channel-command [79](#)
- open request processing
  - external transmission algorithms [138](#)
  - internal transmission algorithms
    - transmission algorithm 0 [143](#)
    - transmission algorithm 1 [143](#)
    - transmission algorithms 2-F [144](#)
- operating system linkage conventions [16](#)
- OS linkage conventions [16](#)
- output page
  - accounting [128](#)
  - error verification [134](#)
  - verification on 3270P-type links [130](#)

## P

- packages, sample exit routines [36](#)
- packaging exit routines
  - communication considerations [31](#)
  - distribution considerations [34](#)
  - transmission algorithms [144](#)
- page account verification [130](#)
- PARDSECT macro [279](#)
- PAREND macro [281](#)
- PARKEY macro [282](#)
- PAROPT macro [284](#)
- passing control to a routine [288](#)
- persistent subpools [25](#), [27](#)
- PORT (port table)
  - description [368](#)
  - mapping macro [319](#)
  - tracing [276](#)



- PORT macro [319](#)
- PRDBLOK macro [319](#)
- print information, obtaining [128](#)
- print output link
  - termination processing (exit 44) [127](#)
- print record vector [148](#), [189](#), [212](#)
- print server, using LPD-type link as [230](#)
- printer flag fields [188](#)
- printing shifts, defining [18](#)
- processing dumps [110](#)
- processing NJE user sections
  - in data set headers
    - exit 12 (creation) [64](#)
    - exit 15 (reception) [72](#)
    - exit 38 (transmission) [115](#)
  - in job headers
    - exit 11 (creation) [61](#)
    - exit 14 (reception) [69](#)
    - exit 37 (transmission) [113](#)
  - in job trailers
    - exit 13 (creation) [67](#)
    - exit 16 (reception) [74](#)
    - exit 39 (transmission) [117](#)
  - in non-header records
    - exit 40 (reception) [119](#)
- processing, data buffers [128](#)
- PROFILE GCS [30](#)
- programming considerations
  - exit routines [27](#)
  - gateway programs [169](#)
  - identifying entry points [28](#)
  - separating load modules [28](#)
  - transmission algorithms [137](#), [142](#)
- purging files
  - creating accounting records [49](#)
  - from NOTIFY-type link [112](#)

## Q

- QSABLOK macro
  - example [27](#)
  - format [286](#)
- QSABLOKs
  - defining [286](#)
  - example [27](#)
  - predefined [25](#)
  - tracing [276](#)
- quick storage allocation (QSA) routines [25](#)

## R

- RCALL macro
  - example usage [20](#)
  - format [288](#)
- RDEVBLK (request spool device block) [138](#)
- RDEVBLK macro [319](#)
- RDR macro [319](#)
- READ socket call [312](#)
- reason code responses, NJE [173](#)
- receiver-online ECB, gateway program [172](#)
- receiving NMRs [105](#)
- record characteristics change section
  - data set header reception [72](#), [123](#)

- record characteristics change section (*continued*)
  - data set header transmission [115](#)
- recovery procedures, link [94](#)
- RECV socket call [312](#)
- RECVFROM socket call [313](#)
- reentrant code
  - calling conventions [14](#)
  - definition [13](#)
- registers, restoring [16](#)
- rejecting
  - input files [183](#)
  - nodal message records [105](#)
  - output file [175](#)
  - sign-on attempts [57](#), [59](#)
  - spool files [45](#), [83](#)
  - unknown commands [99](#)
  - UPARM values [107](#)
- RENTY macro
  - calling conventions [14](#)
  - example usage [18](#), [144](#)
  - format [290](#)
- REORDER command
  - in accept request processing [139](#)
  - use in exit 2 [45](#)
  - use in exit 21 [84](#)
- replace for exit 20 [10](#)
- request spool device block (RDEVBLK) [138](#)
- REROUTE command [101](#)
- reroute interception (exit 30) [101](#)
- REROUTE macro [319](#)
- RESBLOK macro [319](#)
- RESCHAIN macro [318](#)
- response examples, notation used in [xviii](#)
- restoring registers [16](#)
- restrictions
  - macro usage [265](#)
  - writing exit routines [1](#)
- return code
  - exit facility [39](#)
  - gateway program [169](#)
  - TCP/IP [305](#)
- return point, defining [295](#)
- REXIT macro
  - format [295](#)
  - restoring registers [16](#)
- RIB (receiver information block)
  - description [368](#)
  - mapping macro [319](#)
  - user field [31](#)
- RIB macro [319](#)
- RLOADEP macro [318](#)
- RMOD macro
  - calling conventions [14](#)
  - example usage [18](#)
  - format [297](#)
- RMSG macro
  - example usage [24](#)
  - format [299](#)
- ROUTEGRP macro [319](#)
- routing codes, suppressing [97](#)
- RSCS commands
  - screening [81](#), [99](#)
  - spool manager [109](#)
- RSCS Data Interchange Manager

## RSCS Data Interchange Manager (*continued*)

- accounting exits [363](#)
- command exits [364](#)
- creating exit routines [363](#)
- format recognition exits [364](#)
- security exits [365](#)

### RSCS exit facility

- calling conventions [14](#)
- description [1](#)
- distributing [34](#)
- entry conditions [39](#)
- exit facilities [39](#)
- exit summary table [9](#)
- installation-defined exits [3](#)
- invoking [3](#)
- language requirements [13](#)
- link-editing considerations [33](#)
- return codes [39](#)
- specifying order [35](#)
- tracing calls [35](#)

RSCSEXIT LOADLIB [36](#)

RWORK macro [302](#)

RWORKEND macro [303](#)

## S

S data type [354](#)

SAFTAG macro [319](#)

### sample exit routine modules

- ASCX749E [165](#)
- ASCXDSOE [157](#)
- ASCXDWRE [157](#)
- ASCXONE [163](#)
- ASCXPROP [157](#)
- ASCXPSE [158](#)
- ASCXSPWE [157](#)
- ASCXZETE [166](#)
- enabling the routines [36](#)
- LPDXMANY [224](#)
- LPRXONE [201](#)
- LPRXPSE [204](#)
- packages
  - summary of [36](#)
  - using [36](#)
- UFTXIN [257](#)
- UFTXOUT [241](#)

### save areas

- format [15](#)
- generating, gateway program [169](#)
- predefined [15](#)

SAVEAREA macro [319](#)

### screening

- CP spool commands (exit 24) [88](#)
- message requests (exit 27) [96](#)
- post-CP spool commands (exit 25) [91](#)
- RSCS commands (exit 19) [81](#)
- unknown commands (exit 29) [99](#)

### select request processing

- external transmission algorithms [141](#)
- internal transmission algorithms
  - transmission algorithm 0 [143](#)
  - transmission algorithm 1 [144](#)
  - transmission algorithms 2-F [144](#)

SELECT socket call [313](#)

### selecting

- message languages [98](#)
- NOTIFY driver notes [85](#)
- separator page styles [76](#)

selector data type format [354](#)

SEND socket call [314](#)

SENDTO socket call [314](#)

### separator page

- generation (exit 18) [78](#)
- selection (exit 17) [76](#)

### SEPBLOK (separator page control block)

- description [368](#)
- macro format [319](#)
- mapping macro [319](#)
- user field [31, 79](#)

SEPBLOK macro [319](#)

### serially reusable

- calling conventions [14](#)
- definition [13](#)

SETSOCKOPT socket call [315](#)

SHIFT command [104, 139](#)

SHUTDOWN command [43](#)

SHUTDOWN socket call [315](#)

### sign-on, auto-answer

- rejection (exit 10) [59](#)
- time out (exit 7) [54](#)
- validation (exit 9) [57](#)

simplifying socket calls [304](#)

### SNA3270P-type links

- driver initialization (exit 47) [132](#)
- special processing (exit 48) [134](#)

SOCKBLOK macro [319](#)

SOCKCBLK macro [319](#)

socket calls, simplifying [304](#)

### SOCKET macro

- format [304](#)
- function parameters [307](#)

SOCKET socket call [316](#)

sort priority, changing [103](#)

specific order [265](#)

specifying exit routine order [35](#)

### spool file processing

- accept accounting (exit 2) [45](#)
- accept/reject (exit 21) [83](#)
- purge accounting (exit 4) [49](#)
- receive accounting (exit 5) [50](#)
- send account record (exit 3) [47](#)
- TAG priority change (exit 6) [52](#)

spool IDs, 5-digit [353](#)

spool manager commands [109](#)

SRCBs (sub record control bytes)

- supported values [172](#)
- use in exit 40 [120](#)

standard return codes, exit facility [39](#)

statements, repository control [345](#)

### storage

- allocation routines [25](#)
- obtaining [25](#)
- releasing [43](#)

### stream identification

- external transmission algorithms [139](#)
- internal transmission algorithms [142](#)

STREAMS values, validating [143](#)

subpools, GCS [25](#)

substitution fields, TEMPLATE [86](#)

substitutions, message [357](#)

subtasks, gateway program [172](#)

summary tables

ASCII printer and plotter exit routines [11](#)

channel-command opcodes [79](#)

CRV routines [321](#)

data areas [367](#)

data type definitions, message repository [350](#)

function byte values [138](#)

IBM-defined exit points [9](#)

internal transmission algorithms [11](#)

LPD exit routines [12](#)

LPR exit routines [12](#)

NJE file control block fields [184](#)

NJE reason code responses [173](#)

routing codes, message [347](#)

samples supplied with RSCS [36](#)

severity codes, message [348](#)

supported SRCB values [172](#)

UFT exit routines [12](#)

suppressing

dumps [110](#)

messages [96](#), [98](#)

separator pages [76](#)

unknown commands [99](#)

syntax diagrams, how to read [xv](#)

SYSIDENT macro [319](#)

SYSIN files

data set headers

creation [64](#)

reception [72](#), [123](#)

transmission [115](#)

NJE file control block [185](#)

opening [180](#)

processing by gateway program [170](#)

SYSOUT files

data set headers

reception [72](#), [123](#)

transmission [115](#)

NJE file control block [185](#)

opening [180](#)

processing by gateway program [170](#)

## T

T data type [354](#)

table

defining end of keyword [281](#)

defining keyword [279](#)

TABLE option, message definition [348](#)

TAG element

changing priority [52](#)

description [368](#)

mapping macro [319](#)

user field [31](#)

TAG macro [319](#)

TAKESOCKET socket call [316](#)

TAPARM values, validating [143](#)

TASHADOW (TAG shadow)

description [369](#)

mapping macro [319](#)

TASHADOW macro [319](#)

task abend processing [110](#)

TASKBLOK macro [319](#)

TCP/IP

LPD exit points [211](#)

LPR exit points [187](#)

return codes [305](#)

socket calls [304](#)

UFT exit points [231](#)

UFTD exit points [245](#)

telling RSCS about exit routines

ASCII printer and plotter exits [5](#)

entry points [28](#), [29](#)

gateway programs [5](#)

load libraries [27](#)

LPD exits [6](#)

LPR exits [6](#)

transmission algorithms [4](#), [137](#)

UFT exits [7](#)

UFTD exits [8](#)

TEMPLATE file [85](#), [86](#)

terminate ECB, gateway program [171](#)

TERMINATE socket call [316](#)

termination processing (exit 1) [43](#)

termination processing, link (exit 44) [127](#)

TIB (transmitter information block)

description [369](#)

mapping macro [319](#)

user field [31](#)

TIB macro [319](#)

time-outs, auto-answer sign-on [54](#)

TOD

clock formats [359](#)

data type format [354](#)

tracing exit routines

ITRACE macro [274](#)

using ITRACE facility [35](#)

translation repositories

columnar messages [358](#)

compiling [360](#)

dictionary items [358](#)

naming convention [356](#)

substitutions [357](#)

text messages [357](#)

TOD clock formats [359](#)

transmission algorithms

calling conventions [14](#)

function byte values [138](#)

introduction [4](#)

invoking [4](#)

language requirements [13](#)

link-editing considerations [33](#)

packaging suggestions [144](#)

programming considerations [137](#)

sample link-edit control file [145](#)

specifying [137](#)

stream identification [139](#), [142](#)

troubleshooting [33](#)

## U

UFT exits

data record vector [232](#)

end of file routine [237](#)

initialization routine [233](#)

programming considerations [231](#)

- UFT exits (*continued*)
  - record processing routine [235](#)
  - sample module [240](#)
  - TAG processing routine [234](#)
  - termination routine [239](#)
  - UFT command routine [238](#)
  - UFT commands [232](#)
  - UFTXOUT sample module
    - configuration file [242](#)
    - EPARM parameters [242](#)
    - UFT commands [241](#)
- UFTD exits
  - command processing routine [250](#)
  - connect processing routine [248](#)
  - data processing routine [252](#)
  - data record vector [246](#)
  - end of file routine [254](#)
  - initialization routine [247](#)
  - programming considerations [245](#)
  - sample module [257](#)
  - termination routine [256](#)
  - UFTXIN sample module
    - configuration file [259](#)
    - EPARM parameters [259](#)
    - UFT commands [258](#)
- UFTXIN configuration file [259](#)
- UFTXIN sample exit routine module [257](#)
- UFTXOUT configuration file [242](#)
- UFTXOUT sample exit routine module [241](#)
- unknown commands (exit 29) [99](#)
- unrecognizable sign-on data [55](#)
- UPARM values, processing [107](#)
- user fields
  - summary table [31](#)
  - use in exit 0 [42](#)
  - use in exit 18 [79](#)
  - use in exit 22 [85](#)
  - use in exit 23 [87](#)
  - use in exit packages [31](#)
- user parameters, DEFINE command [107](#)
- USING option, message definition [348](#)
- using sample exit packages [36](#)

## V

- validating, TAPARM values [143](#)
- verifying output page errors [134](#)
- VMDUMP command [110](#)
- VMFHLASM exec [18](#)
- VMFLKED exec [18](#)
- VMSES/E enablement [36](#)

## W

- W data type [355](#)
- warning
  - macro usage [265](#)
  - writing exit routines [1](#)
- word data type format [355](#)
- work areas
  - considerations, gateway program [169](#)
  - defining module [302](#), [303](#)
  - mapping, example [26](#)

- WORK parameter, specifying [169](#)
- WRITE socket call [316](#)
- writing exit routines
  - calling conventions [14](#)
  - distribution considerations [34](#)
  - issuing messages [17](#)
  - link-editing considerations [33](#)
  - linkage conventions [16](#)
  - problem solving [33](#)
  - restoring registers [16](#)
  - samples
    - creating a new command [22](#)
    - defining printing shifts [18](#)
    - exit routine communication [19](#)
    - mapping a work area [26](#)
  - standard
    - entry conditions [39](#)
    - exit conditions [39](#)
    - return codes [39](#)
  - storage considerations [25](#)
  - using
    - data areas [17](#)
    - RSCS routines [17](#)
    - storage allocation routines [25](#)
  - warning [1](#)

## X

- X data type [355](#)
- XABHDR (external attribute buffer header)
  - description [369](#)
  - mapping macro [319](#)
- XABHDR macro [319](#)
- XZ data type [355](#)





Product Number: 5741-A09

Printed in USA

SC24-6317-73

