z/VM
Version 7 Release 2

CP Programming Services

SC24-6272-04

Note:

Before you use this information and the product it supports, read the information in "Notices" on page 1085.

This edition applies to Version 7.2 of IBM z/VM (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2020-08-12

© Copyright International Business Machines Corporation 1991, 2020.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures xix

Tables xxv

About This Document xxxv
Intended Audience xxxv
Syntax, Message, and Response Conventions xxxv
Where to Find More Information xxxviii
Links to Other Documents and Websites xxxviii

How to Send Your Comments to IBM xxxix

Summary of Changes for z/VM CP Programming Services xli
SC24-6272-04, z/VM 7.2 (September 2020) xli
SC24-6272-03, z/VM 7.1 (January 2020) xli
SC24-6272-02, z/VM 7.1 (June 2019) xli
SC24-6272-01, z/VM 7.1 (April 2019) xli
SC24-6272-00, z/VM 7.1 (September 2018) xlii
SC24-6179-13, z/VM 6.4 (March 2018) xlii
Enhanced Support for External Management of z/VM CPU Resources xlii
SC24-6179-12, z/VM 6.4 (December 2017) xliii
VSwitch Link Aggregation Load Balancing Enhancements xliii
SC24-6179-11, z/VM 6.4 (August 2017) xliii
z/VM Support for the IBM z14 xliii
Extended Address Volume Minidisk Support xliii
SC24-6179-10, z/VM 6.4 (March 2017) xliv
Distributed IUCV Enhancements xliv
NICDEF Security Controls xliv
VMUDQ Macro Identifies SUBCONFIGs That Contain MDISKs xliv
SC24-6179-09, z/VM 6.4 (November 2016) xliv
ESA/390 Removal xliv
zManager Support Removed xlv
Guest Transactional Execution Support xlv
Expanded Storage (XSTORE) Support Removed xlv

Part 1. CP DIAGNOSE Instructions 1

Chapter 1. The DIAGNOSE Instruction in a Virtual Machine 3
Instruction Format 3
Macro Format 4
Privilege Classes 5
Address Translation Modes and Restrictions 5
How Addresses Are Processed 5
How Address Spaces Are Selected 7
How Error Conditions Are Reported 7
Access Exceptions 8
Condition Codes and Return Codes 9
Storage Protection Mechanisms 9
DIAGNOSE Codes That Are Not Programming Interfaces 11

Chapter 2. The IBM-Supplied DIAGNOSE Codes 13
DIAGNOSE Code X'00' – Store Extended-Identification Code 13
Usage Notes 13
Responses 16
DIAGNOSE Code X'04' – Examine Host Storage 16
Usage Notes 17
Responses 19
DIAGNOSE Code X'08' – Virtual Console Function 19
Usage Notes 21
Responses 21
Examples 22
DIAGNOSE Code X'0C' – Pseudo Timer 22
Usage Notes 23
Responses 23
DIAGNOSE Code X'10' – Release Pages 24
Usage Notes 24
Responses 25
DIAGNOSE Code X'14' – Input Spool File Manipulation 25
Subcode X'0000'—Read the Next Spool File Buffer (Data Record) 26
Subcode X'0004'—Read the Next Print Spool File Block 27
Subcode X'0008'—Read the Next Punch Spool File Block 28
Subcode X'000C'—Order a File to the Front of a Queue 28
Subcode X'0010'—Repeat the Active File a Specified Number of Times 28
Subcode X'0014'—Restart an Active File at the Beginning 29
Subcode X'0018'—Backspace One Record 29
Subcode X'001C'—Read the Next Monitor Spool File Block 29
Subcode X'0020'—Read the Next Monitor Spool Record 30
Subcode X'0024'—Read the Last Spool File Buffer 30
Subcode X'0028'—Position a Spool File to the Designated Record 30
Subcode X'002C'—Select a File for Processing and Read the Next Spool Buffer 30
Subcode X'0FFE'—Select the Next File Not Previously Selected 31
Subcode X'0FFF'—Retrieve Next File Descriptor 34
Usage Notes 34
Responses 35
DIAGNOSE Code X'18' – Standard DASD I/O 36
Usage Notes 36
Responses 37
Example 38
DIAGNOSE Code X'20' – 370 Synchronous I/O for DIAGNOSE Support 38
Usage Notes 38
Responses 39
DIAGNOSE Code X'24' – Device Type and Features 39
Usage Notes 40
Responses 41
DIAGNOSE Code X'28' – Dynamic Channel Program Modification 41
Usage Notes 42
Responses 43
DIAGNOSE Code X'34' – Read System Dump Spool File 43
Usage Notes 44
Responses 44
DIAGNOSE Code X'3C' – Activate z/VM CP Directory 44
Usage Notes 45
Responses 45
DIAGNOSE Code X'44' – Voluntary Time Slice End 46
Usage Notes 46
Responses 46
DIAGNOSE Code X'48' – Second Level SVC 76 46

DIAGNOSE Code X'4C' – Generate Accounting Records 47
Usage Notes 48
Responses 49
DIAGNOSE Code X'54' – Control the Function of the PA2 Key 50
Usage Note 51
Responses 51
DIAGNOSE Code X'58' – 3270 Virtual Console Interface 51
Usage Notes 52
Responses 60
DIAGNOSE Code X'5C' – Error Message Editing 60
Usage Notes 61
Responses 61
DIAGNOSE Code X'60' – Determine Virtual Machine Storage Size 62
Responses 62
DIAGNOSE Code X'64' – Named Saved Segment Manipulation 62
Subcode X'00' — LOADSHR 63
Subcode X'04' — LOADNSHR 64
Subcode X'08' — PURGESEG 65
Subcode X'0C' — FINDSEG 65
Subcode X'10' — LOADNOLY 66
Subcode X'18' — SEGEXT 66
Subcode X'20' — LOADSHR (64-Bit) 73
Subcode X'24' — LOADNSHR (64-Bit) 73
Subcode X'2C' — FINDSEG (64-Bit) 73
Subcode X'38' — SEGEXT (64-Bit) 73
Usage Notes 75
Responses 77
DIAGNOSE Code X'70' – Time-of-Day Clock Accounting Interface 80
Usage Notes 81
Responses 81
DIAGNOSE Code X'74' – Saving and Loading an Image Library File 82
Usage Notes 83
Responses 83
DIAGNOSE Code X'7C' – Logical Device Support Facility 84
Usage Notes 86
Responses 86
Logical Device External Interrupt Code X'2402' 88
Logical Device Support Facility Functions 89
DIAGNOSE Code X'84' – Directory Update-in-Place 91
Usage Note 102
Responses 102
DIAGNOSE Code X'88' – Validate User Authorization/Link Minidisk 105
Subcode -1 – Verify Authorization to Use DIAGNOSE Code X'88' 106
Subcode X'00' – Validate User Authorization 106
Subcode X'04' – Link Minidisk 106
Subcode X'08' – Validate User Authorization 108
Responses 109
DIAGNOSE Code X'8C' – Access 3270 Display Device Information 110
Usage Notes 111
Responses 111
DIAGNOSE Code X'90' – Read Symbol Table 112
Usage Note 112
Responses 112
DIAGNOSE Code X'94' – VMDUMP and Symptom Record Service 113
Supported Parameters 113
Dump Address List 116
Usage Notes Regarding Dumping a Virtual Machine 118
Usage Notes Regarding Dump Address Lists 118

Usage Notes Regarding Symptom Records....119
Responses....120
DIAGNOSE Code X'98' – Real I/O....124
LOCK Subfunction....124
UNLOCK Subfunction....125
SSCH-Real Subfunction....126
Block Diagnose X'98' Request....126
Usage Notes....131
Responses....132
DIAGNOSE Code X'9C' – Voluntary Time Slice Yield....133
Usage Notes....133
Responses....134
DIAGNOSE Code X'A0' – Obtain ACI Information....134
Responses....135
DIAGNOSE Code X'A4' – Synchronous I/O (Standard CMS Blocksize)....135
Synchronous Block I/O Parameter List (HCPSBIOP)....136
Block Entries (SBILIST)....141
Usage Notes....141
Responses....142
DIAGNOSE Code X'A8' – Synchronous I/O (for All Devices)....144
Synchronous General I/O Parameter List (HCPSGIOP)....144
Usage Notes....147
Responses....147
DIAGNOSE Code X'B0' – Access Re-IPL Data....149
Re-IPL Information....149
IPL Statement Information....151
Usage Notes....151
Responses....151
DIAGNOSE Code X'B4' – Read/Write/Erase the Virtual Printer XAB....151
Responses....152
DIAGNOSE Code X'B8' – Spool File XAB Manipulation....153
Usage Note....154
Responses....154
DIAGNOSE Code X'BC' – Open and Query Spool File Characteristics....155
Usage Note....158
Responses....158
DIAGNOSE Code X'C8' – Set Language....159
Responses....159
DIAGNOSE Code X'CC' – Save Message Repository....160
Responses....160
DIAGNOSE Code X'D0' – Volume Serial Support....161
Usage Note....162
Responses....162
DIAGNOSE Code X'D4' – Set Alternate User ID....162
Responses....164
DIAGNOSE Code X'D8' – Read Spool File Blocks on System Queues....164
Responses....166
DIAGNOSE Code X'DC' – Control Application Monitor Record Collection....166
Usage Notes....170
Responses....170
DIAGNOSE Code X'E0' – System Trace File Interface....171
Usage Notes....172
Responses....172
Content and Format of Trace Blocks....172
DIAGNOSE Code X'E4' – Return Minidisk Information/Define Full-Pack Overlay....174
Function X'00' and Function X'01'....175
Function X'02'....177
Function X'03'....180

Usage Notes....182
Responses....183
DIAGNOSE Code X'EC' – Query GUEST Trace Status....185
Responses....186
DIAGNOSE Code X'F8' – Spool File Origin Information....187
Usage Notes....188
Responses....188
DIAGNOSE Code X'210' – Retrieve Device Information....189
Virtual/Real Device Characteristics Block....190
Usage Notes....196
Responses....197
DIAGNOSE Code X'218' – Retrieve Real CPU Identification....197
Usage Notes....199
Responses....199
Examples....199
DIAGNOSE Code X'238' – Time-Based Unique Identifiers....201
Usage Notes....201
Responses....201
DIAGNOSE Code X'248' – Copy-To-Primary Service....202
Usage Note....203
Responses....203
DIAGNOSE Code X'250' – Block I/O (Standard Blocksize)....203
Initialize Block I/O to a Device....204
Read/Write to DASD....206
Remove the Block I/O Environment....212
Responses....212
Block I/O External Interruption....215
DIAGNOSE Code X'258' – Page-Reference Services....216
Page-Reference Services....216
DIAGNOSE Code X'260' – Access Certain Virtual Machine Information....222
Subcode X'00000000'....222
Subcode X'00000004'....222
Subcode X'00000008'....223
Subcode X'0000000C'....223
Subcode X'00000010'....224
Responses....224
DIAGNOSE Code X'268' – 370 Accommodation Services....225
Subcode 0 — Convert a BC-mode or mapped PSW to EC mode....225
Responses....226
DIAGNOSE Code X'26C' – Access Certain System Information....226
Subcode X'00000004'—Return the BYUSER ID For a Given User ID....228
Subcode X'00000008'—Return Virtual LAN System Information....228
Subcode X'0000000C'—Return Controller List....231
Subcode X'00000010'—Return Controller Information....232
Subcode X'00000014'—Return Guest LAN List....235
Subcode X'00000018'—Return Guest LAN Information....236
Subcode X'0000001C'—Return Virtual Switch List....238
Subcode X'00000020'—Return Virtual Switch Information....239
Subcode X'00000024'—Return Virtual Port, Virtual NIC or HiperSockets Logical Port Information....253
Subcode X'00000030'—MAC Services....261
Responses....263
DIAGNOSE Code X'270' – Pseudo Timer Extended....263
Usage Note....265
Responses....265
DIAGNOSE Code X'274' – Set Timezone Interrupt Flag....265
Usage Notes....266
DIAGNOSE Code X'27C' –Product Enablement Verification....266

Usage Note....268
Responses....268
DIAGNOSE Code X'288' - Control Virtual Machine Time Bomb....268
Usage Notes....269
Responses....269
DIAGNOSE Code X'290' – Perform Privileged Spool Functions....270
Subcode X'0000' – Fetch Current Page of Open Spool File....270
Subcode X'0004' – Fetch XAB Data from Virtual Printer....272
DIAGNOSE Code X'2A8' – Network Diagnose....274
Operation code X'00' - Query Interface....275
Operation code X'01' - Establish Device Connection....277
Operation code X'02' - Send Data Request....278
Operation code X'03' - Receive Data Request....280
Operation code X'04' - Multicast MAC Registration....280
Operation code X'05' - Network Device Options....281
Responses....283
DIAGNOSE Code X'2CC' – SSI Interface....288
Responses....288
DIAGNOSE Code X'2E0' – SYSEVENT Query Virtual Server (QVS)....290
Usage Notes....290
Responses....290
DIAGNOSE Code X'2FC' – Obtain Certain Guest Performance Data....291
Responses....292

Part 2. The Inter-User Communications Vehicle....295

Chapter 3. IUCV Overview....297
How Addresses Are Processed....297
IUCV Paths....297
IUCV Messages....298
Message Data Transfer....298
Message Identification....299
IUCV External Interrupts....299
Avoiding IUCV External Interrupts....301
Security Considerations....301
Virtual Machine-to-Virtual Machine Communication....302
Using Data in a Buffer....302
Using Data in a Parameter List....304
Using Control Paths....305
Invoking IUCV Functions....306
General Description of IUCV Functions....306
Virtual MP Considerations for IUCV Applications....308
IUCV in a Distributed Environment....309

Chapter 4. IUCV Protocols....311

Chapter 5. IUCV Function Descriptions....317
CP System Services....317
ACCEPT Function....319
CONNECT Function....324
DECLARE BUFFER Function....331
DESCRIBE Function....334
INTERRUPT POLL Function....337
PURGE Function....340
QUERY Function....344
QUIESCE Function....345
RECEIVE Function....348

REJECT Function....352
REPLY Function....355
RESUME Function....362
RETRIEVE BUFFER Function....365
SEND Function....366
SET CONTROL MASK Function....372
SET MASK Function....374
SEVER Function....376
TEST COMPLETION Function....379
TEST MESSAGE Function....383

Part 3. The Advanced Program-to-Program Communication/VM....385

Chapter 6. Overview of the APPC/VM Assembler Interface....387
Overview of APPC/VM Assembler Interface....387
Basics of APPC/VM....387
APPC/VM Paths....387
APPC/VM States....388
APPC/VM Interrupts....388
Invoking APPC/VM Communication Functions....390
Using Basic APPC/VM Functions....392
Starting a Conversation....392
Sending and Receiving Data on the Conversation....393
Ending a Conversation....393
Managing a Resource....393
Revoking a Resource....394
Understanding APPC/VM Parameter Lists....394
Setting for Optional Parameters....395
Parameters Reserved for IBM Use Only....395
Reading the Parameter Lists....395
Formatting the Parameter List with MF=L....395
Registers Altered by APPCVM and IUCV Macro Functions....396
Condition Codes and Return Codes....396
Condition Codes....396
Return Codes....397
Virtual MP Considerations for APPC/VM Applications....398
APPC/VM Sever, Error, and Sense Codes That You Can Get....399
Currently-Defined APPC/VM Sever Codes....399
Sever Codes Generated by VM....400
Currently-Defined Error Codes....402
Currently-Defined Sense Code....403
State Table for APPC/VM Functions....403
Examples of Basic States....406
State Table for Error Conditions....407

Chapter 7. APPCVM Macro Functions....411
Using the Online HELP Facility for APPCVM Functions....411
APPCVM CONNECT....412
APPCVM QRYSTATE (Query State)....447
APPCVM RECEIVE....451
APPCVM SENDCNF (Send Confirm)....465
APPCVM SENDCNFD (Send Confirmed)....471
APPCVM SENDDATA....475
APPCVM SENDERR (Send Error)....490
APPCVM SENDREQ (Send Request)....501
APPCVM SETMODFY (Set Modify)....505
APPCVM SEVER....509

Chapter 8. IUCV Macro Functions for Use in APPC/VM....521
Shared Functions That Can Be Used in CMS....521
Shared Functions That Should Be Avoided in CMS....521
Condition Codes and Return Codes for IUCV Macro Functions....522
Condition Codes....523
Return Codes....523
Using the Online HELP Facility for IUCV Macro Functions....523
IUCV ACCEPT....524
IUCV CONNECT....529
IUCV DCLBFR (Declare Buffer)....533
IUCV DESCRIBE....538
IUCV IPOLL (Interrupt Poll)....540
IUCV QUERY....543
IUCV RTRVBFR (Retrieve Buffer)....548
IUCV SETCMASK (Set Control Mask)....550
IUCV SETMASK....553
IUCV SEVER....556
IUCV TESTCMPL (Test Completion)....562
IUCV TESTMSG (Test Message)....566

Chapter 9. Migrating Programs from IUCV to APPC/VM....567
APPC/VM and IUCV Functions That Work Differently....567
IUCV Functions Not Supported on APPC/VM Paths....568
APPC/VM Functions Not Supported on IUCV Paths....569
Shared APPC/VM and IUCV Functions....569
Shared Functions That Can Be Used in CMS....569
Shared Functions That Should Be Avoided in CMS....569

Chapter 10. APPC Mapped with APPC/VM....573
APPC Conversations....573
Establishing a Conversation....573
APPC/VM Interrupts....573
APPC/VM Conversation States....574
APPC/VM Return Codes....575
APPC Verb Names Mapped to APPC/VM Macro Functions....575
APPC ALLOCATE....576
APPC CONFIRM....578
APPC CONFIRMED....579
APPC DEALLOCATE....580
APPC FLUSH....581
APPC GET_ATTRIBUTES....581
APPC PREPARE_TO_RECEIVE....581
APPC RECEIVE_AND_WAIT....582
APPC REQUEST_TO_SEND....584
APPC SEND_DATA....585
APPC SEND_ERROR....586

Part 4. CP System Services....589

Chapter 11. Access Verification System Service (*RPI)....591
Using the CP Access Control Interface....591
Overview....592
HCPRPI Module....593
HCPRPW Module....596
HCPRPD Module....599
HCPRPE Module for handling DIAGNOSE X'A0'....602

HCPRPF Module....606
HCPRPG Module....606
HCPRPL Module....606
HCPRPP Module....606
HCPRWA Module....607
CP Callable Services for the ACI....607
Summary of CP Modules and Entry Points....608
ACI Security Bits....609
HCPDA0 Module for Updating ACI Security Bits....612
ACIPARMS Control Block....622
CP Calls to the ACI....639

Chapter 12. Account System Service (*ACCOUNT)....699
Establishing Communication....699
Receiving Accounting Records....700
Disconnecting from the Accounting System Service....700
Accounting Record Formats....700
Accounting Records for Virtual Machine Resource Usage (Record Type 1)....701
Accounting Records for Dedicated Devices (Record Type 2)....703
Accounting Records for Temporary Disk Space (Record Type 3)....703
Accounting Records for Journaling (Record Types 04, 05, 06, 08, and 0I)....704
Accounting Records for SNA/CCS (Record Type 07)....708
Accounting Records for Inter-System Facility for Communications (Record Type 09)....708
Accounting Records for logging changes to a user's privilege (Record Type 0A)....710
Accounting Records for virtual disk in storage space (Record Type B)....712
Accounting Records Network Data Transmissions (Record Type C)....713
Accounting Records for CPU Capability (Record Types D and E)....715
Accounting Records for Virtual Machine Resource Usage 2 (Record Type F)....717
Adding Your Own Accounting Records and Source Code....718
User-Initiated Accounting Records (Record Type C0)....718

Chapter 13. Asynchronous CP Command Response System Service (*ASYNCMD)....719
Establishing Communication....719
Message Limits....719
Sending and Receiving Data....719
Record Types....720

Chapter 14. DASD Block I/O System Service (*BLOCKIO)....721
Establishing Communication with the DASD Block I/O System Service....721
IUCV CONNECT to the DASD Block I/O System Service....721
Usage Notes....721
IUCV ACCEPT....722
IUCV SEVER....722
IUCV SEND to *BLOCKIO....723
Single Block I/O....723
Multiple Chained Block I/O....725
Ending Communication with the DASD Block I/O System Service....727

Chapter 15. Error Logging System Service (*LOGREC)....729
Establishing Communications with the Error Logging System Service....729
Receiving LOGREC Records....730
Disconnecting from the Error Logging System Service....730

Chapter 16. Identify System Service (*IDENT)....731
Establishing Communication with the Identify System Service....731
Handling Connection Requests for the Resource or Gateway....733
Communicating with CP....733

*IDENT Interface for Communication with CP's Support for the Family of POSIX exec Functions ... 733
When Your Resource is Revoked ... 735
*IDENT Sever Reason Codes ... 736

Chapter 17. Message System Service (*MSG) ... 739

Chapter 18. Message All System Service (*MSGALL) ... 741

Chapter 19. SCLP System Service (*SCLP) ... 743
Establishing Communication with the SCLP System Service ... 743
Connecting to the SCLP System Service ... 743
Sending Events ... 744
Receiving Events ... 745
Disconnecting from the SCLP System Service ... 745

Chapter 20. Signal System Service (*SIGNAL) ... 747
Establishing Communications with the Signal System Service ... 747
IUCV CONNECT to the Signal System Service ... 747
Sending Signals ... 749
Receiving Signals ... 749
Leaving the Signal System Service ... 750

Chapter 21. Spool System Service (*SPL) ... 751
The AFP Printing *SPL Interface ... 752
Establishing Communication with the Spool System Service ... 752
Virtual Machine Communication to the Spool System Service ... 754
Spool System Service Communication to a Virtual Machine ... 764
The Generic *SPL Interface ... 765
Establishing Communication with the Spool System Service ... 765
Processing a File ... 765
Selecting a File To Be Read ... 766
Transferring Information About a Selected File ... 768
Closing a File ... 771
Clearing an Existing Connection ... 771

Chapter 22. Symptom System Service (*SYMPTOM) ... 773
Connecting to the Symptom System Service ... 773
Receiving Symptom Records ... 774
Disconnecting from the Symptom System Service ... 774

Chapter 23. VM Event System Service (*VMEVENT) ... 775
Establishing Communication with the VM Event System Service ... 775
Connecting to the VM Event System Service ... 775
Receiving Events ... 775
Disconnecting from the VM Event System Service ... 780

Part 5. CP Macros for VM Data Spaces and Other Services ... 781

Chapter 24. VM Data Spaces Overview ... 783
What Are Data Spaces? ... 783
Uses for Data Spaces ... 784
ESA/XC Architecture ... 784
Address Space Support ... 784
Summary of Data Space Operations ... 785
Using Data Spaces in Your Applications ... 786
Creating and Using Data Spaces ... 786

Mapping Minidisks to Address Spaces ... 790
Notifying CP of Future Reference Patterns ... 791

Chapter 25. CP Macros ... 793
Using the Online HELP Facility for CP Macros ... 793
Coding CP Macros ... 793
Preferred Use ... 793
Alternative Methods ... 795
ADRSPACE — Address Space Services ... 797
ADRSPACE CREATE ... 800
ADRSPACE DECLARE ... 803
ADRSPACE DESTROY ... 804
ADRSPACE ISOLATE ... 806
ADRSPACE PERMIT ... 808
ADRSPACE QUERY ... 812
ALSERV — Access List Services ... 815
ALSERV ADD ... 817
ALSERV DECLARE ... 820
ALSERV REMOVE ... 821
DEFWORKA — Define Macro Work Area ... 823
MAPMDISK — Mapping Services ... 824
MAPMDISK DECLARE ... 828
MAPMDISK DEFINE ... 829
MAPMDISK IDENTIFY ... 837
MAPMDISK REMOVE ... 843
MAPMDISK SAVE ... 846
PFAULT Macro -- Page-Fault Handshaking Services ... 852
PFAULT CANCEL ... 857
PFAULT DECLARE ... 858
PFAULT TOKEN ... 859
REFPAGE — Page Reference Services ... 863
REFPAGE DECLARE ... 865
REFPAGE INFORMB ... 866
REFPAGE INFORML ... 872
VMUDQ – VM User Directory Query ... 875

Part 6. Architectural Extensions and Accommodations for Virtual Machines ... 881

Chapter 26. Collaborative Memory Management Assist ... 883
Storage ... 883
Collaborative Memory Management Block State ... 883
Modification of Translation Tables ... 886
Assigned Storage Locations ... 886
Control ... 886
Resets ... 886
Interruptions ... 887
Addressing Exception ... 887
Block-volatility Exception ... 887
Control Instructions ... 888
Program Exceptions ... 888
Storage-key Manipulation Instructions ... 888
TEST PROTECTION ... 888
EXTRACT AND SET STORAGE ATTRIBUTES ... 888
Implications for the DIAGNOSE Instruction and Non-CPU Accesses ... 891
Implications for ESA/390 and ESA/XC Guests ... 892
Implications for Saved Systems and Segments ... 892
Implications for the VMDUMP Command ... 892

Chapter 27. 370 Accommodation Facility Overview 893
Background 893
System/370 Constraints 893
High-level Description 893
When Should 370 Accommodation be Used? 894
Choosing a Level of 370 Accommodation 895
Activating 370 Accommodation 895
Running a Restricted CMS MODULE 897
What is Not Provided by the 370 Accommodation Facility 898
Possible Adverse Effects on a Working Program 898

Chapter 28. 370 Accommodation Facility Definition 901
System/370 Instructions 901
System/370 I/O Instructions 901
SET STORAGE KEY (SSK) 902
INSERT STORAGE KEY (ISK) 902
RESET REFERENCE BIT (RRB) 902
ESA-Family Instructions 902
TEST SUBCHANNEL (TSCH) 902
STORE SUBCHANNEL (STSCH) 902
TEST PENDING INTERRUPTION (TPI) 902
Discarding Vestigial Status 903
Other Instructions 903
DIAGNOSE code X'28' 903
The Interval Timer 903
PSW Conversions 904
BC-mode PSW Conversion 904
BC-mode System Mask Conversion 905
Mapped PSW Conversion 905
PSW Conversions During Interruption Processing 905
Interruption Parameters 906
Special Conditions 907
Presentation of Interruptions 907
Vestigial Status 908
The CMS 370 Accommodation Facility 909
PSW Mapping Algorithm 910

Chapter 29. Store Hypervisor Information (STHYI) Instruction 913
Function Code X'0000' - Processor Capacity Information 914
Common Header Section (INFCHDR) 925
Function Code X'0001' - Hypervisor Environment Information 926
Function Code X'0002' - Guest List 941
Function Code X'0003' - Designated Guest Information 943
Function Code X'0004' - Resource Pool List 950
Function Code X'0005' - Designated Resource Pool Information 952
Function Code X'0006' - Resource Pool Member List 955
Special Conditions, Exceptions, and Usage Notes 957

Part 7. Symptom Record Reporting 959

Chapter 30. Symptom Record Reporting 961
Reporting Software Error Symptoms (Symptom Records) 961
The Format of the Symptom Record 961
Section 1 (Environmental Data) 961
Section 2 (Control Data) 961
Section 2.1 (Component Data) 962

Section 3 (Primary SDB—Structured Data Base—Symptoms)....962
Section 4 (Secondary SDB Symptoms)....962
Section 5 (Free-Format Data)....962
Symptom Strings — SDB Format....962
Notes for Applications Using DIAGNOSE Code X'94' SR Option....962

Appendix A. Data Areas Used by DIAGNOSE Codes....969
Data Areas Used by DIAGNOSE Codes X'24' and X'210'....969
CP370 Device Classes....969
CP370 Device Types....969
CP370 Device Features....971
CP370 Virtual Device Status....971
CP370 Virtual Device Flags....972
Data Areas Used by DIAGNOSE Codes X'14' and X'D8'....972
SFBLOK - VM/SP 370 Spool File Control Block....972
SPLINK - VM/SP 370 Spool File Data Block....976
Extended Spool File Block for DIAGNOSE Code X'D8'....978
External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'....979
Suggested Format for an External Attribute Buffer....979

Appendix B. Sample Programs Using DASD Block I/O System Service....981
Write Program....981
Read Program....984

Appendix C. DIAGNOSE Code X'68' and VMCF....987
DIAGNOSE Code X'68'....987
Usage Notes....987
Responses....988
The Virtual Machine Communication Facility....989
Using the Virtual Machine Communication Facility....990
VMCF Protocol....992
Descriptions of VMCF Functions....995
Invoking VMCF Functions....1000
VMCF User Doubleword....1006
VMCF in an MP Environment....1006
DIAGNOSE Code X'68' Return Codes....1007
Data Transfer Error Codes....1009

Appendix D. The Special Message Facility....1011

Appendix E. Logical Device Support Facility....1013

Appendix F. Reserved DIAGNOSE Codes....1015
DIAGNOSE Code X'40' – Clean-Up After Virtual IPL by Device....1015
Usage Note....1015
Responses....1015
DIAGNOSE Code X'E0' – System Trace File Interface....1015
Subcode X'00000008' — Open (read-only)....1017
Subcode X'0000000C' — Read....1017
Subcode X'00000010' — Close (read-only)....1018
Usage Notes....1019
Trace Block Containing CP Trace Table Entries....1019
DIAGNOSE Code X'214' – Pending Page Release....1019
Responses....1021
DIAGNOSE Code X'23C' – Address Space Services....1021
Create-Space Function....1022

Destroy-Space Function.... 1023
Query-Space Function.... 1023
Permit-Access Function.... 1024
Isolate-Space Function.... 1025
DIAGNOSE Code X'240' – Access List Services.... 1026
Add-ALE Function.... 1026
Remove-ALE Function.... 1027
Responses.... 1028
DIAGNOSE Code X'244' – Mapping Services.... 1028
Identify-pool Function.... 1028
Define-mapping Function.... 1029
Remove-mapping function.... 1030
Save-list Function.... 1031
Responses.... 1032
DIAGNOSE Code X'254' – Access Real Subsystem.... 1032
Hardware Specifications.... 1032
Open CP Connection.... 1033
Close CP Connection.... 1035
Perform I/O.... 1036
Responses.... 1038
Access Real Subsystem External Interruption.... 1040
DIAGNOSE Code X'25C' – Directory Query.... 1040
Usage Notes.... 1043
Responses.... 1044
DIAGNOSE Code X'264' – CP Communication.... 1044
Subcode X'00000000'—Establish CP communication area.... 1045
Subcode X'00000004'—Remove CP communication area.... 1045
DIAGNOSE Code X'278' – Extract XLINK Control Blocks.... 1046
Responses.... 1049
DIAGNOSE Code X'280' – Set POSIX IDs - security values.... 1049
Function EXCSETID - Request changes in POSIX security values for an exec() function call.... 1050
Function EXCSSID - Request changes in saved set-IDs for an exec() function call.... 1051
Responses.... 1052
DIAGNOSE Code X'29C' – Set-POSIX-IDs Services.... 1056
Function SPXFUSER - Set User IDs (UIDs) for the Active Process.... 1056
Function SPXFGRP - Set Group IDs (GIDs) for the Active Process.... 1058
Function SPXFNGRP — Change to a New Group.... 1059
Function SPXFSGID — Change the supplementary group ID list.... 1061
Responses.... 1062
DIAGNOSE Code X'2A0' – Query POSIX IDs.... 1062
Function QPXFPROC - Query Process Attributes.... 1063
Function QPXFUSER - Query the User Database.... 1064
Function QPXFGRP - Query the Group Database.... 1067
Function QPXFSGID - Query the Supplementary Group IDs.... 1069
Function QPXFCONF - Query POSIX Configuration Information.... 1071
Usage Note.... 1072
Responses.... 1072
DIAGNOSE Code X'2A4' – POSIX Process ID (PID) Services.... 1073
Function 0 - Identify the POSIX communication area.... 1073
Function 1 - Allocate a PID.... 1074
Function 2 - Deallocate a PID.... 1075
Responses.... 1075
DIAGNOSE Code X'2AC' – HCD Dynamic I/O.... 1076
Responses.... 1076
DIAGNOSE Code X'2C0' – HMC Data Source Load.... 1079
DIAGNOSE Code X'2C4' – FTP Services.... 1080
FPL.... 1081
Responses.... 1082

Notices.....1085
Programming Interface Information.....1086
Trademarks.....1086
Terms and Conditions for Product Documentation.....1086
IBM Online Privacy Statement.....1087

Bibliography.....1089
Where to Get z/VM Information.....1089
z/VM Base Library.....1089
z/VM Facilities and Features.....1091
Prerequisite Products.....1092

Index.....1093

Figures

1. Example of a DIAGNOSE Code in an Assembler Program..... 4

2. DIAGNOSE X'04' Register Entries.....18

3. The Format of the User-Supplied Areas for the SEGEXT Function..... 67

4. The Format of the User-Supplied Areas for the FINDSPACE Operation.....69

5. The Format of the User-Supplied Areas for the FINDSKEL, FINDSEGA, or FINDNSSA operations..... 70

6. The Format of the User-Supplied Output Area – Member List..... 72

7. The Format of the User-Supplied Areas for a 64-Bit FINDSPACE Operation.....73

8. The Format of the User-Supplied Areas for a 64-Bit FINDSKEL or FINDSEGA Operation for a DCSS or Member Segment, or a 64-Bit FINDNSSA Operation..... 74

9. The Format of the User-Supplied Output Areas for a 64-Bit FINDSKEL or FINDSEGA Operation for a Segment Space.....75

10. 31-bit Base format dump address list (without an address space qualifier)..... 116

11. 31-bit Extended format dump address list (with an address space qualifier)..... 116

12. 64-bit Base format dump address list (without an address space qualifier)..... 116

13. 64-bit Extended format dump address list (with an address space qualifier)..... 117

14. DIAGNOSE X'A8' Synchronous General I/O Parameter List (HCPSGIOP) Format.....145

15. Fields in the VRDCBLOK DSECT..... 191

16. IUCV Two-Way Data Transfer.....299

17. Flow of the IUCV CONNECT/ACCEPT Protocol..... 311

18. Flow of the IUCV SEND/RECEIVE Protocol..... 312

19. Flow of the IUCV SEND/RECEIVE/REPLY Protocol.....314

20. Flow of the IUCV SEND Protocol..... 315

21. Flow of the IUCV SEND/REPLY Protocol..... 315

22. APPCVM CONNECT Input Parameter List..... 416

23. Connection Parameter List Extension..... 418

24. Format of the PIP Variable.....423

25. Format of a PIP Subfield.....423

26. Example Format for a PIP Variable..... 424

27. APPCVM CONNECT Output Parameter List (Connection Complete Interrupt).....426

28. Connection Complete Extended Data..... 430

29. Connection Pending External Interrupt..... 432

30. Connection Pending Extended Data, Part One: VM Area.....435

31. Connection Pending Extended Data, Part Two: FMH5.....438

32. Security Subfield in an Attach FMH5 for VM..... 439

33. Connection Pending Extended Data, Part Three: VM-Defined Variable-Length Section..... 440

34. VM Communication Server Area.....445

35. APPCVM QRYSTATE Input Parameter List.....448

36. APPCVM QRYSTATE Output Parameter List..... 449

37. APPCVM RECEIVE Input Parameter List..... 453

38. APPCVM RECEIVE Output Parameter List (Function Complete Interrupt)..... 456

39. APPCVM SENDCNF Input Parameter List.....466

40. APPCVM SENDCNF Output Parameter List (Function Complete Interrupt).....468

41. APPCVM SENDCNFD Input Parameter List..... 472

42. APPCVM SENDCNFD Output Parameter List (Function Complete Interrupt)..... 473

43. APPCVM SENDDATA Input Parameter List..... 477

44. APPC Logical Record Format..... 478

45. APPCVM SENDDATA Output Parameter List (Function Complete Interrupt)..... 482

46. Message Pending External Interrupt.....489

47. APPCVM SENDERR Input Parameter List.....492

48. Error Log GDS Variable Format....494
49. APPCVM SENDERR Output Parameter List (Function Complete Interrupt)....496
50. APPCVM SENDREQ Input Parameter List....502
51. APPCVM SENDREQ Output Parameter List....503
52. SENDREQ (Request-to-Send) Interrupt....504
53. APPCVM SETMODFY Input Parameter List....506
54. APPCVM SETMODFY Output Parameter List....507
55. APPCVM SEVER Input Parameter List....511
56. Error Log GDS Variable Format....513
57. APPCVM SEVER Output Parameter List (Function Complete Interrupt)....516
58. SEVER External Interrupt....519
59. IUCV ACCEPT Input Parameter List....526
60. IUCV ACCEPT Output Parameter List....527
61. User Data Field for CONNECT....530
62. IUCV CONNECT Input Parameter List....531
63. IUCV DCLBFR Input Parameter List....535
64. IUCV DESCRIBE Output Parameter List....539
65. IUCV IPOLL Output Parameter List....541
66. IUCV QUERY Input Parameter List....544
67. IUCV QUERY Output Parameter List (QRYTYPE=BUFFERS)....545
68. IUCV QUERY Output Parameter List (QRYTYPE=CONNECT)....545
69. IUCV SETCMASK Input Parameter List....551
70. IUCV SETMASK Input Parameter List....554
71. IUCV SEVER Input Parameter List....557
72. IUCV SEVER Output Parameter List (Sever Complete Interrupt)....558

73. IUCV SEVER External Interrupt....560
74. IUCV TESTCMPL Input Parameter List....563
75. IUCV TESTCMPL Output Parameter List....563
76. Overview of the CP Access Control Interface to an ESM....592
77. Interface Specifications for the HCPRPIRA Entry Point....595
78. Interface Specifications for the HCPRPWEP Entry Point....597
79. Interface Specifications for the HCPRPEPX Entry Point....604
80. Interface Specifications for the HCPRPESG Entry Point (Part 1 of 2)....605
81. Interface Specifications for the HCPRPESG Entry Point (Part 2 of 2)....606
82. Interface Specifications for the HCPPWAPF Entry Point....608
83. Interface Specifications for the HCPDA0RL Entry Point (Part 1 of 2)....613
84. Interface Specifications for the HCPDA0RL Entry Point (Part 2 of 2)....614
85. Interface Specifications for the HCPDA0UL Entry Point (Part 1 of 2)....615
86. Interface Specifications for the HCPDA0UL Entry Point (Part 2 of 2)....616
87. Interface Specifications for the HCPDA0MC Entry Point....617
88. SPGBK DSECT....767
89. SPRBK DSECT....767
90. Created Data Spaces Being Shared With Other Users....783
91. Granting Another User Access to an Address Space....787
92. Graphic Representation of an ALET and an ALE....788
93. Accessing an Address Space....789
94. Using DEFWORKA within a Reentrant Program....794
95. Using DEFWORKA within a Non-reentrant Program....795
96. Using DEFWORKA to Force Unique Macro Work Areas....796
97. Using a Remote Macro Work Area....797

98. Page Fault Processing when VERSION=2 861

99. Group of 1 Subgroup That Has 4 Spans of Pages 870

100. Group of 2 Subgroups That Have 2 Spans of Pages in Each Subgroup 870

101. Group of 2 Subgroups That Have 2 Spans of Pages in Each Subgroup 870

102. Group of a Span of 18 Pages in One Logical Block 871

103. Group of a Span of 18 Pages in One Logical Block 871

104. Contents of the LSTMDISK Function Parameter List 876

105. Summary of permissible collaborative memory management state combinations 885

106. Suggested Format of an External Attribute Buffer 979

107. The SEND Protocol 993

108. The SEND/RECV Protocol 994

109. The SENDX Protocol 995

110. The IDENTIFY Protocol 995

111. DIAGNOSE X'2C4' FPL Parameter List Format 1081

Tables

1. Examples of Syntax Diagram Conventions..... xxxv

2. Summary of Storage Protection Mechanisms..... 10

3. DIAGNOSE code X'00' — Bit Map Fields..... 13

4. Normal Exit Results with the SEGEXT Function..... 77

5. Results of Exit with Error from DIAGNOSE code X'64'..... 78

6. DIAGNOSE Code X'84' Operations..... 93

7. DIAGNOSE Code X'94' Condition Codes..... 120

8. DIAGNOSE Code X'94' Return Codes..... 121

9. DIAGNOSE X'94' Symptom Record Processing Return Codes..... 123

10. DIAGNOSE X'94' Reason Codes for Return Code X'00'..... 123

11. DIAGNOSE X'94' Reason Codes for Return Code X'04'..... 123

12. DIAGNOSE X'94' Reason Codes for Return Code X'08'..... 123

13. DIAGNOSE X'94' Reason Codes for Return Code X'12'..... 123

14. DIAGNOSE X'A0' Program Checks..... 135

15. DIAGNOSE Code X'A4' Return Codes..... 142

16. DIAGNOSE Code X'A4' Return Codes..... 143

17. DIAGNOSE Code X'A4' Return Codes in the Guest's Register 15 with CC=3..... 143

18. DIAGNOSE Code X'A8' Return Codes in the Guest's Register 15 with CC=1..... 148

19. DIAGNOSE Code X'B4' Return Codes..... 152

20. Return Codes..... 154

21. Fields in the VRDCBLOK DSECT..... 192

22. Summary of the Effects of Byte X'19' Bits 6 and 7 on Read/Write Processing..... 208

23. Status codes for block I/O entries..... 210

24. DIAGNOSE X'250' condition codes 213

25. Condition codes and return codes for the Initialize function 213

26. Condition codes and return codes for the Read/Write function 213

27. Condition codes and return codes for the Remove function 214

28. Program exceptions 215

29. Version definitions 227

30. Subcode X'00000004' Return Codes 228

31. Return Virtual LAN System Information (DSECT CSISRESP) 229

32. Return IVL Membership Information (DSECT CSISISTR) 230

33. Subcode X'00000008' Return Codes 231

34. Subcode X'0000000C' Return Codes 232

35. Return Controller Information (DSECT CSICRESP) 232

36. Controller Information (DSECT CSICCSTR) 233

37. Vswitch Information (DSECT CSICVSTR) 234

38. Subcode X'00000010' Return Codes 235

39. Subcode X'00000014' Return Codes 235

40. Return Guest LAN Information (DSECT CSIGRESP) 236

41. Guest LAN Information (DSECT CSIGGSTR) 236

42. Connected Adapter Information (DSECT CSIGASTR) 238

43. Authorized User Information (DSECT CSIGUSTR) 238

44. Subcode X'00000018' Return Codes 238

45. Subcode X'0000001C' Return Codes 239

46. Return Virtual Switch Information (DSECT CSIVRESP) 240

47. Virtual Switch Information (DSECT CSIVVSTR) 240

48. RDEV Information (DSECT CSIVRSTR) 246

49. Segment Information (DSECT CSIVSSTR)..... 250
50. Take-Over MAC Address Information (DSECT CSIVTSTR)..... 251
51. Connected Adapter Information (DSECT CSIVASTR)..... 251
52. Authorized User Information (DSECT CSIVUSTR)..... 252
53. Authorized Port Information (DSECT CSIVPSTR)..... 252
54. Authorized User VLAN Information (DSECT CSIVLSTR)..... 253
55. Global Virtual Switch Member Information (DSECT CSIVMSTR)..... 253
56. Subcode X'00000020' Return Codes..... 253
57. Return Virtual Port or Virtual NIC Information (DSECT CSIPRESP)..... 255
58. Port or NIC Information (DSECT CSIPNSTR)..... 255
59. Device Information (DSECT CSIPDSTR) 257
60. Data Device Details (DSECT CSIPTSTR)..... 258
61. Active Segment Information (DSECT CSIPSSTR)..... 258
62. MAC Address Information (DSECT CSIPMSTR)..... 259
63. Subcode X'00000024' Return Codes..... 260
64. MAC Services Return MAC Address (DSECT CSIMRESP)..... 261
65. Subcode X'00000030' Return Codes..... 262
66. Sequence of Functions..... 302
67. CP System Services and Their User IDs..... 318
68. Applicable Codes Based on the Condition Code..... 396
69. Possible APPC/VM Sever Codes..... 399
70. Sever Codes Generated by VM..... 400
71. APPC/VM States..... 404
72. APPC/VM States for Coordinated Resource Recovery..... 405
73. Error Conditions..... 408

74. Summary of Locally Known LU Names....420
75. APPC/VM-Defined SENDERR Codes....494
76. APPC Conversation States and Corresponding APPC/VM Implementation.... 574
77. Base Set of APPC Verbs and APPC/VM Functions.... 575
78. APPC Operator Control Verbs Mapped to AVS Commands.... 576
79. Supported HCPRPD Return Codes.... 602
80. Format of information in the HCPA0LBK control block.... 617
81. Generic command audit format of ACIPARMS....639
82. Supported Return Codes.... 640
83. Generic DIAGNOSE Call Format of the ACIPARMS Parameter List....640
84. Supported Return Codes.... 640
85. ACILVIDL and Buffer Data Examples for VLAN AWARE Virtual Switches.... 641
86. ACILVIDL and Buffer Data Examples for Guest LANs and VLAN UNAWARE virtual switches....641
87. (X)AUTOLOG Command Format of the ACIPARMS Parameter List....642
88. (X)AUTOLOG ESM output fields in ACIPARMS....643
89. Supported HCPRPWEP Return Codes for the AUTOLOG and XAUTOLOG.... 643
90. CHANGE Command Audit Format of the ACIPARMS Parameter List....644
91. Supported HCPRPIRA Return Codes....644
92. CHANGE Command Format of the ACIPARMS Parameter List....645
93. Supported HCPRPIRA Return Codes....645
94. CLOSE TO Command Format of the ACIPARMS Parameter List....645
95. Supported HCPRPIRA Return Codes for the CLOSE TO Command....646
96. COUPLE Command Format of the ACIPARMS Parameter List....646
97. Supported HCPRPIRA Return Codes for the COUPLE Command....647
98. COUPLEN Command Format of the ACIPARMS Parameter List....647

99. Supported HCPRPIRA Return Codes for the COUPLEN Command..... 647

100. FOR Command MAC/AUDIT Format of the ACIPARMS Parameter List..... 648

101. Supported Return Codes for the FOR Command..... 649

102. GIVE return..... 649

103. LINK Command Format of the ACIPARMS Parameter List..... 650

104. Supported HCPRPIRA Return Codes for the LINK Command..... 650

105. LOGOFF Command Format of the ACIPARMS Parameter List..... 651

106. AT Command Guest LOGOFF Format of the ACIPARMS Parameter List..... 651

107. LOGON Command Format of the ACIPARMS Parameter List..... 652

108. LOGON ESM output fields in ACIPARMS..... 654

109. Supported HCPRPWEP Return Codes for the LOGON Command..... 655

110. AT Command Guest LOGON Format of the ACIPARMS Parameter List..... 655

111. MESSAGE Command MAC/AUDIT Format of the ACIPARMS Parameter List..... 656

112. Supported Return Codes for the MESSAGE Command..... 656

113. Supported HCPRPIRA Return Codes for the PURGE Command..... 657

114. QUERY TAG FILE Command Format of the ACIPARMS Parameter List..... 657

115. Supported Return Codes..... 658

116. QUERY RDR/PRT/PUN Command Format of the ACIPARMS Parameter List..... 658

117. Supported HCPRPIRA Return Codes..... 658

118. SEND Command Audit Call Format of the ACIPARMS Parameter List..... 659

119. SEND Command Security Label MAC Check Format of the ACIPARMS Parameter List..... 659

120. Supported Return Codes..... 660

121. SPOOL Command Format of the ACIPARMS Parameter List..... 660

122. Supported HCPRPIRA Return Codes for the SPOOL TO Command..... 660

123. ACIPARMS format..... 661

124. Supported Return Codes..... 661
125. START real printer with SECLABEL option authorization call.....661
126. Supported Return Codes for the START Real Printer with SECLABEL..... 662
127. STORE HOST Command Format of the ACIPARMS Parameter List..... 662
128. Supported Return Codes..... 663
129. TAG Command Format of the ACIPARMS Parameter List.....663
130. Supported Return Codes for the TAG Command.....664
131. TRANSFER Command Format of the ACIPARMS Parameter List.....664
132. Supported Return Codes for the TRANSFER Command.....664
133. TRANSFER Command Format of the ACIPARMS Parameter List.....665
134. Supported HCPRPIRA Return Codes..... 665
135. TRSAVE TO Command Format of the ACIPARMS Parameter List..... 666
136. Supported HCPRPIRA Return Codes for the TRSAVE TO Command..... 666
137. TRSOURCE Command Format of the ACIPARMS Parameter List..... 666
138. Supported HCPRPIRA Return Codes for the TRSOURCE Command..... 667
139. TRSOURCE ENABLE Command Format of the ACIPARMS Parameter List..... 667
140. Supported Return Codes..... 668
141. VMRELOCATE Command Format of the ACIPARMS Parameter List.....668
142. VMDUMP TO Command and DIAGNOSE X'94' Format of the ACIPARMS Parameter List..... 669
143. Supported Return Codes..... 669
144. DIAGNOSE Code X'14' Format of the ACIPARMS Parameter List.....670
145. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'14'.....670
146. DIAGNOSE Code X'64' Format of the ACIPARMS Parameter List.....670
147. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'64'.....671
148. DIAGNOSE Code X'68' Format of the ACIPARMS Parameter List.....671

149. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'68'.....672
150. DIAGNOSE Code X'88' Format of the ACIPARMS Parameter List.....672
151. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'88'.....672
152. DIAGNOSE Code X'B8' Format of the ACIPARMS Parameter List.....673
153. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'B8'.....673
154. DIAGNOSE Code X'BC' Format of the ACIPARMS Parameter List.....673
155. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'BC'.....674
156. DIAGNOSE Code X'D4' Format of the ACIPARMS Parameter List.....674
157. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'D4'.....675
158. DIAGNOSE Code X'E4' Command Format of the ACIPARMS Parameter List.....675
159. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'E4'.....676
160. DIAGNOSE Code X'290' Command Format of the ACIPARMS Parameter List.....676
161. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'290'.....677
162. DIAGNOSE Code X'23C' Format of the ACIPARMS Parameter List.....677
163. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'23C'.....677
164. APPC CONNECT Format of the ACIPARMS Parameter List.....678
165. Supported HCPRPIRA Return Codes for APPC CONNECT.....678
166. APPC setting of VMDALTID format of ACIPARMS.....679
167. Supported Return Codes.....679
168. APPC SEVER Format of the ACIPARMS Parameter List.....680
169. APPC connect with password validation.....680
170. Supported Return Codes.....681
171. Directory Command Format of the ACIPARMS Parameter List.....681
172. Supported Return Codes for a Directory Command.....681
173. IUCV CONNECT Format of the ACIPARMS Parameter List.....682

174. Supported HCPRPIRA Return Codes for IUCV CONNECT..... 682
175. IUCV SEVER Format of the ACIPARMS Parameter List..... 682
176. MAINTCCW Format of ACIPARMS.....683
177. Supported HCPRPIRA Return Codes..... 683
178. MDISK Command Format of the ACIPARMS Parameter List.....684
179. Supported HCPRPIRA Return Codes for MDISK.....684
180. POSIX Set ID Format of the ACIPARMS Parameter List..... 685
181. POSIX Set IDs ESM output fields in ACIPARMS..... 686
182. Supported HCPRPIRA Return Codes for POSIX Set ID Functions..... 686
183. Query POSIX Group Database Format of the ACIPARMS Parameter List.....686
184. Query POSIX group database output fields in ACIPARMS..... 687
185. Supported HCPRPIRA Return Codes for POSIX group database query.....688
186. User Database Query Format of the ACIPARMS Parameter List.....688
187. Query POSIX user database output fields in ACIPARMS.....690
188. Supported HCPRPIRA Return Codes for user database query..... 690
189. Resource Access Authorization Check Format of the ACIPARMS Parameter List.....691
190. Supported HCPRPIRA Return Codes for Resource Access Authorization Check.....692
191. Class and Resource Names Used by CP for Resource Access Authorization Check.....692
192. RSTDSEG format of the ACIPARMS Parameter List.....693
193. Supported HCPRPIRA Return Codes for RSDTSEG..... 693
194. SCIF Event Audit Format of the ACIPARMS Parameter List.....694
195. SCIF Event MAC Check Format of the ACIPARMS Parameter List..... 694
196. Supported Return Codes..... 694
197. SPFOPEN Format of the ACIPARMS Parameter List.....695
198. Supported HCPRPIRA Return Codes..... 695

199. ACIPARMS Parameter List....696
200. ACIPARMS Parameter List....696
201. Supported HCPRPIRA Return Codes for a Promiscuous Mode Audit....697
202. PRINT Format of the ACIPARMS Parameter List....697
203. Supported HCPRPIRA Return Codes for PRINT....697
204. CP SET Commands with an IUCV Option....739
205. Converting an EC-mode PSW to a mapped PSW....910
206. Constructing a mapped PSW....911
207. VMCF Return Codes from DIAGNOSE code X'68'....988
208. Virtual Machine Communication Facility (VMCF) Functions....989
209. VMCF Function Codes for DIAGNOSE Code X'68'....1001
210. Required VMCPARM Fields for VMCF Functions....1003
211. VMCMFUNC Subcodes - DIAGNOSE Code X'68'....1005
212. DIAGNOSE Code X‘68’ Return Codes....1007
213. DIAGNOSE Code X‘68’ Data Transfer Error Codes....1009
214. Summary of Logical Device Support Facility Functions....1013
215. Hardware Flags....1034
216. Hardware Flags....1035
217. Hardware Flags....1036
218. I/O Request Flags....1037
219. General condition code descriptions for all functions....1038
220. Condition codes and return codes for the Open CP Connection function....1038
221. Condition codes and return codes for the Close CP Connection function....1039
222. Condition codes and return codes for the Perform I/O function....1039
223. Perform I/O function completion status codes....1040

224. DIAGNOSE X'25C' Function List....1042

225. DIAGNOSE X'25C'—Condition codes....1044

226. Diagnose X'25C' - Return codes....1044

227. CP Subfunctions....1045

228. DIAGNOSE X'280' condition codes....1052

229. Condition codes and return codes for changing effective and saved set-IDs....1052

230. Condition codes and return codes for changing saved set-IDs only....1054

231. Condition Codes for Query I/O Configuration Information (Function 0)....1077

232. Condition Codes and Return Codes for Perform Dynamic I/O Changes (Function 1)....1077

233. DIAGNOSE Code X'2C4' Return Codes in Ry....1082

About This Document

This document contains reference information pertaining to specific services and facilities of CP, such as the DIAGNOSE instruction, IUCV, APPC/VM, and VM data spaces.

Intended Audience

This information is intended for systems programmers and applications programmers who will be writing programs for IBM® z/VM®.

To get the most out of this information, you should have a general idea of what z/VM does and what a virtual machine is. You should also have a working knowledge of Basic Assembler Language programming.

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 xxxv.

Table 1. Examples of Syntax Diagram Conventions

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

© Copyright IBM Corp. 1991, 2020

Table 1. Examples of Syntax Diagram Conventions (continued)

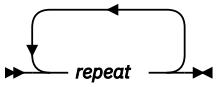
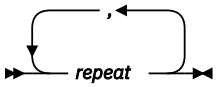
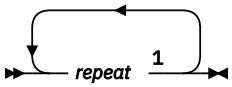
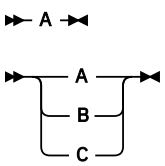
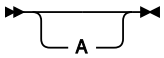
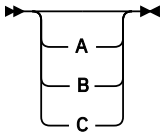
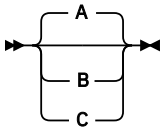
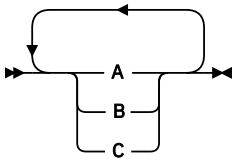
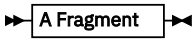
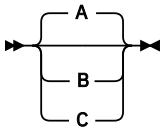
Syntax Diagram Convention	Example
Abbreviations Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated. In this example, you can specify KEYWO, KEYWOR, or KEYWORD.	
Symbols You must specify these symbols exactly as they appear in the syntax diagram.	***** Asterisk **:** Colon **,** Comma **=** Equal Sign **-** Hyphen **()** Parentheses **.** Period
Variables A variable appears in highlighted lowercase, usually italics. In this example, *var_name* represents a variable that you must specify following KEYWORD.	
Repetitions An arrow returning to the left means that the item can be repeated. A character within the arrow means that you must separate each repetition of the item with that character. 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. Syntax notes may also be used to explain other special aspects of the syntax.	   Notes: [1] Specify *repeat* up to 5 times.
Required Item or Choice When an item is on the line, it is required. In this example, you must specify A. 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.	

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	**Example**
Optional Item or Choice When an item is below the line, it is optional. In this example, you can choose A or nothing at all. 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.	 
Defaults 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. In this example, A is the default. You can override A by choosing B or C.	
Repeatable Choice 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. In this example, you can choose any combination of A, B, or C.	
Syntax Fragment 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."	 **A Fragment** 

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 z/VM functions, see the other documents listed in the “Bibliography” on page 1089.

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 Send Your Comments to IBM

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

To send us your comments, go to z/VM Reader's Comment Form (www.ibm.com/systems/campaignmail/z/zvm/zvm-comments) and complete the form.

If You Have a Technical Problem

Do not use the feedback method. Instead, do one of the following:

- Contact your IBM® service representative.
- Contact IBM technical support.
- See IBM: z/VM Support Resources (www.ibm.com/vm/service).
- Go to IBM Support Portal (www.ibm.com/support/entry/portal/Overview).

© Copyright IBM Corp. 1991, 2020

Summary of Changes for z/VM CP Programming Services

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.

SC24-6272-04, z/VM 7.2 (September 2020)

This edition includes changes to support the general availability of z/VM 7.2.

Accounting Record Formats Information Moved

Information about Accounting Record Formats has been moved from *z/VM: CP Planning and Administration* to Chapter 12, “Account System Service (*ACCOUNT),” on page 699.

SC24-6272-03, z/VM 7.1 (January 2020)

This edition includes changes to support product changes provided or announced after the general availability of z/VM® 7.1.

[VM66329] STHYI support for zCX containers

With the PTF for APAR VM66329, z/VM 7.1 provides enhancements to the STHYI instruction to report zIIP processor type configuration information in support of z/OS zCX containers.

The following topics are updated:

- Chapter 29, “Store Hypervisor Information (STHYI) Instruction,” on page 913
- “Function Code X'0000' - Processor Capacity Information” on page 914
- “Function Code X'0000' Response Machine Section (INF0MAC DSECT)” on page 917
- “Function Code X'0000' Response Partition Section (INF0PAR DSECT)” on page 918
- “Function Code X'0000' Response Hypervisor Section (INF0HYP DSECT)” on page 921
- “Function Code X'0000' Response Guest Section (INF0GST DSECT)” on page 923
- “Special Conditions, Exceptions, and Usage Notes” on page 957

SC24-6272-02, z/VM 7.1 (June 2019)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 7.1.

SC24-6272-01, z/VM 7.1 (April 2019)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 7.1.

[VM66219] Virtual Switch Priority Queuing Enhancement

With the PTF for APAR VM66219, z/VM support is enhanced to improve data transmission from a virtual NIC to a physical network via the virtual switch uplink port. Internal virtual switch data transmission is changed from a first-in, first-out (FIFO) order to a four-level priority queuing scheme. By default, the

© Copyright IBM Corp. 1991, 2020

virtual switch distinguishes between virtual switch and virtual NIC transmissions to the physical network. Virtual switch transmissions required to manage an uplink port or virtual NIC are sent to the physical network ahead of any virtual NIC transmissions. This insures that virtual switch management is always maintained during high bandwidth uplink port usage.

Virtual switch support is added to exploit the OSA-Express feature's priority queuing capabilities. By turning on virtual switch priority queuing, instead of activating a single QDIO output queue on the uplink port, the z/VM virtual switch will establish four QDIO output queues for outbound data transmissions. The four QDIO output queues provide four independent interfaces to the OSA-Express feature, one for each priority. The highest priority queue (Queue 0) is used by the z/VM virtual switch to provide switch management communications and the other three priorities (Queues 1-3) can be configured by a system administrator to prioritize outbound transmissions to the physical network for virtual NICs connected to the virtual switch.

The following DIAGNOSE code has been updated:

- "DIAGNOSE Code X'26C' – Access Certain System Information" on page 226

SC24-6272-00, z/VM 7.1 (September 2018)

This edition includes changes to support the general availability of z/VM 7.1.

Foundational Support for More Than 64 Logical Processors

Foundational support is provided for a follow-on V7.1 deliverable that will increase the number of supported logical processors beyond the current limit of 64. This will allow clients to run a z/VM LPAR with more than 64 cores in an SMT1 environment or more than 32 cores in an SMT2 environment to accommodate workload growth demands.

Support for dedicating processors to guests is disabled.

The following sections are updated:

- "Usage Notes" on page 81
- "Responses" on page 292
- "Function Code X'0000' - Processor Capacity Information" on page 914

QUERY BYUSER Support for Class B Users

This support provides users with privilege class B the ability to issue the QUERY BYUSER command for other users, similarly to the function granted by privilege class E.

The following DIAGNOSE code has been updated:

- "DIAGNOSE Code X'26C' – Access Certain System Information" on page 226

SC24-6179-13, z/VM 6.4 (March 2018)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 6.4.

Enhanced Support for External Management of z/VM CPU Resources

With the PTF for APAR VM66105, z/VM enhances the capability for external performance management tools to control access to CPU resources to achieve workload goals.

- The Store Hypervisor Information (STHYI) instruction is updated with six new function codes (1 - 6) for obtaining hypervisor, guest, and resource pool information.

- The hypervisor section of the STHYI function code 0 response buffer is enhanced to include two new masks to indicate which function codes are supported by the hypervisor and which of those function codes the guest is authorized to use.
- Two new VM Event System Service (*VMEVENT) event types are added.

See Chapter 29, “Store Hypervisor Information (STHYI) Instruction,” on page 913 and “Receiving Events” on page 775.

SC24-6179-12, z/VM 6.4 (December 2017)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 6.4.

VSwitch Link Aggregation Load Balancing Enhancements

With the PTF for APARs VM65918, z/VM support for exclusive and Multi-VSwitch Link Aggregation configurations is enhanced to improve load balancing to leverage both horizontal and vertical growth in single and cross virtual switch networking configurations.

The following DIAGNOSE code has been updated:

- “DIAGNOSE Code X'26C' – Access Certain System Information” on page 226

SC24-6179-11, z/VM 6.4 (August 2017)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 6.4.

z/VM Support for the IBM z14

With the PTF for APAR VM65942, z/VM V6.4 provides support that enables guests to exploit function supported by z/VM on the IBM z14™ (z14™).

IBM z14 does not support the full ESA/390 architectural mode. However, z14 does provide ESA/390-compatibility mode, an environment supporting a subset of DAT-off ESA/390 applications in a hybrid architectural mode. z/VM provides the support necessary for DAT-off guests to run in this new compatibility mode, which allows guests such as CMS, GCS, and those that start in ESA/390 mode briefly before switching to z/Architecture® mode to continue to run on z14.

Extended Address Volume Minidisk Support

With the PTF for APAR VM65943, for 3390-A DASD devices that support 28-bit cylinder addresses, a non-fullpack minidisk can reside anywhere on the volume, including beyond the 64 KB boundary (0-65519), and up to the 1 TB limit. The following DIAGNOSE codes and functions have been updated:

- DIAGNOSE code X'18'
- DIAGNOSE code X'20'
- DIAGNOSE code X'A4'
- DIAGNOSE code X'E4'
- *BLOCKIO
- ACIPARMS
- MAPMDISK IDENTIFY

SC24-6179-10, z/VM 6.4 (March 2017)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 6.4.

Distributed IUCV Enhancements

With the PTF for APAR VM65872, the rules for Distributed IUCV CONNECT in a single system image (SSI) environment are revised. This support allows IUCV CONNECT to work in cases that were originally restricted, primarily because they involved a multiconfiguration virtual machine (IDENTITY) user.

This support also makes it easier for an administrator to change the Distributed IUCV policy for an SSI cluster. Previously, the Distributed IUCV policy within an active SSI cluster could be changed only by shutting down all members at the same time. As this new support is applied to each system, it will be possible for that member to join the cluster regardless of its Distributed IUCV configuration.

NICDEF Security Controls

With the PTFs for APARs VM65925, VM65926 and VM65931, the NICDEF user directory statement is enhanced to provide a set of new operands referred to as Directory Network Authorization (DNA). With DNA, a system administrator can configure and consolidate a virtual NIC device and its network properties in a centralized location - z/VM's User Directory.

Operational differences between PORTBASED and USERBASED VSwitches have been eliminated with this support. A system administrator has the option to manage a VSwitch by user, by port number or using a combination of the two methods. While the management of USERBASED and PORTBASED VSWITCHes is simplified, Live Guest Relocation of a guest connected to a VSwitch still requires the destination system to have a VSwitch with a PORTBASED or USERBASED designation matching that of the source system.

The following DIAGNOSE code has been updated:

- "DIAGNOSE Code X'26C' – Access Certain System Information" on page 226

VMUDQ Macro Identifies SUBCONFIGs That Contain MDISKs

With the PTF for APAR VM65877, the VMUDQ macro is enhanced to support identifying the IDs of SUBCONFIG stanzas that contain MDISK definitions in an SSI-enabled or SSI-ready directory.

SC24-6179-09, z/VM 6.4 (November 2016)

This edition includes changes to support the general availability of z/VM 6.4.

ESA/390 Removal

z/VM enhancements enable hypervisor initialization and termination, the Stand-Alone Program Loader (SAPL), DASD Dump Restore (DDR), and other stand-alone utilities to run entirely in z/Architecture mode. The IBM z13® and z13s® are planned to be the last IBM Z(R) servers to support running an operating system in ESA/390 architecture mode. All future systems will support only operating systems that run entirely in z/Architecture mode. In addition, support has been added to z/VM to simulate a z/Architecture-only environment, by providing a virtual machine environment (MACHINE type Z) that is always in z/Architecture mode and cannot switch to ESA/390 mode.

Updates in this publication include:

- "DIAGNOSE Code X'84' – Directory Update-in-Place" on page 91
- Minor changes throughout the publication to document new machine mode Z

zManager Support Removed

The IBM z Unified Resource Manager (zManager) is no longer supported in z/VM 6.3. The vswitch types of IEDN and INMN have been removed from documentation.

The following DIAGNOSE code has been updated:

- "DIAGNOSE Code X'26C' – Access Certain System Information" on page 226

Guest Transactional Execution Support

z/VM now supports guest exploitation of the Transactional Execution facility on supported hardware (machines that provide this facility). The Transactional Execution facility allows a program to issue multiple instructions, and can improve the efficiency and scalability of multithreaded software.

The following sections have been updated:

- "Instruction Format" on page 3
- "CP System Services" on page 317
- Chapter 29, "Store Hypervisor Information (STHYI) Instruction," on page 913.

Expanded Storage (XSTORE) Support Removed

z/VM V6.4 does not support expanded storage (XSTORE) for either host or guest use.

The following DIAGNOSE code has been updated:

- "DIAGNOSE Code X'84' – Directory Update-in-Place" on page 91

Part 1. CP DIAGNOSE Instructions

The capability of adding customer-written CP routines to the system allows you to define new DIAGNOSE codes or replace existing DIAGNOSE codes by redefining them, and to enable or disable either new or existing DIAGNOSE codes. See *z/VM: CP Exit Customization* for additional information.

This part contains the following chapters:

- Chapter 1, “The DIAGNOSE Instruction in a Virtual Machine,” on page 3, which tells you how to use the DIAGNOSE instruction and the different types of DIAGNOSE instructions.
- Chapter 2, “The IBM-Supplied DIAGNOSE Codes,” on page 13, which provides a detailed description of each IBM-Supplied DIAGNOSE code.

© Copyright IBM Corp. 1991, 2020

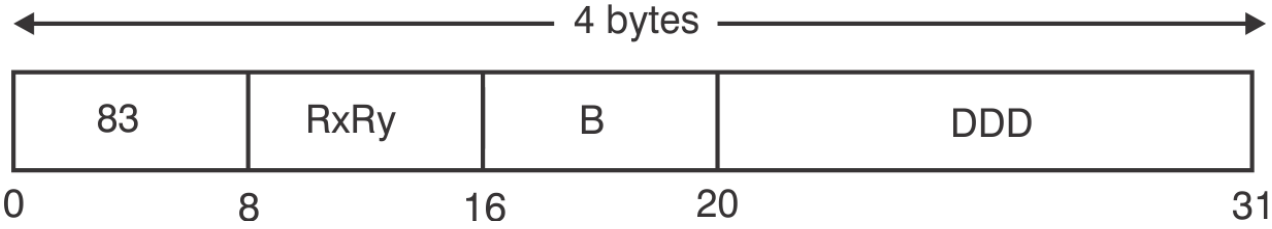
Chapter 1. The DIAGNOSE Instruction in a Virtual Machine

In a real processor, the DIAGNOSE instruction performs processor-dependent diagnostic functions. In a virtual machine, you use the DIAGNOSE interface to request that CP perform services for your virtual machine. When your virtual machine attempts to execute a DIAGNOSE instruction, control is returned to CP. CP uses information provided in the *code* portion of the instruction to determine what service it should perform. Once this service is provided, control returns to the virtual machine.

Thus, because a DIAGNOSE instruction issued in a virtual machine results only in returning control to CP and not in performing real-processor DIAGNOSE functions, think of DIAGNOSE codes as a method of communicating virtual machine requirements to CP.

Instruction Format

The machine language format of a DIAGNOSE instruction is:



83
is X'83', the machine language operation code for the DIAGNOSE instruction.

Note: There is no assembler mnemonic for DIAGNOSE.

RxRy
typically designate the general registers that contain the operand values or operand storage addresses to be passed through the DIAGNOSE interface.

The use of the Rx and Ry fields varies by DIAGNOSE code and subcode. In some cases, Rx and/or Ry may designate a pair of consecutive registers, referred to as registers Rx and Rx+1 and/or Ry and Ry+1.

B
is the base register. IBM recommends that you specify Register 0 as the base register (see note “1” on page 3 following).

DDD
is either all or part of the actual DIAGNOSE code value, in hexadecimal (see note “1” on page 3 following). This value is also called the *displacement*. IBM does not use codes X'100' through X'1FC'.

Notes:

1. With the DIAGNOSE instruction, the effective address (contents of the base register plus the displacement) is **not** used to address the data. Ordinarily, the contents of the base register plus the displacement equal the DIAGNOSE code. However, if you specify the base register to be register 0, its contents are not added to the displacement; the displacement alone determines the DIAGNOSE code. IBM recommends that you always specify the base register as register 0; therefore, you should always set bits 16 through 19 to 0. The DIAGNOSE code, or displacement, must always be a multiple of four.
2. If Rx, Ry or, if applicable, Rx+1 or Ry+1 contains an address, it must be a second-level address (that is, an address in the storage that appears real to the issuing virtual machine), unless otherwise specified.
3. The description of each DIAGNOSE instruction includes an **Addressing Mode** specification which indicates whether it supports 24–bit, 31–bit or 64–bit addressing modes.

© Copyright IBM Corp. 1991, 2020

4. Because DIAGNOSE instructions execute differently in a virtual machine than in a real machine, your program should determine that it is operating in a virtual machine before issuing a DIAGNOSE instruction, and prevent execution of a DIAGNOSE instruction when in a real machine. If the first byte (version code) stored by the Store CPU ID instruction is X'FF', then the program is running in a virtual machine.
5. A virtual machine issuing an I/O DIAGNOSE instruction should run with interrupts disabled. This prevents the loss of information pertaining to the DIAGNOSE operation such as status conditions and sense data.
6. Use of the DIAGNOSE instruction within a transaction will cause the transaction to abort with either a restricted-instruction transaction-abort code or a transaction-constraint exception. (See z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf) for details on transactional execution.)

Example of the DIAGNOSE Instruction: The example in Figure 1 on page 4 demonstrates how to code a DIAGNOSE code in a small Assembler program. This example happens to use DIAGNOSE code X'00', which stores the z/VM extended identification code at a specified location in guest storage.

```
DIAG00   CSECT
*This demonstrates the use of DIAGNOSE code X'00', which
*stores the z/VM Identification Code at a specified address.
         BALR 12,0
         USING *,12
         LA   2,LOC                    STORAGE ADDRESS IN R2 (Rx)
         LA   3,L'LOC                  # BYTES TO BE STORED IN R3 (Ry)
         DC   X'83',X'23',XL2'0000'    PIECE TOGETHER THE DIAG. CODE
         BR   14
         DS   0D                       FORCE DOUBLEWORD BOUNDARY
LOC      DS   XL64                     STORAGE LOCATION
         END
```

Figure 1. Example of a DIAGNOSE Code in an Assembler Program

The DC (Declare Constant) line in this program is the actual DIAGNOSE code.

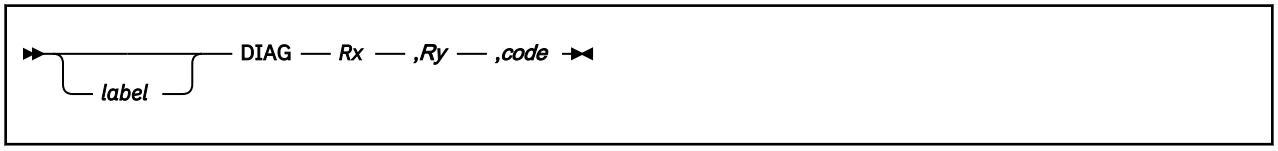
X'83'
is the machine language operation code for the DIAGNOSE instruction.

X'23'
is the specification of the RxRy general register pair. Here, Rx is 2 and Ry is 3.

XL2'0000'
is the base register and the displacement. The base register (0) is the first half byte. The remaining bits are the displacement, which (for the DIAGNOSE instruction) is the 3-digit hexadecimal DIAGNOSE code (000).

Macro Format

Using the DIAG macro automatically generates the correct machine format of the DIAGNOSE instruction. The macro format for the DIAG macro is:



DIAG
Requests that CP perform the DIAGNOSE instruction.

Rx
is the general register that contains the Rx operand value or operand storage address required for the particular DIAGNOSE function.

Ry
is the general register that contains the Ry operand value or operand storage address required for the particular DIAGNOSE function.

code
is the actual DIAGNOSE value. This code can be specified as anything that is acceptable for a Y-type address constant. The constant must resolve to a supported DIAGNOSE code.

Example of the DIAG Macro:

```
DIAG   2,3,X'0000'
```

The above example generates the X'83' instruction for DIAGNOSE code X'00'.

Privilege Classes

Some DIAGNOSE codes, like some CP commands, are reserved for authorized users only. Each DIAGNOSE code description in this section tells you who may issue that particular code. For more information on the letter values used in this entry, see *z/VM: CP Commands and Utilities Reference*.

Address Translation Modes and Restrictions

Some DIAGNOSE codes can be issued only by XC virtual machines, and others can be issued by any type of virtual machine. Some DIAGNOSE codes cannot be issued by an XC virtual machine in access register mode, while some can be issued but only refer to the host-primary address space. Some DIAGNOSE codes can be issued only when 370 Accommodation is active. Some DIAGNOSE codes can use logical addresses, which refer to any guest address space, including the real, primary, secondary, or home space or any address space designated by an ALET in an access register. These capabilities and restrictions are noted in the description of the DIAGNOSE code.

Except for Diagnoses that accept logical addresses, XC virtual machines in primary space mode and non-XC virtual machines ignore the access registers; all addresses refer to the host-primary address space. XC virtual machines in access register mode ignore the contents of access register 0 when an address is passed in general register 0; the access register is assumed to contain an ALET X'00000000', which indicates the host-primary address space. All exceptions are noted in the description of the DIAGNOSE code.

A DIAGNOSE code issued by ESA, XA, and Z virtual machines may refer to a logical address. Such addresses are references to the address space identified by the translation and address-space control bits in the guest Program Status Word (PSW). This means that the guest can refer to an address that is in its real, primary, secondary, or home space, as well as one that is access-register designated. In access-register mode, the access register corresponding to the general register that contains a logical address is used to obtain the ALET for the address space, although if access register 0 is designated it is assumed to contain an ALET of X'00000000'.

The description of each DIAGNOSE instruction includes an **Addressing Mode** specification which indicates whether it supports 24–bit, 31–bit or 64–bit addressing modes.

How Addresses Are Processed

z/VM processes addresses according to the type of virtual machine in which a guest is running and the addressing mode which is set.

Type of Guest	Address Length	Address Space Containing Address
ESA and XA	24- or 31-bit, depending on the addressing mode[1]	host-primary[2]

Type of Guest	Address Length	Address Space Containing Address
ESA, XA, and XC	24- or 31-bit, depending on the addressing mode[1]	guest-real, guest-primary, guest-secondary, guest-home, or ALET-specified[4, 5]
XC, primary-space mode	24- or 31-bit, depending on the addressing mode[1]	host-primary
XC, access-register mode	24- or 31-bit, depending on the addressing mode[1]	host-primary or ALET-specified, depending on the DIAGNOSE function
ESA, XA, and Z in z/Architecture mode	24-, 31-, or 64-bit, depending on the addressing mode[1]	host-primary[2, 3]
ESA, XA, and Z in z/Architecture mode	24-, 31-, or 64-bit, depending on the addressing mode[1]	guest-real, guest-primary, guest-secondary, guest-home, or ALET-specified[5]

Notes:

1. Exception: I/O DIAGNOSE codes are always processed in 31-bit mode.
2. With the exception of address spaces accessed through DIAGNOSE code X'248', these types of virtual machines have access to a single host (CP) created absolute address space, in which all guest real and absolute addresses reside. This is the absolute address space created by CP at log on time to represent the virtual machine's main storage.
3. z/Architecture has 8-byte register values. Address values passed for DIAGNOSE instructions in 64-bit addressing mode use all 8 bytes of the register.
4. An XC virtual machine can operate only in primary-space mode or in access-register mode and thus may use logical addresses that are either guest-real or ALET-specified.
5. The DIAGNOSE must be defined as accepting a logical address, in which case the address space is selected based on the translation and address-space control bits in the guest PSW at the time the DIAGNOSE instruction is issued.

Unless otherwise specified, Most addresses passed in DIAGNOSE functions and CP programming interface macros are guest real addresses. Those functions that perform I/O operations or cross-virtual-machine communications generally take guest absolute addresses. In cases where the addressed area is referenced asynchronously (that is, during a process that continues after the DIAGNOSE instruction itself completes), addresses are also guest absolute. Consult the description of the individual DIAGNOSE function to determine whether its operand addresses are real or absolute.

Guest real addresses in the host-primary space are subject to prefixing, whereas guest absolute addresses are not. Addresses in spaces other than the host-primary space are never subject to prefixing. (In particular, references using an ALET other than X'00000000' are not subject to prefixing.) For more information on real and absolute addresses and prefixing, see the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf), *z/VM: ESA/XC Principles of Operation*, or z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), based on the architecture mode of the virtual machine.

Unless otherwise specified, all storage references by the DIAGNOSE and IUCV instructions and by CP programming interface macros are multiple-access references, as defined in the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf), *z/VM: ESA/XC Principles of Operation*, or z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), based on the architecture mode of the virtual machine.

How Address Spaces Are Selected

When a virtual machine logs on, a *host address space* is created to serve as the virtual machine's storage. This storage appears as absolute storage (as defined in the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf), *z/VM: ESA/XC Principles of Operation*, or z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), based on the architecture mode of the virtual machine) to the program running in the virtual machine. This address space is called the virtual machine's *host-primary* address space. Using VM data spaces, a virtual machine can also gain access to other host address spaces, which may be other virtual machines' host-primary spaces or data spaces created by this or other virtual machines.

XA, ESA, and Z virtual machines can reference these additional address spaces through DIAGNOSE code X'248' only. XC virtual machines can switch between two translation modes, called primary-space mode and access-register mode. Like a non-XC virtual machine, an XC machine in primary-space mode addresses only the host-primary space. An XC virtual machine in access-register mode can reference data in all accessible address spaces directly using nearly the full ESA/XC instruction set, including most of the DIAGNOSE functions described here.

XA, ESA, and Z machines can create their own address spaces and can switch among five translation modes called real-space mode, primary-space mode, secondary-space mode, home-space mode, and access-register mode. DIAGNOSE codes that accept logical addresses for parameters use the address space designated by the translation and address-space control bits in the guest PSW to determine which address space the logical address references.

For each storage address referenced, the descriptions of the individual DIAGNOSE codes specify whether the address is a host-primary address regardless of translation mode, a guest-logical address whose address space designation is determined by the guest PSW, or an AR-specified or ALET-specified address. An AR-specified address is one for which the address space is identified by an Access-List-Entry Token (ALET) in an access register. An ALET-specified address is one for which the address space is identified by an ALET in a parameter list. In either case, the ALET is translated by means of host access-register translation (host ART) to identify the target address space.

For more information on access registers, ALETs, and host ART, see Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf) and *z/VM: ESA/XC Principles of Operation*. For information on creating and gaining access to host address spaces see Chapter 25, “CP Macros,” on page 793.

How Error Conditions Are Reported

While trying to perform a DIAGNOSE function, CP may encounter errors. Most DIAGNOSE functions report these errors through program interruptions, as other instructions do. Some functions report errors through condition codes and return codes instead. The condition codes and return codes are unique to each function and are defined with the individual functions. The program interruptions are explained below.

The DIAGNOSE instruction in a virtual machine can give the following program interruptions:

Program Interruption	Cause
Operation exception Privileged-operation exception	For some DIAGNOSE functions, these exceptions indicate that the virtual machine is not authorized to perform the function.

Program Interruption	Cause
Specification exception	Any of the following: • The DIAGNOSE function code is not supported. • An unknown subfunction was requested. • For some DIAGNOSE functions, the virtual machine is not authorized to perform the function. • The DIAGNOSE function is not supported in the architecture mode of the virtual machine. • A virtual machine issued a DIAGNOSE function that does not support 64-bit addressing mode. • An incorrect register number was specified. • A field in a parameter list is invalid, or a reserved field is non-zero. • An operand or parameter list is not aligned on the required storage boundary. • An operand's address and length are such that the operand would cross a 4K-byte boundary, for a DIAGNOSE function that does not support operands crossing such a boundary.
Operand exception	For I/O functions patterned after 370-XA and ESA/390 I/O instructions, an invalid subchannel ID was specified or an operand contained an invalid field value or a non-zero reserved field.
Access exceptions (see “Access Exceptions” on page 8.)	An error occurred while attempting to reference an operand in storage.

Note that certain conditions, such as an invalid parameter-list field or an attempt to perform an unauthorized function, are reported differently for different DIAGNOSE functions. Consult the description of the individual function for specific information.

Access Exceptions

As explained in Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf) and *z/VM: ESA/XC Principles of Operation*, the term *access exceptions* refers to a class of program interruptions that can arise while trying to refer to an operand in storage. These are:

Addressing exception
The storage address designates a location not available in the target address space. For an address space which is this or another virtual machine's host-primary space, this happens when the address is above the defined storage size and does not designate a location in a saved segment loaded into the address space. For an address space created through VM Data Spaces services, this happens when the address is above the address-space size specified when the space was created.

Protection exception
The attempted storage reference (fetch or store) is forbidden by one of several protection mechanisms. These mechanisms are described below.

ALET-specification exception
For an XC virtual machine in access-register mode, the ALET designating the address space in which the operand resides is not a correctly-formed ALET.

Addressing-capability exception
For an XC virtual machine in access-register mode, the ALET designating the address space in which the operand resides is correctly-formed but designates a host access-list entry that is in the revoked state.

ALEN-translation exception
For an XC virtual machine in access-register mode, the ALET designating the address space in which the operand resides is correctly-formed but does not designate a host access-list entry that is in either the valid or the revoked state.

For more information on ALETs and on host access-list entries and their states, see *z/VM: ESA/XC Principles of Operation*.

Condition Codes and Return Codes

After CP executes a DIAGNOSE code, certain responses may be sent back to you indicating whether the execution was successful. Three types of responses that you may receive are:

- **Condition codes** (sometimes referred to as **completion codes**), which appear in the PSW (program status word).
- **Return codes**, which appear in a register. The exact register where they appear is specified in the responses section of the documentation for each DIAGNOSE code, along with both decimal and hexadecimal values for each return code.
- **Program Exceptions**.

The responses section of each DIAGNOSE code shows the significance of all applicable condition codes and return codes.

Storage Protection Mechanisms

Several means are provided to control access to storage by DIAGNOSE and other instructions. Some of these apply to *fetch* references, that is, attempts to use the contents of storage as input to a function. Others apply to *store* references, that is, attempts to place function results into storage. A third class of reference, *update*, involves both fetching and storing. Both fetch- and store-protection mechanisms apply to update references. Violation of these mechanisms generally results in a program interruption for protection exception, or in a return code specific to the invoked DIAGNOSE function.

Some protection mechanisms apply to *synchronous* references only, that is, to references that occur during the execution of the DIAGNOSE instruction itself. *Asynchronous* references, such as those for an I/O operation initiated by a DIAGNOSE function that does not wait for completion, are not subject to such mechanisms. In addition, some mechanisms are not enforced for I/O and cross-virtual-machine communication functions at all, whether they are synchronous or not. In all cases, an access must pass every enforced protection mechanism to succeed. If any test fails, access is denied.

Unless otherwise specified, the DIAGNOSE instruction enforces all applicable protection mechanisms.

The specific protection mechanisms are:

Key-controlled protection
Under key-controlled protection, a four-bit access key (usually from the PSW, but for some operations, specified as an operand) is compared with the access-control bits of the storage key of each 4K-byte block in guest absolute storage. If the keys match, or if the access key is zero, then the access is permitted; if not, then store accesses are forbidden, and an additional fetch-protection bit for the storage block determines whether fetch accesses are forbidden as well. Not all DIAGNOSE functions enforce key-controlled protection.

In XA, ESA, XC, and Z virtual machines, certain additional mechanisms can modify key-controlled protection for storage references synchronous with the instruction (DIAGNOSE). They are:

Fetch-protection override
Fetch-protection override (FPO) inhibits key-controlled fetch protection for effective addresses 0-2047 in the host-primary address space; these addresses generally map to reserved locations in the prefix area. That is, FPO causes fetches from these addresses to be permitted even if the keys do not match and the block at 0-4095 is marked fetch-protected. FPO is enabled by control register 0, bit 6; if this bit is off, then FPO is not invoked, and addresses 0-2047 are handled exactly like addresses 2048-4095. Not all DIAGNOSE functions honor FPO. Also, because FPO

suppresses key-controlled protection, it is not relevant in cases where key-controlled protection is not enforced.

Storage-protection override
Storage-protection override (SPO) is an optional feature on ESA/390 processors. SPO appears installed on the virtual machine if and only if it is installed on the real machine. SPO suppresses key-controlled protection, for both fetches and stores, for storage blocks whose access-control bits have the value 9, regardless of the value of the access key. SPO is enabled by control register 0, bit 7; if this bit is off, then SPO is not invoked, and key 9 is treated no differently from other keys. Because SPO suppresses key-controlled protection, it is not relevant in cases where key-controlled protection is not enforced.

Fetch-protection override and storage-protection override are not applicable to asynchronous references, nor to references to channel control words (CCWs), indirect data address words (IDAWs), or the data designated by CCWs and IDAWs.

Low-address protection
Low-address protection (LAP) prevents stores into effective addresses 0-511 in the host-primary address space; these addresses generally map to reserved locations in the prefix area. LAP is enabled by control register 0, bit 3; if this bit is off, then LAP is not invoked, and addresses 0-511 receive no special protection. LAP applies only to synchronous store references. LAP does not apply to data buffers referenced by CCWs and IDAWs. Not all DIAGNOSE functions enforce LAP.

Host page protection
Host page protection prohibits storing into those 4K-byte blocks that contain read-only pages of a saved segment. Host page protection also prohibits changing the storage keys for those pages. Host page protection applies to both synchronous and asynchronous store references and is enforced by all DIAGNOSE functions.

Host access-list-controlled protection
Host access-list-controlled protection prohibits storing into address spaces referenced through host access-list entries that convey read-only access. This mechanism never applies to references to the host-primary space made implicitly or through ALET X'00000000', because these references do not involve an access-list entry. Host access-list-controlled protection *does* apply to references through a non-zero ALET which happens to designate the host-primary space. Host access-list-controlled protection is applicable only in access-register mode in an XC virtual machine, because that is the only mode in which the program can store into spaces other than the host-primary space. Host access-list-controlled protection applies to both synchronous and asynchronous store references and is enforced by all DIAGNOSE functions.

Table 2 on page 10 summarizes these storage protection mechanisms and the cases in which they apply to DIAGNOSE functions.

Table 2. Summary of Storage Protection Mechanisms

	Applies to			
Protection Mechanism	**Synch Fetch**	**Synch Store**	**Asynch Fetch**	**Asynch Store**
Key-controlled protection	Varies	Varies	Varies	Varies
Fetch-protection override	Varies[1]	No	No	No
Storage-protection override	Varies[2]	Varies[2]	No	No
Low-address protection	No	Varies	No	No
Host page protection	No	Yes	No	Yes
Host access-list-controlled protection	No	Yes	No	Yes

Table 2. Summary of Storage Protection Mechanisms (continued)

Protection Mechanism	Applies to			
	Synch Fetch	Synch Store	Asynch Fetch	Asynch Store

Legend:

Yes
All DIAGNOSE functions enforce the mechanism on this type of access.

Varies
Some DIAGNOSE functions enforce the mechanism on this type of access. Unless otherwise specified in the DIAGNOSE function description, the mechanism *is* enforced.

No
The mechanism does not apply to this type of access.

1

Fetch-protection override is a modification to key-controlled protection; therefore, it can apply only when key-controlled protection applies. However, some DIAGNOSE functions which enforce key-controlled protection still do not honor fetch-protection override. Unless otherwise specified in the DIAGNOSE function description, fetch-protection override *does* apply to synchronous fetches whenever key-controlled protection is enforced.

2

Storage-protection override is a modification to key-controlled protection for synchronous storage references; it applies when and only when key-controlled protection is enforced.

DIAGNOSE Codes That Are Not Programming Interfaces

The DIAGNOSE codes described in Chapter 2, “The IBM-Supplied DIAGNOSE Codes,” on page 13 are supported Programming Interfaces of z/VM.

The DIAGNOSE codes described in Appendix F, “Reserved DIAGNOSE Codes,” on page 1015 are reserved for IBM use and are NOT Programming Interfaces.

Chapter 2. The IBM-Supplied DIAGNOSE Codes

This chapter contains information on the individual DIAGNOSE codes available to z/VM users. Wherever possible, this information includes the following:

- Privilege class of the DIAGNOSE code
- Entry and exit register values
- Condition codes and return codes
- Programming exceptions
- Usage notes.

These DIAGNOSE codes are provided by IBM and by default are enabled. You can disable, modify, and enable them with system configuration statements or commands. See *z/VM: CP Commands and Utilities Reference* for specific command information and *z/VM: CP Exit Customization* for "how-to" information.

DIAGNOSE Code X'00' – Store Extended-Identification Code

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'00' to examine the z/VM extended-identification code.

Entry Values:

Rx
Contains the guest real storage address where you want to store the z/VM extended-identification code. This address must be aligned on a doubleword boundary.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space into which you want to store the z/VM extended-identification code. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the z/VM extended-identification code is in the host-primary address space.

Ry
Contains the number of bytes that you want to store. This value is an unsigned binary number.

Exit Values:

Ry
Contains its original value less the number of bytes that were stored.

Usage Notes

1. For a discussion of how z/VM processes addresses, see “How Addresses Are Processed” on page 5.
2. Execution of the DIAGNOSE code X'00' instruction makes a copy of the information shown in Table 3 on page 13 available to your virtual machine at the location specified by Rx (and in Ax if your virtual machine is an XC virtual machine in access-register mode).

   Please note that the service level reported assumes that all service has been applied for this level. If the service updates have been applied selectively, then the service level reported may be incorrect.

   Table 3. DIAGNOSE code X'00' — Bit Map Fields

Field	Characteristics	Description
Reserved (formerly "System name")	8 bytes, EBCDIC	Contains the constant ‘VM/ESA’ (for compatibility with the prior product name)

© Copyright IBM Corp. 1991, 2020

Table 3. DIAGNOSE code X'00' — Bit Map Fields (continued)

Field	Characteristics	Description
Environment	2 bytes, binary	Identifies the z/VM execution environment. See “2.a” on page 14 for more information.
Version Information	1 byte, hexadecimal	The version number of the product identified in the System Name field. It is an unsigned binary number. See usage note “5” on page 16.
Version code	1 byte, hexadecimal	Bits 0-7 of the host system's CPUID. This field will always be 00 when the z/VM host system is running in a logical partition. This field will always be FF when the z/VM host system is running on z/VM.
reserved	2 bytes, hexadecimal	z/VM stores zeros in this field.
Processor address	2 bytes, hexadecimal	The address of the processor on which z/VM is currently running
User ID	8 bytes, EBCDIC	The user ID of the virtual machine issuing the DIAGNOSE
Licensed program bit map	8 bytes, hexadecimal	The level of CP that is installed. See “2.b” on page 14 for the bit settings.
Time zone differential	4 bytes, hexadecimal	Represents the time zone differential in seconds from Coordinated Universal Time (UTC) (see note “2.c” on page 16 below).
Release Information	4 bytes, hexadecimal	The first byte is the release number, the second byte is the release modification level, and the third and fourth bytes are the service level. All three subfields are unsigned binary numbers.

Notes:

a. The z/VM execution environment bits contain:

- When bit 0 = 1, CP is running in a logical partition (LPAR mode). Bit 0 will always have a value of 1 because the machines supported by z/VM V6.1 and later provide only LPAR mode.
- When bit 1 = 1, CP is running in 64-bit mode. Bit 1 will always have a value of 1 because the DIAGNOSE code X'00' instruction can be issued only when CP runs in 64-bit mode.
- Bits 2 to 15 are reserved and are currently zeros.

b. The bit map contains:

X'0000000000000000'
for VM/XA SF 1

X'4000000000000000'
for VM/XA SF 2

X'6000000000000000'
for VM/XA SP™ 1

X'7000000000000000'
for VM/XA SP 2

X'7800000000000000'
for VM/XA SP 2 with APSS

X'7C00000000000000'
for VM/XA SP 2.1

X'7E00000000000000'
for VM/XA SP 2.1 spool file origin enhancements

X'7F00000000000000'
for VM/ESA Version 1 Release 1.0 ESA Feature

X'7F80000000000000'
for VM/ESA Version 1 Release 1.1

X'7FC0000000000000'
for VM/ESA Version 1 Release 2.0

X'7FE0000000000000'
for VM/ESA Version 1 Release 2.1

X'7FF0000000000000'
for VM/ESA Version 1 Release 2.2

X'7FF8000000000000'
for VM/ESA Version 2 Release 1.0

X'7FFE000000000000'
for VM/ESA Version 2 Release 2.0

X'7FFF000000000000'
for VM/ESA Version 2 Release 3.0

X'7FFF800000000000'
for VM/ESA Version 2 Release 4.0

X'7FFFC00000000000'
for z/VM Version 3 Release 1.0

X'7FFFE00000000000'
for z/VM Version 4 Release 1.0

X'7FFFF00000000000'
for z/VM Version 4 Release 2.0

X'7FFFF80000000000'
for z/VM Version 4 Release 3.0

X'7FFFFC0000000000'
for z/VM Version 4 Release 4.0

X'7FFFFE0000000000'
for z/VM Version 5 Release 1.0

X'7FFFFF0000000000'
for z/VM Version 5 Release 2.0

X'7FFFFF8000000000'
for z/VM Version 5 Release 3.0

X'7FFFFFC000000000'
for z/VM Version 5 Release 4.0

X'7FFFFFE000000000'
for z/VM Version 6 Release 1.0

X'7FFFFFF000000000'
for z/VM Version 6 Release 2.0

X'7FFFFFF800000000'
for z/VM Version 6 Release 3.0

X'7FFFFFFC00000000'
for z/VM Version 6 Release 4.0

X'7FFFFFFE00000000'
for z/VM Version 7 Release 1.0

X'7FFFFFFF00000000'
for z/VM Version 7 Release 2.0

Bit 13 (X'0004000000000000') indicates whether Year 2000 support is present in CP.

To determine what VM environment you are in and what CP resources are available to you, use the high-order bit from this bit map. Specifically, if the high-order bit is on, the environment is VM/SP, VM/SP HPO, or VM/ESA® (370 Feature). If the high-order bit is off, the environment is VM/XA SP, VM/ESA (ESA Feature), or z/VM. Do not try to get this information from the first five bytes returned by DIAGNOSE code X'00'.

c. The time zone differential is a signed hexadecimal fullword value in seconds. Negative values represent differentials west of Coordinated Universal Time (UTC), and positive values represent differentials east of Coordinated Universal Time.

3. If z/VM is executing in a virtual machine, up to 40 bytes of extended identification data is appended to the first 40 bytes described above. Up to five nested levels of z/VM virtual machines are supported by this DIAGNOSE instruction. Thus, a maximum of 200 bytes of data can be returned to the virtual machine that initially issued the DIAGNOSE instruction.
4. The CP level does not indicate the release of CMS. To determine the release of CMS, use the "Query Functional Level of CP and CMS" routine (DMSQEFL), which is described in *z/VM: CMS Callable Services Reference*. DMSQEFL also returns the release of CP.
5. In VM/ESA Version 2 Release 1.0, the Version Information byte was created from the third byte of the Environment field. Because VM releases prior to VM/ESA Version 2 Release 1.0 are not being updated by APAR, the Version Information for all of these releases is X'00'. For VM/ESA Version 2 Release 1.0 and later, the Version Information will reflect the true version number. For example, the Version Information for VM/ESA Version 2 Release 1.0 is X'02'.
6. When part of an SSI cluster, the diagnose x'00' output depends on the relocation domain to which the issuing user is assigned. This output shows the virtual CP level that is being presented to the issuer of the diagnose. Virtual CP levels may differ among users on the same system when they belong to different relocation domains. See the explanation of relocation domains in *z/VM: CP Planning and Administration* for more details. Also see the documentation of the CP SET VMRELOCATE and CP QUERY VMRELOCATE commands and the VMRELOCATE directory statement for details about how a user is assigned to a relocation domain.

Responses

Program Exceptions: These program exceptions can occur if the DIAGNOSE code X'00' is given incorrect input data:

Problem Encountered	Cause
Specification exception	The address contained in Rx is not on a doubleword boundary.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store the extended ID code.

DIAGNOSE Code X'04' – Examine Host Storage

Privilege Class: C, E

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'04' to examine host storage for diagnostic, monitoring, and tuning purposes only.

Entry Values:

Rx
Contains the guest real address of a list of host addresses that you want to examine.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the list of host addresses that you want to examine. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the list of host addresses is in the host-primary address space.

Ry
Contains:

- The number of entries in the list
- An indicator of the address format (31-bit or 64-bit).
- An indicator of the length of data to be returned for each entry.
- An indicator of the address type (host logical or host real)

Ry+1
Contains the guest real address of the result field. The result field contains the values retrieved from the specified host locations.

When Ry is register 15, Ry+1 is register 0.

Ay+1
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space of the result field. If Ry+1 designates general register 0, if Ay+1 contains X'00000000', or if the virtual machine is not in XC mode, the result field is in the host-primary address space.

Usage Notes

1. The guest real addresses that you specify in Rx and Ry+1 must be in the same page of guest real storage. In XC virtual machines in access register mode, the ALET in Ax (or zero if Ax designates access register 0) must match the ALET in Ay+1 (or zero if Ay+1 designates access register 0).
2. Because this DIAGNOSE code is intended mainly for system performance monitoring, it is recommended that the page addressed by Rx and Ry+1 also be resident in host storage at the time the DIAGNOSE instruction is executed. This can be accomplished by having the instruction itself on the same page or by locking the page in host storage; otherwise, DIAGNOSE code X'04' processing causes a page to be paged-in and therefore may affect the performance monitoring data.
3. For a discussion of how z/VM processes addresses, see “How Addresses Are Processed” on page 5.
4. For each address in the list of host addresses, z/VM provides the data obtained from that address. z/VM stores this data into the result field identified by Ry+1, and possibly by Ay+1 if the virtual machine is an XC virtual machine running in access-register mode.

   Entries in the list of addresses pointed to by Rx correspond (1-to-1) to the entries in the result field pointed to by Ry+1. Execution of this DIAGNOSE can be thought of as a loop that fetches the first element from the list of addresses (pointed to by Rx) and stores the data from that location into the first element of the result field (pointed to by Ry+1). Then the second address is fetched and the second data item is stored. This continues until the number of addresses in the count field have been fetched and the corresponding data stored or until an exception is encountered.

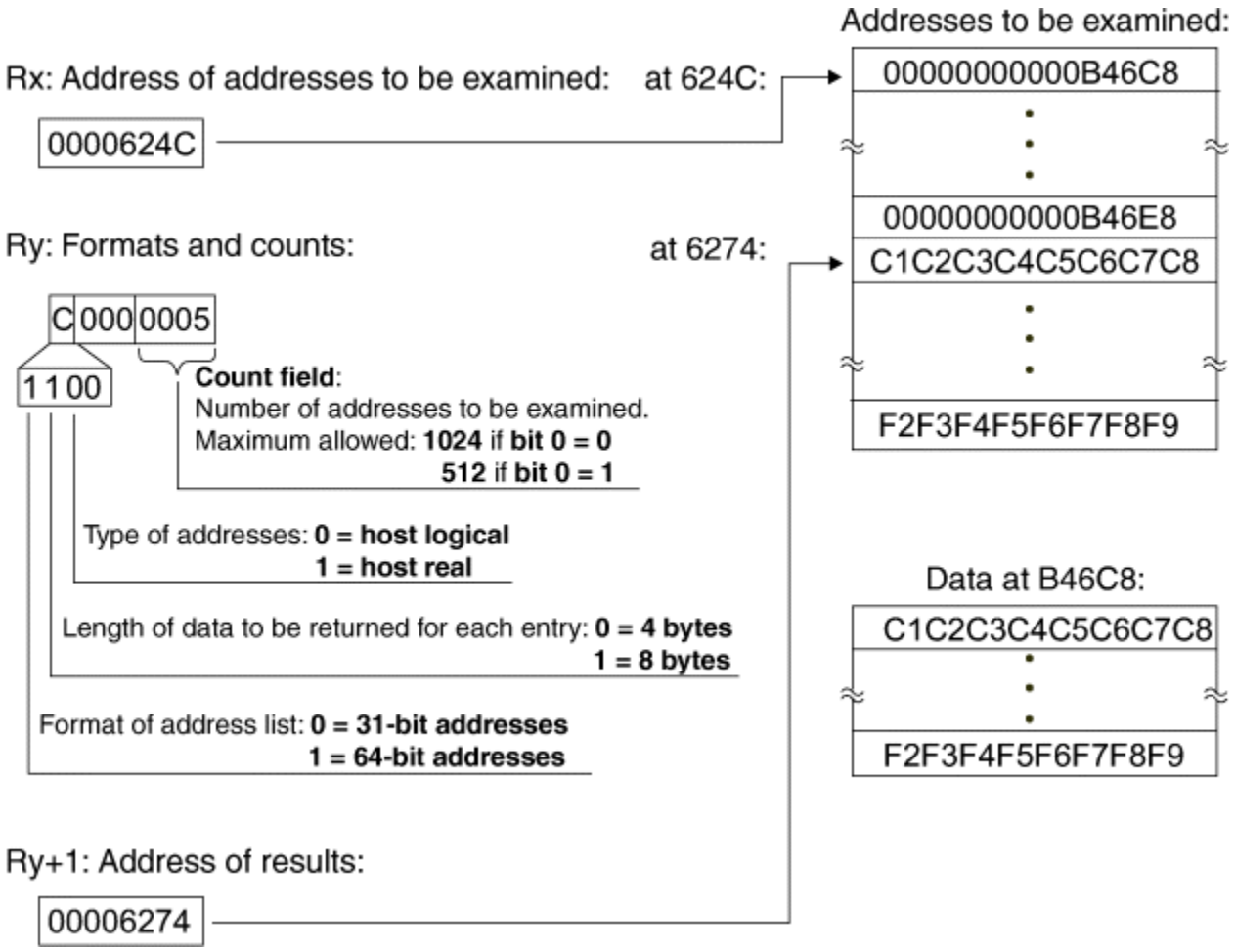


Figure 2. DIAGNOSE X'04' Register Entries

It is possible for the results array to overlap the address array. If destructive overlap occurs, incorrect data may be returned. An addressing exception may be presented when destructive overlap causes an address to change before it is fetched.

An overlap can occur only if the Ry+1 address is greater than the Rx address and yet within the bounds of the addresses to be examined.

5. In a multiprocessor environment, each processor has a prefix register that it uses to relocate addresses between 0 and 4095 (8191 if the host is in z/Architecture mode) to another frame in storage. The prefix register enables each processor to use a different frame to avoid conflict with other processors for such activity as interrupt code recording. Thus, the above range refers to different areas of storage, depending upon which processor generates the address. All references to host real storage are handled as if they were made on the master processor.
6. Fetching and storing of data is consistent with the *ESA/390 Principles of Operation* for both operands of the MOVE (MVC) instruction.
7. Because real storage is volatile, the contents of host storage can change at any time after it is copied into the result field.
8. The number of entries in the list that you specify in Ry must be greater than zero and cannot exceed:
   - 1024 if bit 0 of Ry is 0
   - 512 if bit 0 of Ry is 1

   Otherwise, a specification exception is returned.
9. If bit 0 of Ry is 0 (indicating 31-bit addresses), bit 1 of Ry must be 0 (indicating a length of 4 bytes). If bit 0 of Ry is 1 (indicating 64-bit addresses), bit 1 of Ry must be 1 (indicating a length of 8 bytes).

10. If the loaded data is greater than 4 KB in length, the frames backing the host logical storage pages are not necessarily contiguous in host real storage.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'04' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Any of the following: • In access-register mode in an XC virtual machine, the ALETs from Ax and Ay+1 do not match. • The addresses in Rx and Ry+1 are not on the same page of guest real storage (see Usage Note "1" on page 17). • Incorrect Ry bit settings; only the following combinations are supported: – Bit 0 = 0, bit 1 = 0, bit 2 = 0 (31-bit host logical addresses, 4 bytes of data) – Bit 0 = 0, bit 1 = 0, bit 2 = 1 (31-bit host real addresses, 4 bytes of data) – Bit 0 = 1, bit 1 = 1, bit 2 = 0 (64-bit host logical addresses, 8 bytes of data) – Bit 0 = 1, bit 1 = 1, bit 2 = 1 (64-bit host real addresses, 8 bytes of data) • Any of the non-defined bits in Ry are 1. • The number of list entries is outside the range specified in Usage Note "8" on page 18. • The list of requested storage to examine or the result fields cross a page boundary. • Privilege class violations.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See "Access Exceptions" on page 8.)	An error occurred trying to: • Fetch the address list • Store into the result field.
Addressing exception	• The host address is invalid or designates a directory page. • The host address that you want to examine is not available.

DIAGNOSE Code X'08' – Virtual Console Function

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'08' to have your virtual console issue CP commands. Your virtual machine must specify the command and operands, and indicate whether CP is to return the command response to your display or to a buffer. In addition to returning the command response, CP sets a completion code and may set a condition code.

Entry Values:

Rx
Must point to the character string in guest real storage (storage that appears real to the issuing virtual machine) containing the CP commands and parameters. If the character string contains multiple commands, each command and its associated parameters must be separated from adjacent commands by the value X'15'.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the CP commands and parameters. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the CP commands and parameters are in the host-primary address space.

Rx+1
If the X'40' flag is set in the flag byte of Ry then Rx+1 points to the buffer in guest real storage where CP is to return the command response. This address must be real to your virtual machine. The buffer can cross page boundaries.

Ax+1
Used only by XC virtual machines in access-register mode. If the X'40' flag is set in the flag byte of Ry then Ax+1 contains the ALET for the address space of the buffer for the command response.

Ry
In z/Architecture mode the high order word of register Ry is ignored. The 4 bytes of the low order register word contains the following:

- The high-order byte containing flag bits
- Three bytes that specify the length of the CP commands and parameters; the length is given in bytes and the maximum allowable length is 240 characters.

You can set the flag bits as follows:

X'80'
When this flag is ON, passwords on the LINK, AUTOLOG, and XAUTOLOG commands are permitted only when the SET PASSWORD *command* INCLUDE option is in effect. If the SEPARATE option is in effect for the associated SET PASSWORD command, the command specified in this DIAGNOSE will be rejected with return code 118 (command format not valid) in Ry+1.

When this flag is OFF, the SET PASSWORD command is ignored and the password can be included on the specified command.

X'40'
CP is to return the command response in the buffer pointed to by register Rx+1 (and possibly Ax +1 if the virtual machine is an XC virtual machine running in access-register mode).

X'20'
CP is to return a request to the virtual machine that a password is needed for the LINK, AUTOLOG, or XAUTOLOG commands. This allows the virtual machine to prompt the user for the password.

If the last three bytes in Ry equal the value X'000000', the DIAGNOSE command causes a console function read to be issued to your terminal. The response buffer is ignored.

If the command response is to be returned in a buffer, Rx and Ry cannot be consecutive registers, and neither can be register 15. In addition, the Ry+1 register must be set up as follows:

Ry+1
In z/Architecture mode the high order word of register Ry+1 is ignored. The 4 bytes of the low order register word contains the length of the buffer. The length specified must be greater than 0. Ry+1 only applies if the X'40' flag is specified in the flag byte in the leftmost byte of Ry.

Exit Values:

Ry
If an error is encountered while processing the command issued using DIAGNOSE code X'08', CP issues an error message and sets a return code in Ry. The return code is the hexadecimal

representation of the numeric portion of the error message. For example, if error message 045E is returned, CP sets a return code of X'002D', which is the hexadecimal representation of the message number, 45.

If the virtual machine assumes responsibility for prompting and a password is not in the command buffer on a LINK, AUTOLOG, or XAUTOLOG command issued through DIAGNOSE code X'08', and the information in the command buffer is valid, one of five unique return codes (in hexadecimal representation) is passed back to the virtual machine by register Ry. These return codes indicate which password prompt the virtual machine should issue. For the AUTOLOG and XAUTOLOG commands, the short logon prompt and the extended logon prompt return codes are 8013 (X'1F4F'), 8016 (X'1F50'), and 8017 (X'1F51'), respectively. The LINK, AUTOLOG, or XAUTOLOG request should then be reissued with another DIAGNOSE code X'08' with the password in the command buffer.

Usage Notes

1. If the CP command response is to be returned to the user's buffer, and is an error message, the buffer contains the error code and text if the user's EMSG setting is ON, OFF or IUCV. If the user's EMSG setting is TEXT, only the error message text is placed in the buffer. If the EMSG setting is CODE, only the error code is placed in the buffer. In all cases, the Ry+1 register contains the length of all data placed in the buffer. If the buffer is too small to contain the entire response, as many full lines as will fit are placed in the buffer.

   If the CP command is not returned in the user's buffer, it is displayed according to the EMSG setting.

   An exception to this is if an invalid command is encountered during DIAGNOSE code X'08' processing, the CP error message *HCPCMD001E UNKNOWN CP COMMAND* is not displayed or returned in the user's buffer. The Ry register contains a return code of 1.
2. If CP is executing multiple commands and encounters an invalid command, processing stops, and CP ignores the remaining commands.
3. If you use DIAGNOSE code X'08' to issue the XAUTOLOG command (without the SYNC option), the response is not returned in a buffer. This is because this is an asynchronous command, and it has not necessarily completed by the time the DIAGNOSE instruction finishes execution.
4. You can use the SET D8ONECMD command to control whether CP will accept multiple commands imbedded in a single command and separated with X'15' characters. You can also use the QUERY D8ONECMD command to find out the D8ONECMD settings (number of commands) for your own virtual machine or another user's virtual machine. For more information, see *z/VM: CP Commands and Utilities Reference*.

Responses

Condition Codes: If the command response is to be returned in a buffer, CP sets a condition code and returns information as follows:

Condition Code	Status	Ry Contains	Ry+1 Contains
0	The DIAGNOSE request was successful (although the CP command may not have been).	The return code from the CP command. The return code 6890 indicates that the executing virtual machine's D8ONECMD setting was FAIL.	The length of the response or error message, if it was to be returned in a buffer
1	The DIAGNOSE request was not successful. The response does not fit in the user-specified response buffer.	The return code from the CP command. The return code 6890 indicates that the executing virtual machine's D8ONECMD setting was FAIL.	A value that specifies how many bytes of the response or error message would not fit into the buffer[1]

Note: [1]If the response buffer overflow is greater than X'7FFFFFFF', then Ry+1 contains the value X'7FFFFFFF'.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'08' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • The command buffer is greater than 240 bytes long. • A response buffer is specified, and Rx or Ry is specified as R15. • A response buffer is specified, and Rx and Ry are specified as consecutive registers. • The length of the response buffer is zero. • The response buffer exceeds the architected storage limit (16MB for 24-bit ESA; 2048MB for 31-bit ESA).
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to: • Fetch the command string • Store the command response.

Examples

The following are two examples showing how to specify DIAGNOSE code X'08'. Note that this is not part of the interface definition. Depending on your conventions and programming language, your code may look quite different.

The first example shows how a program issues the SET IMSG OFF command. In this example, the response is returned to the user's terminal:

```
           LA      6,CMMD
           LA      10,CMMDL
           DC      X'83',X'6A',XL2'0008'
            .
            .
            .
CMMD       DC      C'SET IMSG OFF'
CMMDL      EQU     *-CMMD
            .
            .
```

The second example shows how to specify a string of commands when multiple commands are to be issued:

```
           LA      6,CMMD
           LA      10,CMMDL
           DC      X'83',X'6A',XL2'0008'
            .
            .
            .
CMMD       DC      C'SET EMSG OFF'
           DC      X'15'
           DC      C'PURGE PRT ALL'
CMMDL      EQU     *-CMMD
```

DIAGNOSE Code X'0C' – Pseudo Timer

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

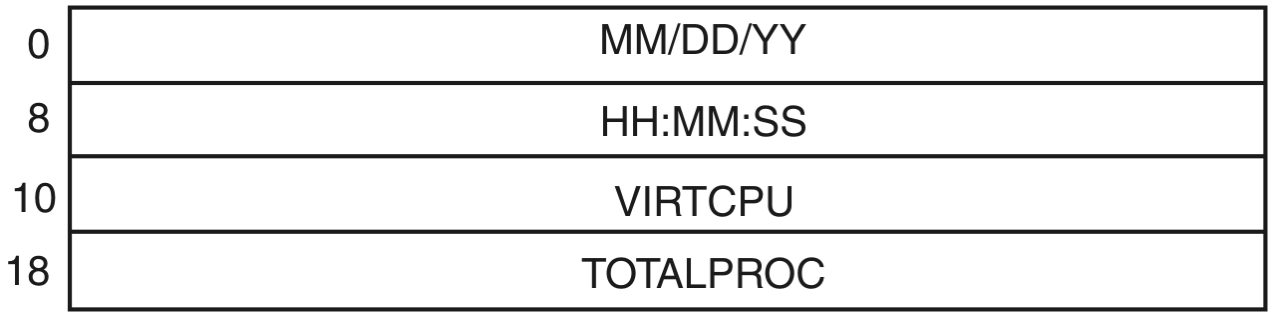
Use DIAGNOSE code X'0C' to cause CP to store four doublewords of time information at the address specified in the Rx register.

Entry Values:

Rx
Contains the address of a 32-byte area where the time information is to be stored. The address must be in second-level storage (that is, in the storage that appears real to your virtual machine) and must be on a doubleword boundary.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the 32-byte area where the time information is to be stored. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the time information area is in the host-primary address space.

Exit Values: The output area contains the following information:



The first eight bytes of the output area contain the date, month/day-of-month/year, in EBCDIC. The next eight bytes contain the time of day, hours:minutes:seconds, in EBCDIC. The next eight bytes (16 through 23), VIRTCPU, contain the virtual time consumed by the virtual CPU that issued the DIAGNOSE instruction. The last eight bytes (24 through 31), TOTALPROC, contain the total of the virtual time (VIRTCPU) and the simulation time spent on behalf of the virtual CPU that issued the DIAGNOSE instruction. Thus, TOTALPROC is always greater than or equal to VIRTCPU. The difference between them represents the time that CP has spent specifically on behalf of the virtual CPU.

These values are also part of the response for the CP command, INDICATE USER. The last 16 bytes contain the virtual and total processor time used by the virtual machine that issued the DIAGNOSE instruction. VIRTCPU and TOTALPROC are doubleword, unsigned integers; the time is expressed in microseconds, not as TOD clock units.

Usage Notes

For a discussion of how z/VM processes addresses, refer to “How Addresses Are Processed” on page 5.

Responses

Program Exceptions: These program exceptions can occur if the DIAGNOSE X'0C' is given incorrect input data:

Problem Encountered	Cause
Specification exception	The address contained in Rx is not on a doubleword boundary.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store into the time-information area.

DIAGNOSE Code X'10' – Release Pages

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'10' to release pages of second-level storage.

Entry Values:

Rx
Contains the guest real address of the first page of storage to be released. This address must be on a page boundary, otherwise a specification exception occurs.

Ax
For XC virtual machines in the access-register mode, Ax contains the ALET designating the address space in which storage is to be released. Even when Rx is general register 0, the **actual contents** of Ax are used in XC access-register mode.

Ry
Contains the guest real address of the last page to be released. This address must be on a page boundary, and must be greater than or equal to the address specified in the Rx register; otherwise, a specification exception occurs.

The results of the DIAGNOSE X'10' operation on a particular page depend on whether the page is a mapped or unmapped page, and for unmapped pages, whether the page is contained in a saved segment or not. A page is considered mapped when the DEFINE function of the MAPMDISK macro has been used to associate the page with a DASD block on a minidisk; otherwise, it is an unmapped page. If the page is:

- An unmapped page that is not contained in a saved segment, then the page is unlocked (if it was locked through the LOCK command), the current contents of the page are discarded and the page is considered to contain binary zeros.
- An unmapped page that is contained in a saved segment, then no operation is performed. The page remains unchanged.
- Is a mapped page, then the page is unlocked (if it was locked through the LOCK command), the current contents of the page are discarded, and upon next reference the contents of the page are refreshed from the associated DASD block.

The range of storage between the addresses specified in the Rx and Ry registers must be contained entirely within the defined storage for the address space. That is, this DIAGNOSE code cannot be used to release discontiguous storage. If an attempt is made to release discontiguous storage, an addressing exception is recognized.

For an XC virtual machine in the access-register mode, a specification exception is recognized if an attempt is made to release storage in a host-primary address space through a nonzero ALET.

Usage Notes

1. The addresses contained in Rx and Ry must be on a page boundary. A page boundary is a storage address whose low-order three digits, expressed in hexadecimal, are zero.
2. Do not use DIAGNOSE code X'10' to release discontiguous storage.
3. If a virtual machine attempts to release a shared page within its defined virtual storage, the page is not released. No further action is taken.
4. It is unpredictable whether a modified mapped page has been saved on its associated mapped DASD block unless the modified page has been specifically saved through the SAVE function of the MAPMDISK macro.

   To ensure that the modified mapped page has been saved on DASD do not release the page through DIAGNOSE code X'10' until the save-completion external interrupt signals completion of the MAPMDISK SAVE function.

5. If DIAGNOSE code X'10' is issued to release pages locked by the CP LOCK command, those pages are unlocked.
6. The largest address allowed for a guest in 24-bit or 31-bit addressing mode is the smaller of the address of the end of the first address space extent and either 2047M for a S/390® Architecture guest or 2048M for a z/Architecture guest. For a z/Architecture guest in 64-bit addressing mode, the limit is the address of the end of the first address space extent.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'10' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • The address of the last page (specified in Ry) is less than the address of the first page (specified in Rx). • The virtual machine attempts to release page 0 of its host-primary address space. • The address contained in Rx or Ry is not on a page boundary (see Usage Note "1" on page 24). • For an XC virtual machine in the access-register mode, an attempt was made to release storage in this or another virtual machine's host-primary address space through a nonzero ALET.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See "Access Exceptions" on page 8.)	An error occurred trying to store into the designated storage area. (Host access-list-controlled protection is the only protection mechanism which generates a protection exception. Key-controlled protection and low-address protection are not enforced. But note that low-address protection is preempted by the specification exception if the page at guest real address 0 is included in the range. Likewise, a host-page-protection exception condition results in either a specification exception (for discontiguous storage) or no operation (for contiguous storage).)
Addressing exception	The designated storage area includes discontiguous storage.
Protection exception	The designated storage area includes read-only storage.

DIAGNOSE Code X'14' – Input Spool File Manipulation

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'14' to access and to control input spool files in your reader. You can access descriptions of files in your reader, change the copy count of a file, read files, backspace to the previous record, or select the next or a specific file to process.

Entry Values:

Rx
The first data register (contents determined by subcode).

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer, if Rx points to a buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
The second data register (contents determined by subcode).

Ry+1
Contains the actual subcode that was issued (Ry and Ry+1 must be an even-odd pair).

The possible subcodes, which are described in the following sections, are:

Subcode
Function

X'0000'
Read the Next Spool File Buffer (Data Record)

X'0004'
Read the Next Print Spool File Block

X'0008'
Read the Next Punch Spool File Block

X'000C'
Order a File to the Front of a Queue

X'0010'
Repeat the Active File a Specified Number of Times

X'0014'
Restart an Active File at the Beginning

X'0018'
Backspace One Record

X'001C'
Read the Next Monitor Spool File Block

X'0020'
Read the Next Monitor Spool Record

X'0024'
Read the Last Spool File Buffer

X'0028'
Position Spool File to Designated Record

X'002C'
Select a File for Processing and Read the Next Spool Buffer

X'0FFE'
Select the Next File Not Previously Selected and Select Read to EOF Files

X'0FFF'
Retrieve Next File Descriptor

Subcode X'0000'—Read the Next Spool File Buffer (Data Record)

Rx
the buffer address (full page)

Ry
the virtual spool reader address

Ry+1
X'00000000' (the subcode)

This subcode returns a page of spool data to the user's buffer. The buffer must be a full page (4K) in length and must not cross a page boundary. If the buffer crosses a page boundary, a specification exception occurs. Subcode X'0000' can be issued consecutively as many times as necessary to read an entire file on the user's virtual reader queue. Different results occur depending on how the virtual reader has been spooled.

If there is no file currently active on the virtual reader device specified, the first data page (in SPLINK format; refer to "SPLINK - VM/SP 370 Spool File Data Block" on page 976) of the first eligible file in the user's virtual reader queue is returned in the user's buffer. A file is eligible if it is not in *hold* status and if its class matches that of the virtual reader. The file is opened on the virtual reader when an eligible file is found.

If there is no file already active on the virtual reader and there are no eligible files, then condition code 2 is returned.

When there is an active file already on the specified virtual reader, condition code 3 with return code X'0C' in Ry+1 is returned if it is not active due to a previous DIAGNOSE code X'14' operation. If it is active due to a previous DIAGNOSE code X'14' operation, the next page of data is returned.

When an SPLINK (data page) from a file created on a punch is returned, all write CCWs in the page are converted and returned as X'41' opcodes.

When EOF (end of file) is reached and *continuous* read is not set, condition code 1 is returned. No data is passed back in the user's buffer and the file remains open.

If *continuous* read is set, processing continues with the first page of data from the next logical file. The search for the next logical file begins either at the top of the user's reader queue or at the file just processed, depending on the reader's *rescan* or *norescan* setting. If the reader is spooled *rescan*, the search begins with the first file in the virtual reader queue. If the reader is spooled *hold/norescan*, the search will begin with the next logical file (after the one just processed) in the virtual reader queue. For all other combinations of *norescan* (with *continuous*) the search will start with the first file in the virtual reader queue.

When a new file is selected because the reader is spooled *continuous*, the previously read file is closed and the newly read file is opened. This may be a read of the first page of the same file if that file was set for multiple copies. When the reader is spooled for *continuous* read, condition code 1 (EOF) is returned when no more eligible files are found. When this condition is reached, no data is returned in the user's buffer and the last file read remains open.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See "Responses" on page 35 to determine the meaning of these return codes.

Subcode X'0004'—Read the Next Print Spool File Block

Rx
the buffer address (13 doublewords)

Ry
the virtual spool reader address

Ry+1
X'00000004' (the subcode)

This subcode returns the next eligible file on the virtual reader queue that was created on a printer or console. A file is eligible if it is not in *hold* status and if its class matches that of the virtual reader. The spool file descriptor block (SFBLOK; refer to Appendix A, "Data Areas Used by DIAGNOSE Codes," on page 969) for the file is returned in the user's virtual buffer. The file is not opened. The buffer must not cross a page boundary. If the buffer crosses a page boundary, a specification exception occurs.

The default size of the spool file block returned is 13 doublewords. However, a different size buffer may be specified by using extended mode. See usage note "1" on page 34 for a description of extended mode.

If the specified reader device is not already in use by DIAGNOSE code X'14' the first eligible printer or console file is returned. If DIAGNOSE code X'14' is using the specified device and if this is a repeat call for a printer spool file block, the chain search continues from the point following the last block given to the virtual machine. In this case, condition code 1 is set when there are no more print files. If this is the first call for a spool file block, or if there have been intervening calls for file reading, the spool input chain is searched from the beginning. If no files are found, condition code 2 is set.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See "Responses" on page 35 to determine the meaning of these return codes.

Subcode X'0008'—Read the Next Punch Spool File Block

Rx
the buffer address (13 doublewords)

Ry
the virtual spool reader address

Ry+1
X'00000008' (the subcode)

Processing for subcode X'0008' is the same as for subcode X'0004', except that only files created on a punch are processed.

Subcode X'000C'—Order a File to the Front of a Queue

Rx
the file ID

Ry
the virtual spool reader address

Ry+1
X'0000000C' (the subcode)

This subcode moves a file to the front of the user's reader queue so that it can be the next one processed.

The spool input queue is searched for the file specified. If it is found and it is not in *hold* status, the file is moved to the front of the queue and condition code 0 is set. The file is ordered to the top of the queue but it is not opened and no spool data is returned.

If the specified file is not found, if the specified file is in *hold* status or if the specified file's class does not match the class of the virtual reader, condition code 2 is set.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See "Responses" on page 35 to determine the meaning of these return codes.

A specification exception occurs if the spool file ID specified is 0 or is greater than 9999.

Subcode X'0010'—Repeat the Active File a Specified Number of Times

Rx
the new copy count for the active file

Ry
the virtual spool reader address

Ry+1
X'00000010' (the subcode)

The count of the active file is set to the specified value, the maximum being 255. If the count is greater than 255, it is set to 255. If the count is negative, a specification error occurs. If there is no active file, condition code 2 is set.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See "Responses" on page 35 to determine the meaning of these return codes.

Subcode X'0014'—Restart an Active File at the Beginning

Rx
unused

Ry
the virtual spool reader address

Ry+1
X'00000014' (the subcode)

If the device has an active file, the record pointers are set to the beginning of the file. Therefore, the next read receives the first file record. If no file is active, condition code 2 is set.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See "Responses" on page 35 to determine the meaning of these return codes.

Subcode X'0018'—Backspace One Record

Rx
the buffer address (full page)

Ry
the virtual spool reader address

Ry+1
X'00000018' (the subcode)

If the device has an active file, the record pointer is set to the previous record unless it is already positioned at the first record. For example, if a normal subcode X'0000' operation was just used to read the third data page of the file, a subsequent subcode X'0018' operation returns the second data page of the file. The record is returned to the user as in a normal X'0000' read subfunction. The buffer must be a full page (4K) in length and must not cross a page boundary. If the buffer crosses a page boundary, a specification exception occurs. The record pointer is set so that the next X'0000' read operation returns the third data page again.

If the record pointer is already positioned at the first record, subcode X'0018' returns condition code 1 and the first page of the file is returned in the buffer. If no file is active, condition code 2 is set.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See "Responses" on page 35 to determine the meaning of these return codes.

When an SPLINK (data page) from a file created on a punch is returned, all write CCWs in the page are converted and returned as X'41' opcodes.

Subcode X'001C'—Read the Next Monitor Spool File Block

Rx
the buffer address (13 double words)

Ry
the virtual spool reader address

Ry+1
X'0000001C' (the subcode)

Condition code 2 is always returned, because monitor spool files are not supported in z/VM.

Subcode X'0020'—Read the Next Monitor Spool Record

Rx
the buffer address (full page)

Ry
the virtual spool reader address

Ry+1
X'00000020' (the subcode)

Condition code 2 is always returned, because monitor spool files are not supported in z/VM.

Subcode X'0024'—Read the Last Spool File Buffer

Rx
the start address of a full-page virtual buffer

Ry
the virtual spool reader address

Ry+1
X'00000024' (the subcode)

The last spool buffer (active file) is read. The specified device is checked for an already active file and, if there is one, the last full-page buffer is made available to the virtual machine in SPLINK format. If there is no active file, the condition code of the virtual machine is set to 2.

The buffer must be a full page (4K) in length and must not cross a page boundary. If the buffer crosses a page boundary, a specification exception occurs.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See “Responses” on page 35 to determine the meaning of these return codes.

When an SPLINK (data page) from a file created on a punch is returned, all write CCWs in the page are converted and returned as X'41' opcodes.

Subcode X'0028'—Position a Spool File to the Designated Record

Rx
the relative number of the record

Ry
the virtual spool reader address

Ry+1
X'00000028' (the subcode)

The number in (Rx) becomes the number of the new current record. The first data page in the file would be record number 1. No spool data is returned with this subfunction. Only the current record pointer is changed so that reading resumes at the specified record the next time a read operation is issued.

Condition code 2 is returned if no file is active or if the device is not ready.

If the value in Rx would set the current record beyond the end of the file, the condition code is set to 3 and code X'14' is returned in Ry+1.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See “Responses” on page 35 to determine the meaning of these return codes.

Subcode X'002C'—Select a File for Processing and Read the Next Spool Buffer

Rx
the start address of full-page virtual buffer

Ry
the virtual spool reader address

Ry+1
nnnnyyyy where *nnnn* is the file identifier of the requested file and *yyyy* is the function subcode.

The spool input chain is searched for the file specified, following the same procedure as subcode X'000C'. If the file is found, it is moved to the head of the chain, the processing continues, and the file is read as if subcode X'0000' had been issued.

If the specified file is not found, if the specified file is in *hold* status or if the specified file's class does not match the class of the virtual reader, condition code 2 is set.

The buffer must be a full page (4K) in length and must not cross a page boundary. If the buffer crosses a page boundary, a specification exception occurs.

Condition code 3 is set if the virtual reader specified does not exist or is not a reader. The appropriate return code is placed in Ry+1 to indicate which of these is the case. See “Responses” on page 35 to determine the meaning of these return codes.

When an SPLINK (data page) from a file created on a punch is returned, all write CCWs in the page are converted and returned as X'41' opcodes.

Subcode X'0FFE'—Select the Next File Not Previously Selected

There are two basic functions of DIAGNOSE code X'14' subcode X'0FFE':

- Select the Next File Not Previously Seen
- Select the Read to EOF Files.

Each of these has an initialize capability and a select capability. The input registers are different for each of these functions. The first byte of Ry determines which of these two functions is being requested. If the first byte of Ry is X'00' then the "Select the Next File Not Previously Seen" function is being requested. If the first byte of Ry is *not* X'00' then the "Select the Read to EOF Files" function is being requested.

(The following two items describe the "Select Next File Not Previously Selected" function.)

Select Next File Not Previously Seen - INIT

Rx
the virtual address of an output buffer; the default size is 252 bytes when invoked without extended mode. See usage note “2” on page 34 for more information. The buffer may cross a page boundary.

Ry
X'00000001'

Ry+1
a flag, optional size of the spool file block in doublewords, and the subcode X'0FFE'. The format of Ry +1 is *aabb*0FFE where

aa
is defined as follows:

bit 0, when 1, identifies extended mode (see usage note “2” on page 34 for more information).
bit 1 is reserved and must be 0
bits 2-7 are either 0 or specify the size, in doublewords, of the buffer space to hold the spool file block.

bb
is not used and should be 00

0FFE
is the DIAGNOSE code X'14' subcode.

All of this user's input spool files that are marked as having previously been seen are reset to the not-yet-seen state, and the descriptor of the first spool file is returned to the user's buffer. Also, CP sets a bit

identifying that this spool file has been seen. If no file is selected, a condition code of 1 is reflected to the virtual machine. The virtual buffer specified by Rx may cross a page boundary.

Select Next File Not Previously Seen - Select

Rx
the virtual address of an output buffer; the default size is 252 bytes when invoked without extended mode. See usage note "2" on page 34 for more information. The buffer may cross a page boundary.

Ry
X'00000000'

Ry+1
a flag, optional size of the spool file block in doublewords, and the subcode X'0FFE'. The format of Ry +1 is *aabb*0FFE where

aa
is defined as follows:

bit 0, when 1, identifies extended mode (see usage note "2" on page 34 for more information).
bit 1 is reserved and must be 0
bits 2-7 are either 0 or specify the size, in doublewords, of the buffer space to hold the spool file block.

bb
is not used and should be 00

0FFE
is the DIAGNOSE code X'14' subcode.

The next reader spool file that has not previously been seen is selected and its descriptor is returned to the user's buffer. Also, CP sets a bit identifying that this spool file has now been seen.

(The next two items describe the "Read to EOF" function.)

Read to EOF - INIT

Rx
not used

Ry
The format of Ry is *nnzzyyyy* where

nn
is not 00

zz
is 00

yyyy
is virtual spool reader address.

Ry+1
a flag, optional size of the spool file block in doublewords, and the subcode X'0FFE'. The format of Ry +1 is *aabb*0FFE where

aa
is defined as follows:

bit 0, when 1, identifies extended mode (see usage note "2" on page 34 for more information).
bit 1 is reserved and must be 0
bits 2-7 are either 0 or specify the size, in doublewords, of the buffer space to hold the spool file block.

bb
is not used and should be 00

0FFE
is the DIAGNOSE code X'14' subcode.

This function initializes (or re-initializes) the virtual spool reader specified in yyyy for reading files to EOF (end of file). Any file previously marked as having been read to EOF on that device is reset, such that it appears to not have been previously read to EOF. A condition code and return code of 0 are reflected to the virtual machine. The INIT function also tells CP to begin to keep track of which spool files are read to EOF on the device. (The SELECT function, described following, resets this.)

Read to EOF - Select

Rx
the virtual address of an output buffer; the default size is 252 bytes when invoked without extended mode. See usage note "2" on page 34 for more information. The buffer may cross a page boundary.

Ry
The format of Ry is *nnzzyyyy* where

nn
is a not 00

zz
is 01

yyyy
is virtual spool reader address.

Ry+1
a flag, optional size of the spool file block in doublewords, and the subcode X'0FFE'. The format of Ry +1 is *aabb*0FFE where

aa
is defined as follows:

- bit 0, when 1, identifies extended mode (see usage note "2" on page 34 for more information).
- bit 1 is reserved and must be 0
- bits 2-7 are either 0 or specify the size, in doublewords, of the buffer space to hold the spool file block.

bb
is not used and should be 00

0FFE
is the DIAGNOSE code X'14' subcode.

A SELECT function is performed. This function tells CP to stop marking files read to EOF on the virtual spool reader specified in *yyyy*, and then returns the descriptor of the first file found to have been read to EOF on the device. The selected file is no longer considered to have been read to EOF. Invoking this function repeatedly (without an intervening INIT function) will return all files which have been read to EOF.

Notes:

1. The Read-to-EOF indicator might be turned off by spooling commands in an SSI environment. If a file had been read to EOF on a system other than the originating system and a spooling command is used to change that file's characteristics, the indicator may be turned off. If this happens, the file is eligible to be read again and is not returned in response to the Read-to-EOF SELECT function.
2. The Read-to-EOF indicator is turned off by a system reset (for example, the CP SYSTEM RESET or IPL command does this). Selective reset of the reader (for example, the CP RESET command) does not turn off the Read-to-EOF indicator.
3. The Read-to-EOF indicator is not turned on for files receiving the EOF indication from their I/O operation unless all records in the file were actually read. This can happen on files which, for example, contain one or more X'5A' records. When these files are read, the X'5A' records are skipped and the Read-to-EOF indicator is not set when the EOF is reached.

For those functions which return a spool file block either the default buffer size can be used or extended mode can be specified. For these functions, the value of Ry+1 is decoded as follows:

If bit 0 of Ry+1 is 0, then the default buffer size is to be used.
If bit 0 of Ry+1 is 1 (that is, extended mode), then bit 1 must be 0 and bits 2 through 7 specify the size, in doublewords, of the spool file block to return.

See usage note “2” on page 34 for the exact format of the data returned.

Subcode X'0FFF'—Retrieve Next File Descriptor

Rx
the output buffer address; the default size is 252 bytes when invoked without extended mode. See usage note “2” on page 34 for more information.

Ry
the spool file ID number

Ry+1
X'00000FFF' (the subcode)

Ry contains a spool file ID or zero and specifies the point at which to begin the search. Zero identifies the beginning of the chain. The next file belonging to the user, regardless of class, file type, or current usage is selected. The spool file block and the first record (generally, the TAG) are placed in the caller's buffer. The buffer may cross a page boundary. If there are no files on the reader queue or if there is no subsequent file, condition code 1 is returned. If a starting file ID is specified and there is no match for this user, condition code 2 is returned.

For subcode X'0FFF' either the default buffer size can be used or extended mode can be specified. Whether the default or extended mode is to be used is determined as follows:

If bit 0 of Ry+1 is 0, then the default buffer size is to be used.
If bit 0 of Ry+1 is 1 (that is, extended mode), then bit 1 must be 0 and bits 2 through 7 specify the size, in doublewords, of the spool file block to return.

See usage note “2” on page 34 for the exact format of the data returned.

Usage Notes

1. You can use extended mode for subcodes X'0004' and X'0008'. In extended DIAGNOSE mode, you can specify the size of a spool file block in the high-order byte of Ry+1. If bit 0 is 1, bit 1 must be 0, and bits 2 through 7 define the size of the block in doublewords to be returned. If a block size of 0 is specified in extended mode, a specification exception occurs.
2. Subcodes X'0FFE' and X'0FFF' can be issued in either extended mode or not in extended mode.

   When invoked without the extended mode indicator, the spool file block (SFBLOK) and the first record (generally, the TAG) are placed in the caller's buffer. The spool file block is 13 doublewords long and the first data record follows. The total length of this data is 252 bytes.

   When invoked in extended mode, an additional 40 bytes of 3800 printer information (such as CHARS, FCB, COPY and FLASH) is returned between the spool file block and first data record. The length specified in Ry+1 is the length of the spool file block returned. The total length of data returned in the user's buffer is the sum of the spool file block length specified, the 40 bytes of 3800 information (SPCHAR to SPPGLEN in SPLINK), and the 148 bytes of the TAG record. A length of 0 can be specified for the spool file block. In this case, only the 3800 printer information and the TAG data are returned.
3. For a description of the spool file block information returned, refer to SFBLOK and SPLINK in Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969. DSECTs are provided in the HCPGPI macro library as SFBLOK COPY and SPLINK COPY.
4. If an external security manager is installed on your system, subcodes X'0000', X'0004', X'0008', X'002C', X'0FFE', and X'0FFF' may be affected. When these subcodes are issued, the system may skip spool files that you are not authorized to work with. For additional information, contact your security administrator.

Responses

Condition Codes: The following condition codes are applicable for all subcodes except the X'0FFE' SELECT function.

Condition Code	Return Code in Ry +1	Meaning
0	None	Successful completion
1	None	End-of-file reached
2	None	File not found
3	04 (X'04')	Device address invalid
3	08 (X'08')	Device type invalid
3	12 (X'0C')	Device busy, reader not ready, or device real
3	16 (X'10')	Fatal paging I/O error
3	20 (X'14')	Invalid file; for subcode X'028': requested record number exceeds the file's size

The following condition codes are applicable for subcode X'0FFE' SELECT function.

Condition Code	Return Code in Ry +1	Meaning
0	None	Successful completion
1	None	No file found meeting SELECT criteria.

Program Exception: These program exceptions can occur if DIAGNOSE code X'14' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • Ry is not an even register. • Ry+1 does not contain a valid subcode. • A buffer crosses a page boundary (except for subfunctions X'0FFE' and X'0FFF'; an exception does *not* occur for these two subfunctions if a buffer crosses a page boundary). • The new copy count is negative. • The file ID specified is greater than 9999. • The file ID specified is 0. • The subfunction called "Select Next File Not Previously Seen" of subcode X'FFE' was issued and Ry is equal to something other than 0 or 1. • Ry+1 indicates extended mode with a buffer length of zero for subcodes X'0004' or X'0008'.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store into the buffer area.

DIAGNOSE Code X'18' – Standard DASD I/O

Privilege Class: Any (370 Accommodation must be active)

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'18' to perform input/output operations to a direct access device of the type fully supported by z/VM.

Note: z/VM includes DIAGNOSE code X'18' primarily for VM/SP, VM/SP HPO, and VM/ESA (370 Feature) compatibility. For new applications and new devices, use DIAGNOSE code X'A4'. When DIAGNOSE code X'A4' is used, CP constructs the appropriate channel program for the device being used. DIAGNOSE code X'A4' can be used in all types of virtual machines.

Entry Values:

Rx
Contains the guest real address of the direct-access device.

Ry
Contains a channel address word (CAW).

The leftmost bits, 0 through 3, of the CAW contain the protection key used in accessing second-level storage for the I/O operation.

Bit 4 (suspend control) must be zero.

Bits 5 through 7 (unassigned) must be zero.

Bits 8 through 31 contain the channel program address.

Bits 29 through 31 must be zero, designating a doubleword-aligned channel program.

Register 15
You must load register 15 with the number of READs or WRITEs in the CCW chain.

Usage Notes

1. No I/O interrupts are returned by CP to the virtual machine; the DIAGNOSE instruction is completed only when all the READ or WRITE commands associated with the DIAGNOSE are completed.
2. Do not use DIAGNOSE code X'18' to read or write record-overflow-formatted data.
3. CP provides error recovery for this DIAGNOSE instruction. The instruction does not complete until the I/O operation has completed, including any error recovery processing.
4. The guest's Rx and Ry registers can be any two registers, neither of which is register 15.
5. The CCW chain specified by the guest must specify I/O on only one cylinder, and must be of the standard format.
6. For the 3350, 3375, 3380, 3390, and 9345 devices, the SET SECTOR CCW should precede each SEARCH CCW.
7. The CCW chain ends with a read or write without command chaining.
8. The CCW chain must not have any indirect data address (IDA), data chain (CD), or program controlled interrupt (PCI) flags.
9. Data crossing 4KB boundaries is handled using indirect data lists.
10. The CCWs must be format 0.
11. The first CCW must be a SEEK CCW.
12. This DIAGNOSE code does not support fixed-block DASD devices. If a program issues a DIAGNOSE code X'18' to a fixed-block DASD device, CP sets cc=1 and places a return code of 2 in register 15.
13. This DIAGNOSE code may be used only when the 370 Accommodation facility is active. Activate the 370 Accommodation facility with the SET 370ACCOM ON command.
14. This DIAGNOSE code does not support HyperPAV alias devices.

15. This DIAGNOSE code does not include Extended Address Volume (EAV) support for minidisks that reside at or above 65520 cylinders. If a program issues a DIAGNOSE code X'18' to such a minidisk, CP sets a condition code of 1 (cc=1) and places a return code of 2 in register 15.

Responses

Condition Codes: On return from DIAGNOSE code X'18', CP sets the condition code as follows. In addition, a return code is placed in register 15.

Condition Code	Return Code in Register 15	Meaning
0	0 (X'00')	I/O completed successfully.
1	1 (X'01')	The device is not attached.
1	2 (X'02')	DIAGNOSE code X'18' does not support this device type, or does not include Extended Address Volume (EAV) support for minidisks.
1	3 (X'03')	An attempt was made to write on read-only disk.
1	4 (X'04')	The cylinder or head number on a seek or seek head CCW is not in the range of the user's disk.
1	5 (X'05')	A virtual device is busy or has an interrupt pending.
2	5 (X'05')	The user's CAW is invalid for one of the following reasons: • Bit 4 (suspend control) is 1. • Bits 5–7 are not B'0000'. • The channel program address is not doubleword-aligned.
2	6 (X'06')	The CCW or the SEEK/SEARCH arguments are not within the range of the user's storage.
2	7 (X'07')	The user's CCW is invalid for one of the following reasons: • Invalid bits are set in the flag byte of the CCW. • The CCWs are not in the order expected by CP. • The byte count in CCW is not valid for the CCW command.
2	8 (X'08')	The READ/WRITE byte count is 0.
2	9 (X'09')	The READ/WRITE byte count is greater than 4096.
2	10 (X'0A')	The READ/WRITE buffer is not within the user's storage or the buffer is protected.
2	11 (X'0B')	The value in register 15, at entry, was not a positive number in the range of 1 to 15, or was less than the number of READs and WRITEs in the channel program.
2	12 (X'0C')	The cylinder number on the SEEK HEAD was not the same number as on the first SEEK.
3	13 (X'0D')	Unrecoverable I/O error. The CSW (8 bytes) was returned to the user. If the CSW indicates a unit check, sense bytes are available if the user issues a SENSE command.

Example

When you are using DIAGNOSE code X'18', your CCW string may look like the following example of a CCW string to read or write two 800-byte records:

```
SEEK,A,CC,6
SET SECTOR (omitted if device does not have RPS feature)
SRCH ID EQ,A+2,CC,5
TIC,*-8,0,0
RD or WRT,DATA1,CC+SILI,bytecount (bytecount to be between 1 and 4096,
                                   inclusive)
SEEK HEAD,B,CC,6                  (omitted if HEAD number unchanged)
SET SECTOR                        (omitted if device does not
                                   have RPS feature)
SRCH ID EQ,B+2,CC,5
TIC,*-8,0,0
RD or WRT,DATA2,SILI,bytecount    (bytecount to be between 1 and 4096,
                                   inclusive)

A      SEEK and SRCH arguments for first read/write
B      SEEK and SRCH arguments for second read/write
DATA1  data area for first read/write
DATA2  data area for second read/write
```

Note: Indirect addressing is not permitted.

DIAGNOSE Code X'20' – 370 Synchronous I/O for DIAGNOSE Support

Privilege Class: Any (370 Accommodation must be active)

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'20' to perform input/output (I/O) operations to tapes or direct access storage devices (DASDs), and dedicated unit record devices supported by z/VM. For a list of supported devices, see *z/VM: General Information*.

Note: z/VM includes DIAGNOSE code X'20' primarily for VM/SP, VM/SP HPO, and VM/ESA (370 Feature) compatibility. For new applications and new devices, use DIAGNOSE code X'A8'. DIAGNOSE code X'A8' can be used in all types of virtual machines.

Entry Values:

Rx
Contains the virtual device number.

Ry
Contains the channel address word (CAW). The leftmost bits, 0 through 3, of the CAW contain the protection key to use in accessing guest storage for the I/O operation.

Bit 4 (suspend control) must be zero.

Bits 5 through 7 (unassigned) must be zero.

Bits 8 through 31 contain the channel program address. Bits 29 through 31 must be zero, designating a doubleword-aligned channel program.

Usage Notes

1. Do not use DIAGNOSE code X'20' to read or write record-overflow-formatted data on DASD devices.
2. The channel program that you specify must not contain self-modifying CCWs.
3. CP provides error recovery for this DIAGNOSE instruction. The instruction does not complete until the I/O operation has completed, including any error recovery processing.
4. This DIAGNOSE code may be used only when the 370 Accommodation facility is active. Activate the 370 Accommodation facility with the SET 370ACCOM ON command.

5. Diagnose I/O operations issued to virtual Parallel Access Volume bases and aliases are randomly scheduled on any available, appropriate real base or alias device. Certain CCWs, such as Reserve and Release, require base or alias real device affinity. This is handled by CP as needed.
6. This DIAGNOSE code is limited to a minidisk size of 65520 cylinders. If a program issues DIAGNOSE code X'20' to a minidisk larger than 65520 cylinders, CP sets a condition code of 3 (cc=3) and places a return code of 13 in register 15.

Responses

Condition Codes: On return from DIAGNOSE code X'20' processing, CP sets the condition codes and the return codes as follows. In addition, a return code may be placed in register 15.

Condition Code	Return Code in Register 15	Meaning
0	Unchanged	I/O was completed with no errors; CSW is stored.
1	1 (X'01')	The device is either not attached, or is not accessible to the virtual machine.
1	5 (X'05')	The virtual device is busy or has an interrupt pending.
2	2 (X'02')	A unit exception is reflected in the unit status field; the CSW is stored.
2	3 (X'03')	An incorrect length is reflected in the channel status field; the CSW is stored.
3	13 (X'0D')	An unsupported device was specified, a permanent I/O error occurred, or a program issued this DIAGNOSE code to a minidisk larger than 65520 cylinders. If the user specifies an unsupported device, both the CSW and the user's Ry register is set to zero. If a permanent I/O error occurred, the CSW is stored and the user's Ry register contains the sense information in the following format: Byte 0 contains sense byte 2 Byte 1 contains sense byte 3 Byte 2 contains sense byte 0 Byte 3 contains sense byte 1

DIAGNOSE Code X'24' – Device Type and Features

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'24' to request identifying information and status information about a particular virtual device.

Note: DIAGNOSE code X'24' will no longer be upgraded for new device support. Applications using DIAGNOSE code X'24' should use DIAGNOSE code X'210' to take advantage of new device support. z/VM will continue to include DIAGNOSE code X'24' as a means of locating the address of the virtual console.

Entry Values:

Rx
Must contain one of the following:

- The virtual device number of the device for which information is requested
- The value negative 1 (–1). Specify this when the device is a virtual console whose device number is unknown to your virtual machine.

Ry
Not used.

Exit Values:

Rx
Contains information about a virtual console. Rx contains this data only if you specified –1 as an entry value and if CP located the console device.

Ry
Contains information about the specified virtual device.

Ry+1
Contains information about the real device that is associated with the specified virtual device. However, if Ry has been specified as register 15, CP returns only virtual device information; no information is returned in register Ry+1.

Usage Notes

1. A CMS application may determine which set of ASCII and APL tables is currently in use with DIAGNOSE code X'24'. The real device terminal code field returned in Rx is set to one of the following values:

   Value
   Meaning

   X'10'
   ASCII mode (TERMINAL APL OFF)

   X'14'
   ASCII/APL, SI state (TERMINAL APL ON, using standard ASCII)

   X'18'
   ASCII/APL, SO state (TERMINAL APL ON, using ASCII/APL)
2. For a DIAGNOSE code X'24' to a SNA device connected through VCNA, the real device model number information is correct; however, the real device type may not be reliable.
3. This DIAGNOSE code returns device information in the same format as you would receive with CP. It returns the real device type class and real device type fields as **CLASSPEC** (X'02') and **TYPUNSUP** (X'01') for all 3390 or newer DASD. For all new device information, you should use DIAGNOSE code X'210'. For more information, see Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969.

   The following tables summarize the type of information that is returned to the Rx, Ry, and Ry+1 registers after the instruction has executed.

   Not all of the bits that are significant in VM/SP, VM/SP HPO, and VM/ESA (370 Feature) remain significant in z/VM.
4. If the virtual device is a virtual disk in storage, it is not mapped to a real device. However, condition code 0 is returned for successful completion, and register Ry+1 contains information about the simulated real device.
5. When issued for a virtual console and the user is logged on to the system console, CP will return real device information indicating that the real device is an undefined line mode terminal.

Rx Information

Byte 0	Byte 1	Byte 2	Byte 3
real device terminal code for a local virtual console		virtual device number	

See Appendix A, "Data Areas Used by DIAGNOSE Codes," on page 969 for the meaning of the values returned in the real device terminal code field.

Ry Information

Byte 0	Byte 1	Byte 2	Byte 3
virtual device type class	virtual device type	virtual device status	virtual device flags

See Appendix A, "Data Areas Used by DIAGNOSE Codes," on page 969 for the meanings of values returned in these fields.

Ry+1 Information

Byte 0	Byte 1	Byte 2	Byte 3
real device type class	real device type	real device model number	real device feature code —or— current device line length for a local virtual console

See Appendix A, "Data Areas Used by DIAGNOSE Codes," on page 969 for the meaning of the values returned in these fields.

Responses

Condition Codes: The following chart lists the condition codes CP can return for DIAGNOSE code X'24', the meaning of each condition code, and the registers where data is returned.

Condition Code	Status	Rx Contains[1]	Ry Contains[2]	Ry+1 Contains
0	Successful completion	Virtual console information	Virtual device information	Real device information
1	Undefined			
2	Virtual device exists but is not associated with a real device	Virtual console information	Virtual device information	
3	Invalid device number, **or** the virtual device does not exist			

Note:

[1]The Rx register contains information only when DIAGNOSE code X'24' specifies a virtual console whose address is unknown.

[2]If Ry is register 15, CP returns only virtual device information; no information is returned in register Ry +1.

DIAGNOSE Code X'28' – Dynamic Channel Program Modification

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'28' to modify selected instructions in a channel program after I/O is initiated and before the operation is completed. The channel command words (CCWs) that you can modify are:

- From a Transfer in Channel (TIC) to a No Operation (NOP)
- From a TIC CCW to point to a new list of CCWs
- From a (No Operation) NOP to a TIC CCW
- The fields of a NOP CCW.

In an XC virtual machine, DIAGNOSE code X'28' can be run in access register mode. It addresses only the virtual machine's host-primary address space.

Entry Values:

Rx
Contains the guest absolute address of the transfer in channel (TIC) or No Operation (NOP) channel command word that was modified by your virtual machine. This channel command word is always in the host-primary address space.

Ry
Contains the subchannel number in bits 16 through 31. This device must be a TERMINAL or GRAPHICS class device, a 3704, or a 37x5.

When executed with the 370 Accommodation facility active (the SET 370ACCOM ON command has been issued), the Ry value is interpreted according to the following heuristic: If the Ry value, when viewed as a System/370 device address, selects a device which exists, and at least one of the following is true, then the Ry value is interpreted as a device address; otherwise, the Ry value is interpreted as a subchannel number.

- There is an active channel program on the device with that device address, and the channel program was started by a System/370 I/O instruction (e.g., SIO or SIOF).
- The Ry value, when viewed as a 370-XA subchannel number, does not select a subchannel with a device assigned to it.
- The Ry value, when viewed as a 370-XA subchannel number, selects a subchannel with a device assigned to it, but there is no active channel program on that device.

Note: Rx and Ry cannot be the same register.

Usage Notes

1. If the AUTOPOLL function has been set on, the operating system running in your virtual machine must use DIAGNOSE code X'28' to notify z/VM whenever a virtual autopolling channel program has been modified. For more information, see the SET AUTOPOLL command in the *z/VM: CP Commands and Utilities Reference*.
2. SUSPEND/RESUME is not supported using DIAGNOSE code X'28'. A request to modify a TIC/NOP CCW in a:
   - Suspended channel program results in a condition code of 2 and a return code of 9.
   - Currently executing channel program with a CCW containing a suspend bit equal to 1, or on, is honored; the suspend bit is reset in the modified channel program.
3. The address specified in the Rx register, the new address in the modified TIC CCW, and the new CCW list to which the modified TIC CCW points must all be addresses that appear second-level; CP knows these addresses are virtual, but the virtual machine treats them as absolute.
4. When a virtual machine modifies a TIC CCW, it is modifying a virtual channel program. CP has already translated that channel program, and the real channel program may or may not have been completed for the guest virtual machine. DIAGNOSE code X'28' must be issued to inform CP of the change in the virtual channel program so that CP can make the corresponding change in the real CCW before it is executed, if possible. If the real CCW has already been executed, the change has no effect unless it is reexecuted during the running of the channel program.
5. When a NOP or TIC CCW is modified to point to a new list of CCWs, CP translates the new CCWs.

Responses

Condition Codes: On return from DIAGNOSE code X'28' processing, CP sets the condition code listed as follows: In addition, a return code is placed in register 15:

Condition Code	Return Code in Register 15	Meaning
0	0 (X'00')	The real channel program is successfully modified.
1	1 (X'01')	The same register is specified for Rx and Ry.
1	2 (X'02')	The device specified by the Ry register is not found.
1	3 (X'03')	The address specified by the Rx register is not within the user's storage space.
1	4 (X'04')	The address specified by the Rx register is not doubleword-aligned.
1	5 (X'05')	A CCW string corresponding to the device (Ry) and address (Rx) specified is not found.
1	6 (X'06')	The CCW at the address specified by the Rx register is neither a TIC nor a NOP; the CCW in the channel program is neither a TIC nor a NOP.
1	7 (X'07')	The new address in the modified TIC CCW is not within the user's storage space.
1	8 (X'08')	The new address in the modified TIC CCW is not doubleword-aligned.
1	10 (X'0A')	The device specified in Ry is not an eligible type.
1	14 (X'0E')	The target CCW is in the middle of a data chained region.
1	16 (X'10')	A NOP or TIC CCW has been modified to a TIC; the new CCW data address of the TIC target CCW is not translated, and data chaining ID active.
2	9 (X'09')	The channel program could not be modified; channel end and/or device end occurred already.
2	12 (X'0C')	The channel program could not be modified; a paging error has occurred when an attempt was being made to derive the real storage address for the CCW pointed to by the modified TIC CCW.

DIAGNOSE Code X'34' – Read System Dump Spool File

Privilege Class: C, E

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'34' to read the system dump spool file.

Entry Values:

Rx

Contains the guest real address of a page-size buffer that receives the spool file data. If the buffer crosses a page boundary, a specification program exception is returned to your virtual machine.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer that receives the spool file data. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
Contains the virtual address of the spool input card reader. Ry must not be register 15.

Exit Values: On return, Ry+1 may contain a return code. See the Responses section for a description of these codes.

Usage Notes

1. The format of a z/VM dump may change from release to release.
2. The format of the dump data is not part of the programming interface.
3. You may not be authorized to issue this DIAGNOSE code if an external security manager is installed on your system. For additional information, contact your security administrator.

Responses

Condition Codes: On return from the DIAGNOSE code X'34' processing, CP sets one of the condition codes listed below. In addition, a return code is placed in Ry+1, as follows:

Condition Code	Return Code in Ry +1	Meaning
0	None	Data transfer completed successfully
1	None	End-of-file reached
2	None	File not found
3	4 (X'04')	Device address invalid
3	8 (X'08')	Device type invalid
3	12 (X'0C')	Device busy
3	16 (X'10')	Fatal paging I/O error
3	20 (X'14')	Invalid file

Program Exceptions: These program exceptions can occur if the DIAGNOSE X'34' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Any of the following: • The address contained in Rx is not on a page boundary. • Ry is register 15. • Privilege class violations.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store into the result buffer.

DIAGNOSE Code X'3C' – Activate z/VM CP Directory

Privilege Class: A, B, C

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'3C' to activate or reactivate a valid directory found in the CP-owned volume list. This is usually done at CP initialization time, or as a result of the DIRECTXA command rebuild of the current z/VM CP directory.

Entry Values:

Rx
Contains the first four bytes of the volume identification of the volume assumed to contain a valid directory.

Ry
Contains the last two bytes of the volume identification (in bytes 0 and 1), and the number of the virtual device to which the volume is attached (in bytes 2 and 3).

Usage Notes

DIAGNOSE code X'3C' activates the z/VM directory residing on the specified volume if one of the following is true:

1. The named volume is the one on which the current CP directory was found at CP initialization time
2. No directory is currently active, and the named volume is currently CP-owned.

Responses

Condition Codes: On return from DIAGNOSE code X'3C', CP sets one of the following condition codes:

Condition Code	Meaning
0	z/VM has successfully activated the CP directory.
1	The directory that was activated is not a current CP directory.
2	The DASD address specified in the volume label was invalid. The volume label and allocation map should be restored to the values that existed prior to the update.
3	A fatal I/O error occurred when z/VM attempted to read the CP directory. The volume label and allocation map should be restored to the values that existed prior to the update.

Program Exceptions: These program exceptions can occur if the DIAGNOSE X'3C' is given incorrect input data:

Problem Encountered	Cause
Specification exception	If Rx is unchanged: The update is not for the current online directory volume and the issuer of the DIAGNOSE X'3C' does not have the appropriate privilege class. If Rx=0: The update is for the current online directory volume and one or more of the following is true: • The issuer of the DIAGNOSE X'3C' does not have the appropriate privilege class. • The new directory is not SSI-enabled, which is not valid in the current system configuration. • The new directory is SSI-enabled, which is not valid in the current system configuration. Rx should be tested, and if it is zero, the volume label and allocation map should be restored by the function that issued the DIAGNOSE X'3C' to the values that existed prior to the update.
Privileged-operation exception	The virtual machine is in the problem state.

DIAGNOSE Code X'44' – Voluntary Time Slice End

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'44' to notify the scheduler that a spin lock loop exists in your virtual machine. DIAGNOSE code X'44' informs the scheduler that the remainder of the CPU time slice allocated to a virtual CPU is no longer useful. z/OS® and z/VM operating systems use DIAGNOSE code X'44' as the standard method for notifying the scheduler that a spin lock loop exists.

Entry Values: Specify the Rx and Ry fields as zero.

Usage Notes

1. The virtual CPU that issues DIAGNOSE code X'44' is given a dispatching priority lower than other virtual CPUs in the same guest multiprocessing configuration. The other virtual CPUs generally run before the issuing virtual CPU. If there is no other virtual CPU for this virtual machine, then DIAGNOSE code X'44' has no effect.
2. DIAGNOSE code X'44' is useful when the operating system on one guest CPU is waiting to obtain a spin lock, and another guest CPU must be run in order to release the spin lock. The DIAGNOSE causes the CPU holding the spin lock to be run before the issuing CPU runs again.
3. The effects of DIAGNOSE code X'44' are temporary. Within a few seconds, the virtual CPU is scheduled as if the DIAGNOSE were never issued.

Responses

None.

DIAGNOSE Code X'48' – Second Level SVC 76

Addressing Mode: 24-bit or 31-bit

Execution of DIAGNOSE code X'48' provides a null function in z/VM. z/VM supports DIAGNOSE code X'48' only to provide compatibility with VM/SP, VM/SP HPO, and VM/ESA (370 Feature).

In VM/SP, VM/SP HPO, and VM/ESA (370 Feature), the function of DIAGNOSE code X'48' is to distinguish between two or more levels of virtual device numbers in EREP records. In z/VM, CP itself provides this function; you do not have to invoke it using the DIAGNOSE instruction.

If your virtual machine is an XC virtual machine, then DIAGNOSE code X'48' is not valid. A specification exception is recognized if DIAGNOSE code X'48' is attempted from an XC virtual machine.

DIAGNOSE Code X'4C' – Generate Accounting Records

Privilege Class: Any, with the directory ACCT option.

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'4C' to generate accounting records with existing accounting information and to change user accounting information for the next accounting record in the current session. Your virtual machine can issue this DIAGNOSE only if the account (ACCT) option has been specified for you in the system directory.

Entry Values:

Rx

Contains the guest real address of one of the following:

- A 24-byte parameter list that identifies the accounting information for the next *charge-to* user. The contents of the parameter list depend on the hexadecimal function subcode placed in Ry. In the list below, the hexadecimal numbers are offsets. The format of the parameter list is:

  X'00'—the next *charge-to* user ID
  X'08'—the next *charge-to* account number
  X'10'—the next distribution code

  If the address represents a parameter list, the list must be doubleword-aligned.

  If the address of the parameter list is zero, no parameter list is used. Instead, the next *charge-to* user ID, account number, and distribution code are taken from the directory entry of the user issuing the DIAGNOSE instruction.
- A variable length data area, up to 70 bytes, that you want to store in an accounting record.

CP interprets the address, based on a hexadecimal function code that is supplied in Ry.

Ax

Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the data area addressed by Rx. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the data area is in the host-primary address space.

Ry

Ry contains a hexadecimal function subcode. In z/Architecture mode the high order word of register Ry is ignored. The subcode is put in the low order byte.

Subcode
Rx points to:

X'0000'
A parameter list containing only a user ID

X'0004'
A parameter list containing a user ID and account number

X'0008'
A parameter list containing a user ID and distribution code

X'000C'
A parameter list containing a user ID, account number, and distribution code

X'0010'
A data area containing up to 70 bytes of user information to be transferred to the accounting record, starting in column 9

Note: If Ry contains X'0010', Ry cannot be register 15.

Ry+1
If Ry contains X'0010', then Ry+1 must contain the length of the data area that Rx is pointing to. In z/Architecture mode the high order word of register Ry+1 is ignored. The 4 bytes of the low order register word contains the length of the buffer. The length specified in Ry+1 must be greater than zero and less than or equal to 70.

If Ry does not contain X'0010', Ry+1 is ignored.

Usage Notes

1. For a discussion of how z/VM processes addresses, refer to "How Addresses Are Processed" on page 5.
2. If Ry contains any valid function subcode other than X'0010':
   - Accounting records are generated for the virtual machine's resources, dedicated devices, temporary disk space, virtual disk in storage, and network data transmission records using the accounting information currently set.

     The accounting information currently set are the values set with the previous DIAGNOSE X'4C'. If no DIAGNOSE code X'4C' has been issued yet, the user ID, account number, and distribution code are set by logon to the values specified in the user directory.

     Columns 79 and 80 of the accounting records generated after DIAGNOSE code X'4C' has been issued have an accounting record identification code of:

     C'C1'—the virtual machine resource records
     C'C2'—the virtual machine dedicated device records
     C'C3'—the virtual machine temporary disk space records
     C'CB'—the virtual machine virtual disk in storage records
     C'CC'—the virtual machine network data transmission records

     For more information about accounting records, see *z/VM: CP Planning and Administration*.
   - The parameter list (a 24-byte area whose guest real address is in Rx) is validated:
     – If the address is invalid, an addressing exception is generated.
     – If the address is not aligned on a doubleword boundary, a specification exception is generated.
     – If the address in Rx is zero, that is, no parameter list exists, the accounting information is reset to equal that of the issuing user ID as set in its directory entry (that is, your user ID, your account number, and your distribution code).

       Control is returned to your virtual machine with a condition code of zero.
     – If the address in Rx is valid, that is, it points to a parameter list:
       - And if the new user ID specified is invalid, that is, it cannot be found in the system directory:

         The accounting information is reset to equal that of the issuing user ID as set in its directory entry (that is, your user ID, your account number, and your distribution code).
         Control is returned to your virtual machine with a condition code of two.
       - And if the new user ID specified is valid, the accounting information is updated from the parameter list pointed to, based on the hexadecimal function subcode set in Ry; and control is returned to your virtual machine with a condition code of zero:

Subcode
Accounting Information Changed

X'0000'
The next *charge-to* user ID updated from parameter list.

The next *charge-to* account number updated from the new user ID system directory entry

The next distribution code updated from the new user ID system directory entry

X'0004'
The next *charge-to* user ID updated from the parameter list

The next *charge-to* account number updated from the parameter list

The next distribution code updated from the new user ID system directory entry

X'0008'
The next *charge-to* user ID updated from the parameter list

The next *charge-to* account number updated from the new user ID system directory entry

The next distribution code updated from the parameter list

X'000C'
The next *charge-to* user ID updated from the parameter list

The next *charge-to* account code updated from the parameter list

The next distribution code updated from parameter list

X'0010'
Not a valid parameter list function subcode (see usage note “3” on page 49)

3. If Rx contains the guest real address of a variable length data area, and Ry contains a function subcode of X'0010':
   - If the address specified in Rx is negative or greater than your virtual machine's storage size, an addressing exception is generated.
   - If the value in Ry+1 is zero, negative, or greater than 70, a specification exception is generated.
   - If both the address and the length are valid, an account buffer is built as follows:
     – The issuing user ID as set at logon (that is, your user ID), is placed in columns 1 through 8.
     – An accounting record identification code of *C0* is placed in columns 79 and 80.
     – The data, pointed to by the address in Rx, is moved into the accounting record starting at column 9, for a length equal to the value in Ry+1.
     – Unused columns are initialized to blanks.
     – The accounting buffer is sent to the accounting virtual machine.
     – Control is returned to your virtual machine with a condition code of zero.
4. For subcodes other than X'0010', the counter containing the number of LINK commands attempted with an invalid password is reset.

Responses

Condition Codes: On return from the DIAGNOSE code X'4C', CP sets one of the following condition codes:

Condition Code	Meaning
0	Both the user ID and hexadecimal function subcode are valid. Accounting records are generated using the currently set accounting information, and the accounting information for the next accounting records for this session is updated from the parameter list, if one was specified.

Condition Code	Meaning
1	The user is not authorized to use the account option.
2	Either of the following: • Your own user ID is no longer in the CP directory. Accounting records have been written, but the accounting information (user ID, account number, distribution code) has not been reset. • The user ID contained in the parameter list is not found in the system directory. Accounting records have been written, and the accounting information has been reset to your user ID.
3	The hexadecimal function subcode in Ry is invalid.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'4C' is given incorrect input data:

Problem Encountered	Cause
Specification exception[1]	Any of the following: • Subcodes X'0000', X'0004', X'0008', X'000C': the parameter list does not start on a doubleword boundary. • Subcode X'0010': Ry is register 15. • Subcode X'0010': the value in Ry+1 is less than 1 or greater than 70.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store into the parameter list or the data area.

Note: [1] Ordinarily, specification and some access exceptions are suppressing or nullifying, that is, the instruction is effectively not executed. However, when these exceptions occur for subcodes X'0000', X'0004', X'0008', or X'000C', of DIAGNOSE code X'4C', accounting records are written for the preceding interval and accounting information is reset to your own user ID, accounting number, and distribution code.

DIAGNOSE Code X'54' – Control the Function of the PA2 Key

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'54' to control the function of the PA2 function key. You can use the PA2 key to simulate an external interrupt to a virtual machine or to clear the output area of a display screen.

Entry Values:

Rx
Contains a value that controls the function of the PA2 key. If Rx contains a nonzero value, the PA2 key simulates an external interrupt to the virtual machine. If Rx contains a zero, the PA2 key clears the output area of the display screen.

Ry
Not used.

Usage Note

The external interrupt is simulated only when the display screen is in the VM READ, HOLD, or MORE status and the CP TERMINAL APL ON command has been issued.

Responses

None.

DIAGNOSE Code X'58' – 3270 Virtual Console Interface

Privilege Class: Any

Addressing Mode: 31-bit

Use DIAGNOSE code X'58' to enable your virtual machine to communicate with IBM 3270 display stations.

DIAGNOSE code X'58' operates in all virtual machines; however, only format 0 CCWs are allowed. All storage addresses referred to by DIAGNOSE code X'58' are guest absolute addresses and all designated areas are in host-primary space. Addresses of CCWs and the addresses contained in the CCW, including the pointer to indirect address words (IDAWs) as well as directly referenced data buffers, are 24-bit addresses. DIAGNOSE code X'58' may be used in one of two modes:

- Line mode–the user can write data to the screen, which is in the control of CP and is formatted into specific areas: input, output, and status. The user can write data to either of the following places:
  - The input or output area, beginning at a specified line. Up to a full-screen of data can be written.
  - The output area starting at the next available line. More than a full-screen of data can be written.
- Full screen mode–the user has full control over the format of the screen, and is responsible for error checking and recovery. The user may read from, as well as write to, the display.

In either mode, the virtual machine can provide data stream orders along with data that is sent to the display station. An attribute character provides control information for the display (an example of control information is a request to intensify data when it is displayed).

Entry Values:

Rx
Contains the channel address word (CAW). The leftmost bits, 0 through 3, of the CAW contain the protection key to use in accessing guest absolute storage for the I/O operation.

Bit 4 (Suspend Control) must be zero.

Bits 5 through 7 (Unassigned) must be zero.

Bits 8 through 31 contain the channel program address.

Bits 29 through 31 must be zero, designating a doubleword aligned channel program.

Ry
Contains the device number of the display station where the operation is to be performed. This value must be right-justified.

For 370-only applications running in an XA, ESA, or XC virtual machine with 370 Accommodation enabled (see the SET 370ACCOM command in the *z/VM: CP Commands and Utilities Reference*), DIAGNOSE code X'58' follows the architecture for the Start I/O (SIO) instruction processing. For more information on the SIO instruction and the status returned, see the *Enterprise Systems Architecture/390 Principles of Operation*. The guest virtual machine's condition codes (CCs) are changed on the following conditions:

Code
Description

0
The SIO function has been accepted

1
The CSW has been stored

2
The channel or subchannel is busy

3
Not operational.

For ESA applications running in an XA, ESA, XC, or Z virtual machine (including one in access register mode), DIAGNOSE code X'58' follows the interface architecture for the Start Subchannel (SSCH) instruction processing. For more information on the SSCH instruction and the status returned, see the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf), *z/VM: Enterprise Systems Architecture/Extended Configuration Principles of Operation*, or z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), based on the architecture mode of the virtual machine. The guest virtual machine's condition codes are changed on the following conditions:

Code
Description

0
The start function is initiated

1
The subchannel is status-pending

2
The subchannel is busy

3
Not operational.

Usage Notes

1. In an XC virtual machine, DIAGNOSE code X'58' can run in access register mode, but it addresses only the virtual machine's host-primary address space.
2. When a virtual machine issues DIAGNOSE code X'58', it must provide one or more channel command words (CCWs). These CCWs specify the operation to be performed, provide control information for the display station, and specify the address of data to be displayed during a write operation or the address of a buffer where data is to be stored during a read operation. As is usual for I/O operation, storage accesses to CCWs, IDAWs, and data areas are not subject to low-address protection, fetch-protection override, or storage-protection override. Key-controlled protection applies, using the key in Rx bits 0-3. See also the usage note (“8” on page 56) for valid channel programs.
3. When invoking CCW code X'49', both APL and TEXT must be off. Having either APL or TEXT on causes command rejects.
4. Issuing CCW code X'49' from a device other than a 3270 causes command rejects.
5. DIAGNOSE code X'58' is for a virtual 3215 console and causes command rejects if executed with CONMODE 3270.
6. When DIAGNOSE code X'58' is being executed in line mode:
   - To display up to a full screen of data starting at a specified line, code a CCW using the following assembler language instructions:

     ```
     DS 0D
     DC X'19',AL3(dataddr),AL1(flags),AL1(ctl),AL2(count)
     ```

     X'19'
     is the subcode.

     dataddr
     is the virtual storage address of the first byte of data to be displayed.

flags
are standard CCW flags. The suppress-incorrect-length indicator, bit 34, must be set to a value of 1. Set other bits as needed. To use 31-bit addresses, the use of indirect data address words is required.

ctl
is a control byte defined as follows:

- If the high-order bit (bit 0) is on, CP erases the display station screen before new data is displayed.
- Bits 2 through 7 identify the line on the display screen where the display is to start. A value of 0 (B'*xx*00 0000') corresponds to the first or top line, a value of 1 (B'*xx*00 0001') corresponds to the second line, and so forth.
- If the control byte contains the value X'FF', CP erases the display station's output area. No new data is displayed.
- If the control byte contains the value X'FE', CP:
  - Erases the entire screen
  - Rewrites the field attribute bytes for the CP screen format
  - Resets the cursor to the beginning of the input area.

count
specifies the number of bytes of data to be displayed. The maximum value allowed is 3991 bytes. The maximum amount of data that can be displayed at one time depends upon the screen size of the 3270 display station, the line on the screen where the data is to start, and the character set used.

For APL and TEXT character sets, each character that is translated into a compound character counts as two bytes. Therefore, the maximum count of data is 1995. For APL and TEXT data streams that contain some untranslated compound character data, the maximum count lies somewhere between 1995 and 3991. If an APL or TEXT data stream is sent that, after translation, exceeds 3991 bytes, the data is truncated.

For the default or EBCDIC character set:

- A model 2 can display in an area that extends from:
  - Lines 1 through 22 (a maximum of 1760 bytes)
  - Line 23 up to and including the left 59 character positions of line 24 (a maximum of 139 bytes).
- A model 3 can display in an area that extends from:
  - Lines 1 through 30 (a maximum of 2400 bytes)
  - Line 31 up to and including the left 59 character positions of line 32 (a maximum of 139 bytes).
- A model 4 can display in an area that extends from:
  - Lines 1 through 41 (a maximum of 3280 bytes)
  - Line 42 up to and including the left 59 character positions of line 43 (a maximum of 139 bytes).
- A model 5 can display in an area that extends from:
  - Lines 1 through 25 (a maximum of 3300 bytes)
  - Line 26 up to and including the left 111 character positions of line 27 (a maximum of 243 bytes).

  The above lengths are dependent on hardware characteristics, not on DIAGNOSE code X'58' interface limitations.

- To display data starting at the next available line on the screen, code a CCW using the following assembler language instructions:

```
DS 0D
DC X'49',AL3(dataddr),AL1(flags),AL1(ctl),AL2(count)
```

X'49'
is the subcode.

dataddr
is the virtual storage address of the first byte of data to be displayed.

flags
are standard CCW flags.

ctl
is ignored.

count
specifies the number of bytes of data to be displayed. The maximum value allowed is 3991 bytes.

When the virtual machine issues CCW code X'49', CP treats the channel program as a normal 3215 channel program, with the following exceptions:

– The data stream is processed as if TERMINAL LINESIZE OFF has been issued. That is, the data stream is broken up only when a X'15' is encountered in the data stream.
– Each time a X'15' is encountered, CP counts exactly one line.
– CP does not translate any code point from X'40' to X'FE', inclusive. CP also does not translate code points X'0E' and X'0F'.

DIAGNOSE code X'8C' should be issued before DIAGNOSE code X'58' and code X'49' to determine the width of the screen. This information should then be used to determine the number of character positions that can be taken up between occurrences of X'15'. The width of the screen minus 2 should be the maximum for that number. If this restriction is not observed, CP cannot manage the screen correctly.

To provide attribute characters for the data, place the attribute character in the data stream immediately following a 3270 start-field order. The start-field order, a 1-byte value, notifies the 3270 display station that the next byte in the data stream is an attribute character. For a description of how the 3270 display station uses attribute characters, and to determine the values to specify for attribute characters and the start-field order, see the *IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide*, GA23-0061. This is a requirement of the hardware, not of the DIAGNOSE code X'58' interface.

Note: Through the attribute character, it is possible to define a display field as selector-pen detectable. However, when the selector pen selects the field, CP does not return data from the field to the virtual machine. This is a requirement of the hardware, not of the DIAGNOSE code X'58' interface.

When DIAGNOSE X'58' is specified, any start-field order in the data stream has its associated attribute byte set with the modified data tag off and the protection bit on, regardless of previous settings.

After processing DIAGNOSE code X'58', CP sets a condition code. If the operation is successful (no I/O errors occurred), CP sets a condition code of zero. If an I/O error occurs, CP sets a nonzero condition code.

For 370-only applications running in an XA, ESA, or XC virtual machine with 370 Accommodation enabled, the returned condition codes and CSW status are the standard condition codes and status as defined in the . The application program is responsible for all I/O status and error checking, just as if START I/O (SIO) were being used instead of DIAGNOSE. This is done by using the TEST I/O (TIO) instruction and examining the returned condition code and the virtual CSW.

For ESA applications running in XA, ESA, XC, or Z virtual machines, the application program is responsible for all I/O status and error checking, just as if a Start Subchannel (SSCH) were being used instead of DIAGNOSE code X'58'. This is done by using the Test Subchannel (TSCH) instruction and by examining the subsequent IRB. For more information on the IRB and the status returned, see the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf), *z/VM: ESA/XC Principles of Operation*, or z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), based on the architecture mode of the virtual machine.

7. When DIAGNOSE code X'58' is being executed in full-screen mode:

   DIAGNOSE X'58' provides a means by which a virtual machine may share, with CP, control of a 3270 display station. Two CCW operations, X'29' and X'2A', in addition to performing the requested I/O, also notify CP that the display station is operating under the control of the virtual machine.

   CCW code X'29' performs a Write, Erase/Write, Erase/Write Alternate, or Write Structured Field operation, depending on the value of the control field. The virtual machine must provide appropriate control information beginning with the Write Control Character (WCC) and including 3270 orders following the WCC. Data may be written anywhere on the screen. The virtual machine must provide the address where the write is to begin; it uses a Set Buffer Address (SBA) command to do this. Writing can also start at the current cursor address. CCW code X'29' performs a Write Structured Field operation when the value of the control field is X'20'. The Write Structured Field command sends control information to a 3274 controller.

   CCW code X'2A' performs a Read Buffer or a Read Modified operation, depending on the value of the control field.

   To specify the full-screen mode CCW, use the following assembler language instructions:

```
DC 0D
DC X'(ccwcode)',AL3(dataddr),AL1(flags),AL1(control),AL2(count)
```

   ccwcode
   is a 2-digit CCW code (X'29' or X'2A')

   dataddr
   for a write operation, specifies the first byte of the data stream (WCC) to be written. For a read operation, specifies the address of the read buffer.

   flags
   is the standard CCW flag field

   control
   for a write operation (CCW code X'29') the following control field values cause various operations to be performed:

   Value
   Operation Performed

   X'80'
   Erase/Write

   X'C0'
   Erase/Write Alternate

   X'40'
   Erase/Write Alternate

   X'20'
   Write Structured Field

   All other values
   Write

   For a read operation (CCW code of X'2A'), the following control field values cause various operations to be performed:

Value
Operation Performed

X'80'
Read Modified

All other values
Read Buffer

By adding X'10' to the CONTROL field values for Erase/Write or Erase/Write Alternate, making them X'90' or X'D0' respectively, the break key interrupt is reflected to the virtual machine. This replaces the normal break key function of returning the virtual machine to CP mode, and allows a virtual machine to have full control of the keyboard.

Normal break key function is restored when either full screen mode is reset or a DIAGNOSE code X'58' CCW code X'29' Erase/Write (control code X'80') or Erase/Write Alternate (control code X'C0' or X'40') is issued.

count
for a write operation, specifies the number of bytes to be displayed in addition to the number of bytes of control information. For a read operation, it specifies the number of display characters to be read in addition to the number of bytes of control information.

If you are using a local terminal (whose controller is connected to the host computer through a direct channel) the maximum count is 65,535; otherwise, the maximum count is 65,503. The maximum number of displayable positions (a hardware dependency, not a DIAGNOSE code X'58' requirement) for the supported devices is:

Max. Displayable
Supported Devices

1920 bytes
3277, 3278, and 3279 Model 2

2560 bytes
3278 and 3279 Model 3

3440 bytes
3278 Model 4

3564 bytes
3278 Model 5

8. For valid channel programs:

   The channel programs presented by DIAGNOSE code X'58' may contain one or more CCWs. The CCW operation codes supported are the DIAGNOSE code X'58' operation codes X'19', X'49', X'29', and X'2A', and the two general operation codes, X'08' (TIC) and X'03' (NOP). The channel program must be either line mode or full-screen mode; the two modes cannot be mixed in one channel program. The X'19' operation code that clears the screen may be used to start either a line mode or full-screen mode write operation, or it may comprise a whole channel program in itself.

   The following rules apply:

   a. The channel program **must** begin with a valid DIAGNOSE code X'58' operation code: X'19', X'49', X'29', or X'2A'.
   b. A X'19' or X'49' write can be command-chained to another X'19', a X'49', a X'08' (TIC), or a X'03' (NOP operation code).
   c. A X'19' with a CTL value of X'FF' or X'FE' can be chained to any other X'19', X'49', or X'29' operation code, or to an intervening X'08' (TIC) or X'03' (NOP) operation code. In the case of a X'29' operation code, the X'19' **must** be the first CCW in the channel program.
   d. A X'29' or X'2A' operation code can be chained to another X'29', a X'2A', a X'08' (TIC), or a X'03' (NOP) operation code.

9. For full-screen interactions (for both DIAGNOSE code X'58' and 3270 SIO/SSCH):

The virtual machine console operates in one of two modes: CP or full-screen. CP mode is the default and is indicated by the screen status field in the lower right-hand corner of the screen. When in CP mode, CP controls the screen format, and the data that appears on the screen is provided by CP and the programs running in the virtual machine.

A guest virtual machine can use either DIAGNOSE code X'58' or the 3270 SIO/SSCH instruction to initiate full-screen mode, but not both. Full screen console support enables a guest virtual machine and CP to share a locally-attached display station controlled by CP. The virtual machine can use the display station in full-screen mode; CP can use the same display station as a device in line mode.

When in full-screen mode, the screen format data checking and error checking are under complete control of the program running in the virtual machine.

Line mode is terminated and full screen mode is initiated when the application program issues an Erase/Write, Erase/Write Alternate, or Write Structured Field instruction. Full screen mode may be terminated by a line mode type I/O to the screen anytime the keyboard is in a locked state.

The Terminal BRKKEY command allows the user to specify a PF key, the Clear key, or the PA2 key, as well as the PA1 key, as the break key in full-screen mode; it also allows BRKKEY NONE to be specified. The default break key is PA1. When you press the user-defined break key while in full-screen mode, z/VM puts your virtual machine in CP mode and displays *CP READ* in your status area. When PA1 is not defined as the BRKKEY, PA1 attentions are sent to the virtual machine. Some application programs that could be running in the virtual machine may interpret the PA1 attention as a user request to enter the CP environment. For a further explanation of the use of the PA1 key, see usage note “10” on page 57 (*DIAGNOSE code X'58' full-screen I/O*).

In full-screen mode, all CP messages are queued. The entire queue of CP messages is processed after each of the following operations:

a. A full-screen read operation (any read operation locks the keyboard)

b. A full-screen write operation (that locks the keyboard)

c. The expiration of a 60-second timer for CP priority messages.

If a priority CP message (such as a warning message from the system operator) is to be displayed while in full-screen mode, z/VM posts an attention interruption to the application program, and a 60-second timer is set. The attention interrupt informs the application program that a read operation should be initiated. In general, if the application program does not issue a read request before the 60 seconds have expired, CP erases the screen and displays all queued messages. However, if the application program has issued a full-screen support Write Structured Field instruction, CP does not take over the screen.

Other non-full-screen messages are displayed immediately by z/VM when in full-screen mode.

Interactions between CP and the application program in the virtual machine using full-screen support are in the *IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide*, GA23-0061. The application programmer must be familiar with the operation of the IBM 3270 display station.

If Terminal Breakin GuestCTL has been specified, CP is allowed to break in when the break key is used. An audible alarm is sounded when CP messages are queued. Priority CP messages and DIAGNOSE code X'08' output can still break in and take over the full screen.

Other non-full-screen messages are displayed immediately by z/VM when in full-screen mode (except for those exceptions noted for the Terminal Breakin command).

A full-screen Erase/Write, Erase/Write Alternate, or Write Structure Field operation establishes full-screen mode.

10. For DIAGNOSE code X'58' full-screen I/O:

    Listed below are general programming considerations that must be followed to effectively use the DIAGNOSE code X'58' instruction for full-screen I/O:

    a. A full-screen Erase/Write or Erase/Write Alternate operation establishes full-screen mode.

b. When a mode switch has occurred and the screen is in CP mode, the application program is notified by an X'8E' in the CSW/IRB unit status byte following a full-screen I/O operation. An Erase/Write, Erase/Write Alternate, or Write Structure Field operation instruction should be issued to reestablish full-screen mode and reformat the screen.

   An X'8E' in the CSW/IRB unit status byte following an Erase/Write or Erase/Write Alternate instruction indicates that non-full-screen data (CP mode) is waiting to be read. The application program should issue a non-full-screen Read and then reissue the Erase/Write instruction.

c. The application program must establish an environment to handle attention interruptions. This could be done using the CMS macros HNDINT and WAITD. CP posts an attention interruption to the application program in one of the following conditions:

   1) CP receives an attention interruption indicating that the virtual machine console operator has caused an interruption (for example, when the Enter key or a PF key is pressed)
   2) A CP priority message is to be displayed.

d. If the test request key is pressed at a local 3270 when in full-screen mode, X'604040' is returned to the application program in the read buffer.

e. For break key operation in full-screen mode:

   1) If a 3270 SIO/SSCH or DIAGNOSE code X'58' with bit X'10' of the control option is not set:

      If the break key is pressed, CP posts an attention interrupt to the virtual machine. If the virtual machine responds with a READ, or the break key is pressed a second time, the virtual machine is put in line mode and a CP READ is displayed on the screen's status area.

   2) If DIAGNOSE code X'58' with bit X'10' of the control option is set:

      If the break key is pressed, CP posts an attention interrupt to the virtual machine. If the virtual machine responds with a Read, the break key is passed to the virtual machine. If the virtual machine does not respond with a Read and the break key is pressed a second (or more) time, CP posts another attention interrupt to the virtual machine. In both cases, the passing of the break key interrupt to the virtual machine overrides the BRKKEY setting (if the interrupt came by request of the application program using the X'10' bit in the control byte of the previous Erase/Write or Erase/Write Alternate operation).

11. 3270 SIO/SSCH full-screen mode interactions:

    Before a guest virtual machine can issue 3270 SIO/SSCH commands, it must first ensure that the console mode is set to 3270. For more information on setting the console mode, see the Terminal Conmode command in the *z/VM: CP Commands and Utilities Reference*. After the console is designated a 3270 console, whenever CP is ready to give up control of the screen it presents a CLEAR attention interrupt to the virtual machine. It is the responsibility of the application program to issue an Erase/Write to refresh the screen. If a virtual machine issues only a Write that does not cover the entire screen, information that CP displayed remains on the screen.

12. Double-byte character set (DBCS) line-mode console output may be sent to the virtual machine console by CP, or when CP obtains messages and responses from a DBCS NLS message repository, or by the virtual machine.

    The virtual machine sends line-mode output to its virtual machine console by using mixed DBCS data streams. Mixed DBCS data streams contain DBCS character strings surrounded by a shift out/shift in (SO/SI) pair. Single-byte character set (SBCS) data may or may not be present in the data stream.

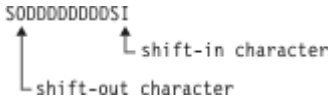
    Example 1—A Data Stream Containing SBCS Data

    The following contains SBCS data identified by a B:

    ```
    BBBBBBBBBBBBBBBBBBBBBB
    ```

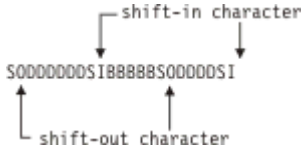
    Example 2—A Data Stream Containing DBCS Data

    The following contains DBCS data identified by a D:



Example 3—A Data Stream Containing Mixed DBCS Data

The following contains SBCS data identified by a B and DBCS data identified by a D:



The virtual machine can send DBCS line-mode data by issuing SIO, SIOF, or SSCH to a channel program containing a Write CCW addressing a mixed DBCS data stream, or by issuing DIAGNOSE X'58' opcode X'49'. For the I/O request to succeed, TERMINAL TEXT and TERMINAL APL must be turned off. This can be checked with the QUERY TERM command. If these are not turned off, the request fails with a command reject indication.

When a virtual machine issues I/O to a channel program, CP ensures the console is DBCS capable. If a console is DBCS capable and is not an SNA/CCS terminal, CP formats the data for the screen and properly displays the data. If it is not DBCS capable, the request is rejected with a data check indication.

You do not have to insert new line (NL) characters in the data stream to format the data for the physical line size of the terminal on an SIO, SIOF, or SSCH. CP formats the data, using the smaller of the:

- Physical width of the screen or
- Logical line size.

This is determined by the TERMINAL LINESIZE command, as follows:

- If LINESIZE=0, LINESIZE is turned off and the physical width of the screen is used.
- If LINESIZE=1, 2, or 3, a line size of 4 is used, because this is the smallest line-size value that displays one byte of DBCS data.

CP ensures, for the DBCS data, that each line displayed contains an even number of DBCS bytes surrounded by an SO/SI pair. If the data is not an even number of bytes or if the data is not surrounded by an SI/SO pair, the request is rejected with a data check indication.

Note: If the data is spooled to the console log or passed to applications over the *MSG or *MSGALL connection, the DBCS data stream is properly formatted and displayed after the data is sent.

For SNA/CCS terminals, NL characters are inserted in the data stream based on the smaller of the logical or the physical line size of the terminal before the data is sent to the terminal; the data is then passed to the appropriate SNA/CCS or real terminal processor to complete writing the data to the terminal.

When DIAGNOSE code X'58' opcode X'49' is used, it is the user's responsibility to format the data by inserting NL characters in the data. This is based on the smaller of the logical line size or the physical width of the screen, which is determined by the TERMINAL LINESIZE command. If LINESIZE=0, LINESIZE is turned off and the physical width of the screen is used; if LINESIZE=1, 2, or 3, a line size of 4 is used, because this is the smallest line-size value that displays one byte of DBCS data. When determining the length of the data, keep in mind the following:

- SI, NL, and SO characters must be inserted, so space in the data stream must be considered for these characters.
- When an NL character is inserted within DBCS data, the data cannot be split in the middle of a double-byte character.

Responses

None.

DIAGNOSE Code X'5C' – Error Message Editing

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'5C' to edit error messages according to your setting of the EMSG function, as determined by the SET EMSG command, or according to the setting of the EMSG function of another virtual machine.

Entry Values:

Rx
Contains the address where the first byte of the message to be edited can be found. If the subcode is X'40', the Rx register cannot be register 15.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the message text. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the message text is in the host-primary address space.

Rx+1
Contains the length of the message header if the subcode is X'40'.

Ry
Consists of *ssxxllll*

ss
is the subcode and modifier code

xx
is not checked

llll
is the length of the message to be edited.

If the alternate user ID modifier (X'80') is specified, Ry cannot be register 15.

Ry+1
Contains the address of an 8-byte field containing the alternate user ID and padded with blanks if necessary. (Only if Modifier code X'80' is specified.)

Ay+1
Is used only by XC virtual machines in access-register mode. Ay+1 contains the ALET for the address space containing the user ID.

Subcode
Meaning

X'00'
Indicates that a 10-byte message header (code) is assumed; the actual contents of the message are not checked.

X'20'
Indicates that the length of the message header must be calculated; DIAGNOSE code X'5C' assumes that the header runs up to, but does not include, the first blank. It also assumes the text begins at the location following the first blank.

X'40'
Indicates that the length of the message header is being passed in Rx+1; the actual contents of the message are not checked.

Modifier Code
Meaning

X'80'
Indicates that the issuer wishes to use the EMSG setting of the user ID specified by Ry+1. Although X'00', X'20', and X'40' are all mutually exclusive, you can combine modifier code X'80' with any one of the above subcodes. For example, a subcode of X'A0' is a combination of X'20' and X'80'. Modifier code X'80' is valid only for class B enabled user IDs.

Exit Values: On return from DIAGNOSE code '5C', CP sets a condition code (described in the Responses section) and, based on the EMSG setting, returns Rx and Ry.

EMSG Setting: CP tests the EMSG setting and returns Rx and Ry to the caller modified as follows:

EMSG Setting	Rx	Ry
ON IUCV	No change	The subcode/modifier code byte is zeroed out. The rest of the register remains unchanged.
CODE	No change	The length of the header, as determined by the subcode.
TEXT	The pointer to the text part of the message	The length of the text alone
OFF	No change	0

Usage Notes

1. DIAGNOSE code X'5C' does not write the message; it merely rearranges the starting pointer and length.
2. Modifier code X'80' is only valid for class B enabled user IDs.

Responses

Condition Codes: When the alternate user ID modifier (X'80') is specified, one of the following condition codes is returned:

Condition Code	Meaning
0	Successful completion
2	The specified user ID is not logged on

When the alternate user ID modifier is not specified, the condition code remains unchanged.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'5C' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Any of the following: • The subcode in Ry bits 1-7 is invalid. • For subcode X'40': Rx is register 15. • For modifier code X'80': the user ID is not aligned on a doubleword boundary, or Ry is register 15.

Problem Encountered	Cause
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state or has requested but is not authorized for alternate user ID processing. • Modifier code X'80' was specified and the user does not have class B privileges.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to: • Fetch the message text (subcode X'20' only) • Fetch the user ID (modifier code X'80')

DIAGNOSE Code X'60' – Determine Virtual Machine Storage Size

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'60' to determine your virtual machine's defined storage size, that is, to determine the size of your virtual machine's host-primary address space, excluding any discontiguous saved segment or saved system storage loaded above the defined size.

Note: For a virtual machine in 64-bit addressing mode, subcode X'0000000C' of Diagnose X'260' may be used to obtain the virtual machine storage size. Subcode X'00000000' of Diagnose X'260 ' may be used as an alternative to Diagnose X'60' for a virtual machine in 24-bit or 31-bit addressing mode.

Entry Values: You must specify a register number for Rx.

Exit Values: On return, the register specified as Rx contains the virtual machine storage size, in bytes.

Responses

None.

DIAGNOSE Code X'64' – Named Saved Segment Manipulation

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'64' to manipulate named saved segments from a user's virtual machine. The DIAGNOSE code X'64' interface is used in application programs as a linkage mechanism to previously-created saved segments. These segments have been defined by the DEFSEG command, initialized by a product or application with the necessary code and data, and saved by the SAVESEG command.

A discontiguous saved segment (DCSS) is a saved segment that occupies one or more architected segments. It can be embedded above the virtual machine's defined storage size and begins and ends on a megabyte boundary.

Saved segments may have been defined as segment spaces or as members of a segment space. All references to named saved segments through any DIAGNOSE code X'64' function is by the name of the saved segment. This means that all names are unique, whether they have been defined as segment spaces or member saved segments. The results of a function are based on whether the object of the processing was a segment space name or a member saved segment name.

Entry Values:

Rx
Subcodes X'00', X'04', X'08', X'0C', X'10', X'20', X'24', X'2C': Contains the address of an 8-byte buffer containing the name of a saved segment. The saved segment name must be 8 characters or less in

length, on a doubleword boundary, left-justified, and padded with blanks to 8 characters. Access to the segment name is subject to key-controlled protection.

Subcodes X'18', X'38': Contains the address of a parameter area. Parameters include the name of the saved segment, a code for the kind of information to be returned about the segment, and specifications for an output area to receive information. The output area must be in the same address space as the parameter area. Access to the parameter area is subject to key-controlled protection.

Note: When running in 64-bit mode, the address in Rx is interpreted as a 31-bit address. In 24- and 31-bit modes, the address is interpreted according to the addressing mode of the guest.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer to which Rx points. In the case of subcode X'18', the output area is also in this address space. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
Contains a function subcode.

In the following pages, the possible functions are listed in the order of their subcodes. After each subcode number is a description of what the function does.

Subcode X'00' — LOADSHR

This function loads a saved segment in shared mode. After this function executes, the storage occupied by the named saved segment is addressable by the virtual machine, even if the loaded saved segment is outside the addressing range of the virtual machine. However, any storage beyond that defined for the virtual machine and below that defined for the saved segment is not addressable. The following describes the processing characteristics as they relate to the different forms of saved segments:

DCSS
Purges any saved segment that is within the range definition of the DCSS being loaded. If the saved segment being loaded was defined without the SPACE operand, then this segment is made accessible to the user virtual machine.

Rx contains the actual beginning address of the DCSS.

- The beginning address of the DCSS is the first byte of the lowest page value that was specified on the DEFSEG command.

Ry contains a rounded ending address of the DCSS.

- The ending address of a DCSS is rounded up to the last byte of the highest page value that was specified on the DEFSEG command.

Segment space
This function purges any saved segment within the range definition of the segment space being loaded. If the segment space being loaded has members, each member of the segment space is made accessible to the user virtual machine. Rx contains the actual beginning address of the segment space.

- The beginning address of a segment space is the first byte of the lowest page value that was specified for a member having the lowest page value on the DEFSEG command.

Ry contains a rounded ending address of the segment space.

- The ending address of a segment space is rounded up to the last byte of the highest page value that was specified for a member having the highest page value on the DEFSEG command.

Member saved segment
This function purges any saved segment that is within the range definition of the segment space that contains the loaded member. However, any previously loaded members that are also part of the same segment space as the member being loaded are unaffected. The loaded member saved segment is made accessible to the user virtual machine.

Rx contains the actual beginning address of the member saved segment.

- The beginning address of the member saved segment is the first byte of the lowest page value that was specified on the DEFSEG command.

Ry contains the actual ending address of this member saved segment.

- The ending address of the member saved segment is rounded up to the last byte of the highest page value that was specified on the DEFSEG command.

If the member saved segment was outside the addressable range of the virtual machine to which it was loaded/attached, the virtual machine may now reference any portion of storage occupied by the segment space that contains the member.

Notes:

1. A member saved segment loaded through LOADSHR after a previous member saved segment of the same segment space was loaded through LOADNSHR also produces a nonshared copy of the segment.
2. If a DCSS is already loaded and a request to load it again occurs, then this results in a refreshed copy of any exclusive writeable storage for that DCSS in the virtual machine being attached. If a member saved segment is already loaded and a request to load it again occurs, then this results in a refresh of any exclusive writeable storage for that member saved segment in that virtual machine, but storage for other member saved segments in the same segment space is not refreshed. If a segment space was already explicitly loaded by name, and an explicit request is made to load it by name again, then there is no effect and the segment space storage is not refreshed.

Subcode X'04' — LOADNSHR

Loads a saved segment in exclusive-write mode after purging any existing named saved segment or saved segment.

If the DCSS or segment space was defined with the LOADNSHR operand, any user will be allowed to load an exclusive-write mode copy. If a member saved segment is requested and any member of the space in which it will be loaded was so defined, the same is true.

When a user requests a nonshared copy of a saved segment and that segment fits below the user's maximum definable address space size, the user will be allowed to load the segment. (The maximum definable address space size is determined from the user's directory entry.)

If the saved segment was not defined with the LOADNSHR operand, and its address range(s) goes above the user's maximum definable address space size, a NAMESAVE entry is required in the user's directory entry. If the segment being loaded is a member saved segment, then the NAMESAVE entry is the segment space name.

The effect of loading a segment space or a member of a segment space is similar to what is described under the LOADSHR function. The difference is that the segment space storage is loaded in exclusive-write mode, regardless of the mode in which it was originally defined.

Notes:

1. If a member saved segment is loaded through LOADNSHR after a previous member saved segment of the same segment space but with a different name was loaded through LOADSHR, then the load request will terminate in an error. This is because the segment space containing both members has already been loaded in shared mode.
2. The user receives the saved segment in exclusive-write mode, regardless of the mode in which it was originally defined.
3. If a DCSS is already loaded non-shared and a request to load it non-shared again occurs, then this results in a refreshed copy of the entire segment being attached. If a member saved segment is already loaded non-shared and a request to load it non-shared again occurs, then this results in a refreshed copy of the member saved segment being attached, but other member saved segments in the same segment space are not refreshed. If a segment space was already explicitly loaded by name non-shared, and an attempt is made to explicitly load it by name non-shared again, then there is no effect and the segment space storage is not refreshed.

Subcode X'08' — PURGESEG

Releases a saved segment from guest real storage.

If PURGESEG is invoked for a saved segment not previously loaded, the request is terminated with a condition code of 1. The following describes the processing characteristics as they relate to the different forms of saved segments:

DCSS
If the DCSS being purged was defined without the SPACE operand, then the result would be the same as described for segment space.

When a PURGESEG is executed for a segment in nonshared mode residing in the users virtual machine address space, the storage is cleared to binary zeros and the keys are reset to zero.

Segment space
If the segment space being purged was originally loaded by spacename, then each member of the segment space that is loaded or attached to the user virtual machine is purged (see note). This releases the associated storage that was acquired when the corresponding load function was executed.

Note: If a user had previously loaded a segment space by spacename and also loaded a member of the same segment space by a DCSS name, and then issued a PURGESEG for the spacename, the result of the purge function does not detach the member saved segment that was loaded through the DCSS name.

If the storage occupied by the saved segment was beyond the defined virtual machine storage size, that storage is no longer addressable by the virtual machine.

Member saved segment
The loaded or attached member saved segment is purged from the user virtual machine. This releases the associated control block storage that was acquired when the corresponding load function was executed. If the purged member saved segment was the last member of the segment space that was loaded for this virtual machine, the storage associated with the segment space is released.

If the storage occupied by the member saved segment was beyond the defined virtual machine storage size, that storage would still be addressable by the virtual machine if another member of the same segment space was still loaded to the virtual machine.

Subcode X'0C' — FINDSEG

Returns the start and end page addresses of the named saved segment.

In z/VM, the existence of the saved segment is found by searching for an active segment, and then for a skeleton segment. An active segment search is defined as follows:

1. If the saved segment is attached to the virtual machine, the address information is returned from the description in storage.
2. If the saved segment is a member of a segment space, and if another member of this same segment space is attached to the virtual machine, then the address information is returned from the description in storage. This is true even if the member or space is pending purge.
3. If an active segment (a segment for which a class A or R SDF file is created) is found, the address information is returned from the description in the SDF.

   Note: A saved segment that is in pending purge status (where its SDF file class is P) is bypassed in the search process.

If a skeleton segment search is entered when no active segment exists:

- Search for a skeleton segment definition (a segment having only a class S SDF file created); if found, then the address information is returned from the description in the SDF.

The following describes the processing characteristics as they relate to the different forms of saved segments.

DCSS
The Rx register contains the beginning address.

- The beginning address of the DCSS segment is the first byte of the lowest page value that was specified on the DEFSEG command.

The Ry register contains the ending address.

- The ending address of the DCSS is rounded up to the last byte of the highest page value that was specified on the DEFSEG command.

Segment space
If the request is for a segment space that has members, the beginning and ending addresses are the lowest and highest page addresses of any of the members.

The Rx register contains the beginning address.

- The beginning address of a segment space is the first byte of the lowest page value that was specified for a member having the lowest page value on the DEFSEG command.

The Ry register contains the ending address.

- The ending address of a segment space is rounded up to the last byte of the highest page value that was specified for a member having the highest page value on the DEFSEG command.

Member saved segment
If the request is for a member of a segment space, the actual beginning and ending addresses are returned for the specified member saved segment as determined from the actual page ranges specified through the DEFSEG command.

The Rx register contains the beginning address, that is, the first byte of the lowest page value that was specified on the DEFSEG command.

The Ry register contains the ending address, that is, the address that is rounded up to the last byte of the highest page value that was specified on the DEFSEG command.

Subcode X'10' — LOADNOLY

Loads a saved segment in shared mode only if no overlay condition exists.

The effect of loading a segment space or a member of a segment space is the same as described under the LOADSHR function. In this case, the load would be completed only if an overlay condition did not exist.

Subcode X'18' — SEGEXT

Returns information relative to saved segments through user-supplied areas.

Rx is set up to point to a user-supplied buffer (that is, contains a guest real address). The parameter portion of the buffer describes the input to the function and on return from the function the parameter area contains status information. The buffer has a minimum length of three doublewords. The parameter portion of the buffer must reside on a doubleword boundary and cannot cross a page boundary (Rx content + the buffer length cannot cross a page boundary). This reserves a parameter area that may be up to one 4KB-page in length.

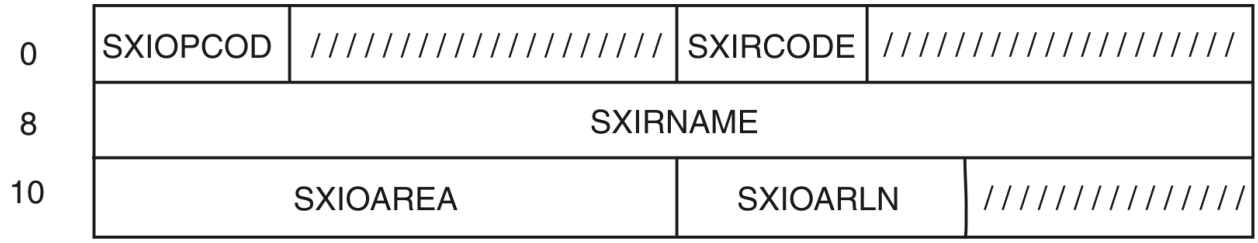
The SXIOAREA and SXIOARLN fields of the parameter area define an output area address and length. This area must also reside on a doubleword boundary and cannot cross a page boundary (SXIOAREA content + SXIOARLN content cannot cross a page boundary). This reserves an output area up to one 4KB-page in length. The output area must be in the same address space as the parameter area.

The input buffer format is provided to map the parameter area, and the input buffer area is provided to map the output area for the functions supported.

The format of the parameter and output areas is shown in Figure 3 on page 67.

Ry is set up to contain the X'18' function code.

Input Buffer Format - Parameter Area
HCPSXIBK COPY (found in the HCPGPI macro library)



Output Area
HCPSXOBK COPY (found in the HCPGPI macro library)

Figure 3. The Format of the User-Supplied Areas for the SEGEXT Function

The definition of the parameter area is as follows:

SXIOPCOD
is a 1-byte field containing the code for the operation to be performed. Possible values are:

X'01'
FINDSPACE

X'02'
FINDSKEL

X'0C'
FINDSEGA

X'0D'
FINDNSSA

Descriptions of these operations follow. Opcodes not defined by this function are reserved for IBM use.

////...
is a 3-byte field reserved for IBM use.

SXIRCODE
is a 1-byte field containing a return code from the function.

The following return codes and meanings are valid for FINDSPACE, opcode X'01':

X'00'
The SXIRNAME and SPACENAM fields are equal, the request was for DCSS.

X'04'
The SXIRNAME and SPACENAM fields are not equal, the request was for a member saved segment.

X'08'
The SXIRNAME and SPACENAM fields are equal, the request was for a segment space.

The following return codes and meanings are valid for FINDSKEL, opcode X'02' and FINDSEGA, opcode X'0C':

X'00'
The name of the segment in SXIRNAME was found to be a DCSS structure.

X'04'
The name of the segment in SXIRNAME was found to be a member saved segment structure.

X'08'
The name of the segment in SXIRNAME was found to be a segment space structure.

The following return code and meaning are valid for FINDNSSA, opcode X'0D':

X'00'
There is an IPLed NSS.

////...
is a 3-byte field reserved for IBM use.

SXIRNAME
is an 8-byte field containing the requested segment name. This name is left-justified and padded with blanks if necessary.

For the FINDNSSA operation, SXIRNAME is ignored.

SXIOAREA
is a fullword field containing the address of the output area. This area must begin on a doubleword boundary. The output area resides in the same host address space as the input parameter list.

SXIOARLN
is a halfword field containing the length of the output area in bytes. The output area must not cross a page boundary.

////...
is a halfword field reserved for IBM use.

The definition of the output area is based on the OPCODE field function.

SEGEXT Function Operation Codes

The possible operation codes under the SEGEXT function are listed on the following pages in order of their codes. Following each operation name is a description of how the operation works.

Opcode X'01' — FINDSPACE

This operation returns the beginning and ending addresses of a segment space if either a member saved segment name or a segment space name is supplied. In addition, the name of the segment space is returned. If a DCSS name is supplied, then the beginning and ending addresses returned for a segment space are the same as what was defined for a DCSS.

FINDSPACE searches for an active segment, then for a skeleton. The search is defined as follows:

1. If the saved segment is attached to the virtual machine, the address information is returned from the description in storage.
2. If the saved segment is a member of a segment space, and if another member of this same segment space is attached to the virtual machine, the address information is returned from the description in storage.
3. An active segment is sought (a segment for which a class A or R SDF file is created); if it is found, the address information is returned from the description in the SDF.

   Note: A saved segment that is in pending purge status (where its SDF file class is P) is bypassed in the search process.
4. Search for a skeleton segment definition (a segment having a class S SDF file created); if it is found, the address information is returned from the description in the SDF.

The format of the buffer for the FINDSPACE operation is as shown in Figure 4 on page 69.

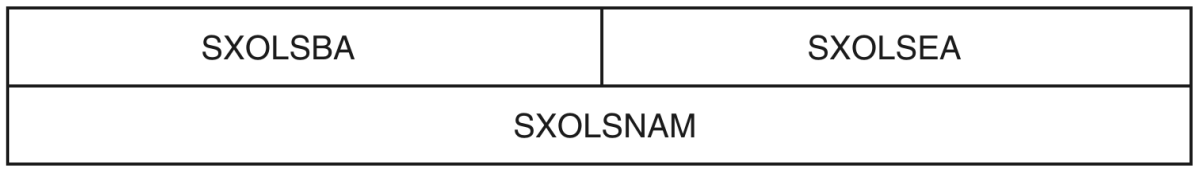


Figure 4. The Format of the User-Supplied Areas for the FINDSPACE Operation

SXOLSBA
A fullword containing the beginning guest absolute address of the segment space that contains the requested saved segment. The address of the megabyte boundary containing the lowest page definition of the segment space is returned.

SXOLSEA
A fullword containing the ending guest absolute address of the segment space that contains the requested saved segment. The last address of the megabyte in which the page with the highest definition resides, is returned.

SXOLSNAM
A doubleword field containing the name of the segment space that contains the requested saved segment. If this name and SXIRNAME are the same, the requested named segment is the segment space or a DCSS. If this name is different from SXIRNAME, the requested named segment is a member of this segment space. The name is returned left-justified, padded with blanks.

If the member saved segment belongs to multiple segment spaces, the segment space name would pertain to the segment space that would have been loaded if a DIAGNOSE code X'64' LOAD was issued.

Opcode X'02' — FINDSKEL

The FINDSKEL operation searches for a skeleton segment search definitions (a segment having a class S SDF file created). If it is found, the address information is returned from the description in the SDF. Any search for any active definitions is bypassed.

For information on the buffer used for the FINDSKEL operation and other information common with FINDSEGA, see the second half of the description of FINDSEGA that follows.

Opcode X'0C' — FINDSEGA

This operation returns the page range values and page range attributes that were specified on a DEFSEG command. When operation code X'02' is specified, the page definition information is related to the skeleton file, a nonactive segment definition whose spool file class is S. The specification of operation code X'0C' returns the page definition information related to an active segment definition.

When FINDSEGA is issued from your virtual machine, only an active segment search is performed (a segment for which a class A or class R SDF file is created). The order of search is as follows:

1. If the saved segment is attached to your virtual machine, the address information is returned from the description in storage.
2. If the saved segment is a member of a segment space, and if another member of this same segment space or the space itself is attached to your virtual machine, the address information is returned from the description in storage.
3. If the saved segment is attached to another virtual machine or is contained inside a segment space and is not in a pending purge state, the address information is returned from the description in storage. If the saved segment was in a pending purge state, the saved segment is bypassed in the search operation. This behavior is the same even if the saved segment is a DCSS structure.
4. The SDF files are searched next. Search for an active segment; if it is found, the address information is returned from the description in the SDF.

   Note: A saved segment in pending purge state (where its SDF file class is P) is bypassed in the search process.

The FINDSKEL and FINDSEGA operation codes may be used by install programs to initialize the areas within the returned page range addresses with the appropriate code or data. Because z/VM may have an active segment definition as well as a skeleton definition at the same time, the product can now explicitly select the desired definition of the segment.

The page range definition information applies only to a DCSS, a member saved segment, or an NSS. If the FINDSKEL or FINDSEGA is issued for a name that is a segment space, no page range table data is returned.

To determine the segment structure in terms of a DCSS, member saved segment, or a segment space, the return code may be checked.

The format of the buffer for the FINDSKEL, FINDSEGA, or FINDNSSA operation is as shown in Figure 5 on page 70. The output area shown in this figure is used when SXIRNAME is a DCSS, or a member saved segment, or when the FINDNSSA function is performed. See Figure 6 on page 72 for a picture of the output area when SXIRNAME is a segment space.

Output area with maximum output
HCPSXOBK COPY (found in the HCPGPI macro library)

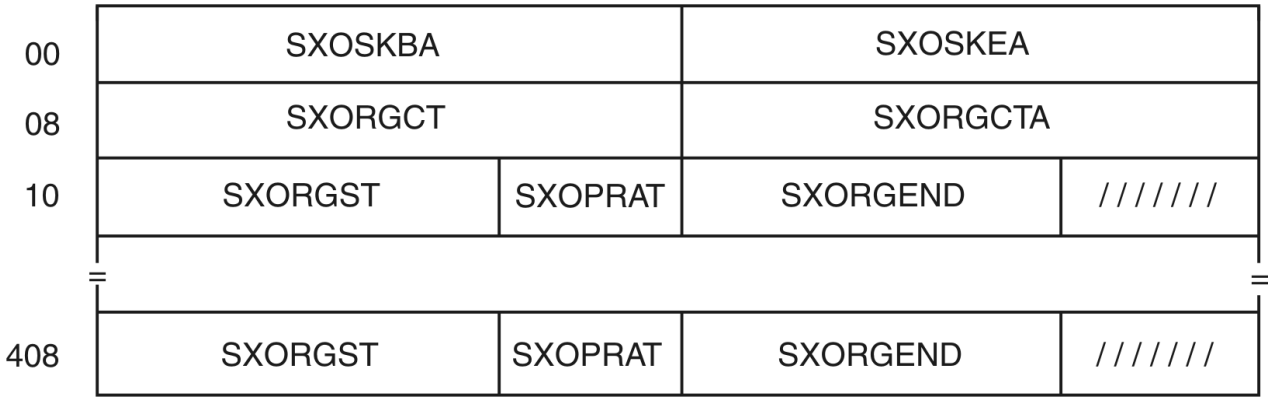


Figure 5. The Format of the User-Supplied Areas for the FINDSKEL, FINDSEGA, or FINDNSSA operations

SXOSKBA
A fullword that contains the beginning guest absolute address of the global range of the skeleton or active segment as defined by the DEFSEG command or of the IPLed NSS as defined by the DEFSYS command. The address returned is the lowest page value defined for the member, DCSS, or NSS.

SXOSKEA
A fullword that contains the ending guest absolute address of the global range of the skeleton or active segment as defined by the DEFSEG command or of the IPLed NSS as defined by the DEFSYS command. The address returned is the last byte of the page for the highest page value defined for the member, DCSS, or NSS.

SXORGCT
A fullword field containing the count of valid page range entry pairs (Page Range Start and Page Range End) in the output area.

Note:

1. If the output area length specified by the SXIOAREA field is not large enough to accommodate all the entries, SXORGCT is **not** adjusted to reflect the number of entries contained in the limited size buffer. Refer to the SXORGCTA field.

SXORGCTA
A fullword field containing the count of valid page range entry pairs (SXORGST and SXORGEND) that were actually placed in the output area page range table. This may be used when the output buffer provided was not large enough to contain all the page range entries that were available.

The start of a 1024-byte page range information table that contains doubleword entries that are broken down into a start (SXORGST) and ending (SXORGEND) address range definition. The attribute of the page range definition (SXOPRAT) is contained in the fourth byte of the entry. The page range entries are

ordered from lowest value to highest value. A maximum of 128 page range entries may be placed in this table.

If the request is issued against a segment space, the page range table is **not** modified. A segment space has no page range definitions associated with itself, only with its members.

SXORGST
A 3-byte field containing the high-order three bytes of the start page range entry definition. To produce a valid start page address, this value must be placed in the high-order three bytes of a register, and the low order byte of the register must be set to zero. The result is a guest absolute address.

SXOPRAT
A 1-byte field containing the page range attribute flag. Bit 7 (X'01' – SXOEXCL) of this field indicates an exclusive copy; each user gets a separate copy of this page range. Bit 6 (X'02' – SXOPROT) of this field indicates page ranges that are page protected; users may access these pages in read-only mode. Bit 5 (X'04' – SXONDAT) of this field indicates page ranges, the data in which is not saved in the system data file (SDF). Combinations of these bits produce the following codes:

X'00'
SW — shared read/write access.

X'01'
EW — exclusive read/write access.

X'02'
SR — shared read-only access.

X'03'
ER — exclusive read-only access.

X'04'
SN — shared read/write access, no data saved.

X'05'
EN — exclusive read/write access, no data saved.

X'06'
SC — shared read-only, no data saved, CP writeable pages.

SXORGEND
A 3-byte field containing the high-order three bytes of the end page range entry definition. To produce a valid end page address, this value must be placed in the high-order three bytes of a register, and the low-order three hexadecimal digits must be set to X'F'. The result is a guest absolute address.

////...
A 1-byte field reserved for IBM use.

Note: The output area length specified by SXIOAREA dictates the amount of information provided in the output buffer area.

The output area buffer for the FINDSKEL or FINDSEGA operation for SXIRNAME that is a segment space will contain two doublewords of member information for each member entry in the following format:

Output area with maximum output
HCPSXOBK COPY (found in the HCPGPI macro library)

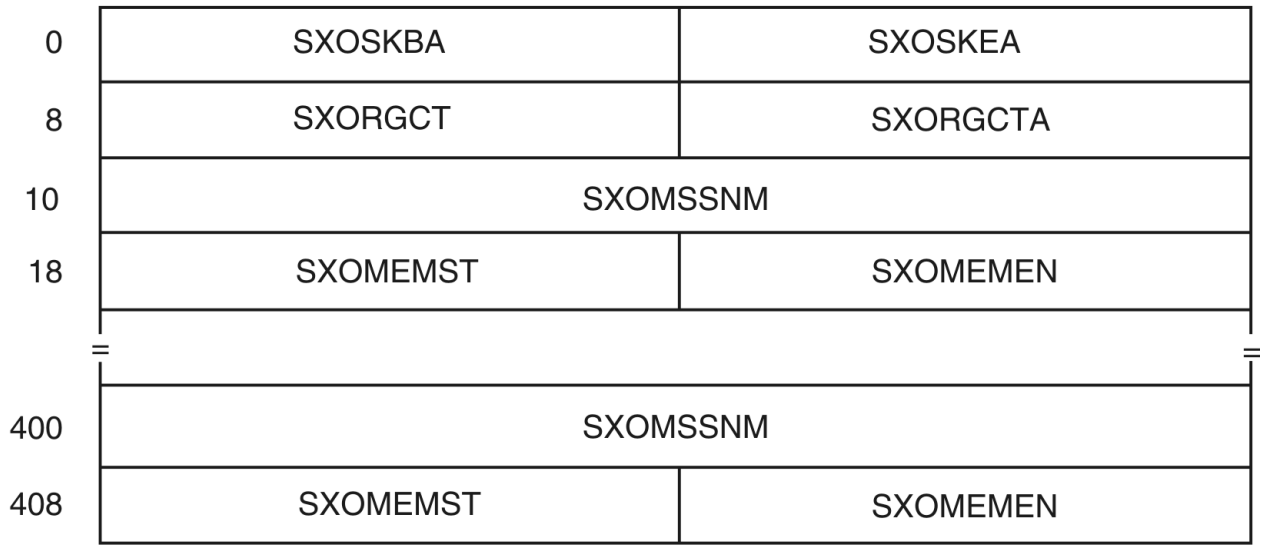


Figure 6. The Format of the User-Supplied Output Area – Member List

SXOSKBA
A fullword that contains the beginning guest absolute address of the global range of the skeleton or active segment as defined by the DEFSEG command. The address returned is the first byte of the lowest page value that was specified for a member having the lowest page value on the DEFSEG command.

SXOSKEA
A fullword that contains the ending guest absolute address of the global range of the skeleton or active segment as defined by the DEFSEG command. The address returned is rounded up to the last byte of the highest page value that was specified for a member having the highest page value on the DEFSEG command.

SXORGCT
A fullword field containing the count of defined member saved segment for this segment space.

Note: If the output area length specified by SXIOAREA field is not large enough to accommodate all the entries, SXORGCT is **not** adjusted to reflect the number of entries contained in the limited size buffer. Refer to SXORGCTA.

SXORGCTA
A fullword field containing the count of member saved segment entries that are actually placed in the output area member list table. This may be used when the output buffer provided was not large enough to contain all the member saved segment entries that were available.

Note: SXORGCTA reflects only full (two doubleword) entries that were placed in the output area.

The start of a 1024-byte member list information table containing two doubleword entries of the member saved segment names that were defined for the segment space and the starting and ending page range defined for the member.

SXOMSSNM
The name of a member saved segment that was defined for the segment space identified by the SXIRNAME (requested segment name) field.

SXOMEMST
The lowest page value defined for the member saved segment when defined by the DEFSEG command. The first 3 bytes contain this page value; the fourth byte contains the status information for the member saved segment.

X'01'
MPENP — indicates the member saved segment is in the pending purge state.

X'02'
MSAVD — indicates the member saved segment has been saved. If this bit is not on, then it indicates this member is not saved.

SXOMEMEN
The highest page value defined for the member saved segment when defined by the DEFSEG command. The first 3 bytes contain this page value. The fourth byte is reserved for IBM use.

Opcode X'0D' — FINDNSSA

The FINDNSSA operation returns the page range values and page range attributes that were specified on a DEFSYS command for the issuer's currently-IPLed NSS.

The input buffer content is the same as defined for the FINDSEGA operation. See Figure 3 on page 67 and the accompanying explanation.

The output area content is as defined for the FINDSKEL and FINDSEGA functions for a DCSS or a member saved segment. For more information, see Figure 5 on page 70 and the accompanying explanation.

Subcode X'20' — LOADSHR (64-Bit)

Subcode X'20' performs the same function as "Subcode X'00' — LOADSHR" on page 63, except that output addresses are 64-bit instead of 31-bit.

Subcode X'24' — LOADNSHR (64-Bit)

Subcode X'24' performs the same function as "Subcode X'04' — LOADNSHR" on page 64, except that output addresses are 64-bit instead of 31-bit.

Subcode X'2C' — FINDSEG (64-Bit)

Subcode X'2C' performs the same function as "Subcode X'0C' — FINDSEG" on page 65, except that output addresses are 64-bit instead of 31-bit.

Subcode X'38' — SEGEXT (64-Bit)

Subcode X'38' performs the same function as "Subcode X'18' — SEGEXT" on page 66, except that output addresses are 64-bit instead of 31-bit and the output areas for the operation codes have different formats.

The format of the output areas for opcode X'01' (FINDSPACE) is as shown in Figure 7 on page 73.

Output Area
HCPSXOBK COPY (found in the HCPGPI macro library)

00	SXOLSBAG
08	SXOLSEAG
10	SXOLSNMG

Figure 7. The Format of the User-Supplied Areas for a 64-Bit FINDSPACE Operation

The 64-bit output areas have different names but are used for the same functions as the corresponding 31-bit output areas. See the descriptions of the 31-bit areas shown in Figure 4 on page 69.

64-Bit Area Name	31-Bit Area Name
SXOLSBAG	SXOLSBA

64-Bit Area Name	31-Bit Area Name
SXOLSEAG	SXOLSEA
SXOLSNMG	SXOLSNAM

The format of the output areas for opcode X'02' (FINDSKEL) and opcode X'0C' (FINDSEGA) for a DCSS or member segment, and opcode X'0D' (FINDNSSA), is as shown in Figure 8 on page 74.

Output area with maximum output
HCPSXOBK COPY (found in the HCPGPI macro library)

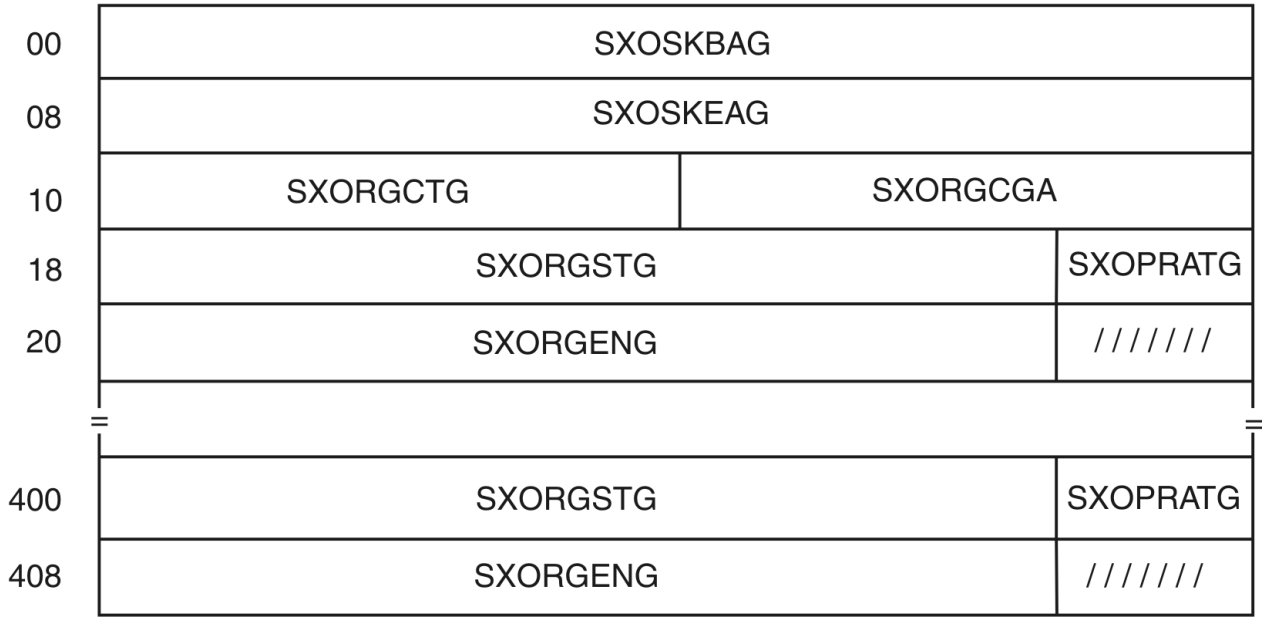


Figure 8. The Format of the User-Supplied Areas for a 64-Bit FINDSKEL or FINDSEGA Operation for a DCSS or Member Segment, or a 64-Bit FINDNSSA Operation

The 64-bit output areas have different names but are used for the same functions as the corresponding 31-bit output areas. See the descriptions of the 31-bit areas shown in Figure 5 on page 70.

64-Bit Area Name	31-Bit Area Name
SXOSKBAG	SXOSKBA
SXOSKEAG	SXOSKEA
SXORGCTG	SXORGCT
SXORGCGA	SXORGCTA
SXORGSTG	SXORGST
SXOPRATG	SXOPRAT
SXORGENG	SXORGEND

The format of the output areas for opcode X'02' (FINDSKEL) and opcode X'0C' (FINDSEGA) for a segment space is as shown in Figure 9 on page 75.

Output area with maximum output
HCPSXOBK COPY (found in the HCPGPI macro library)

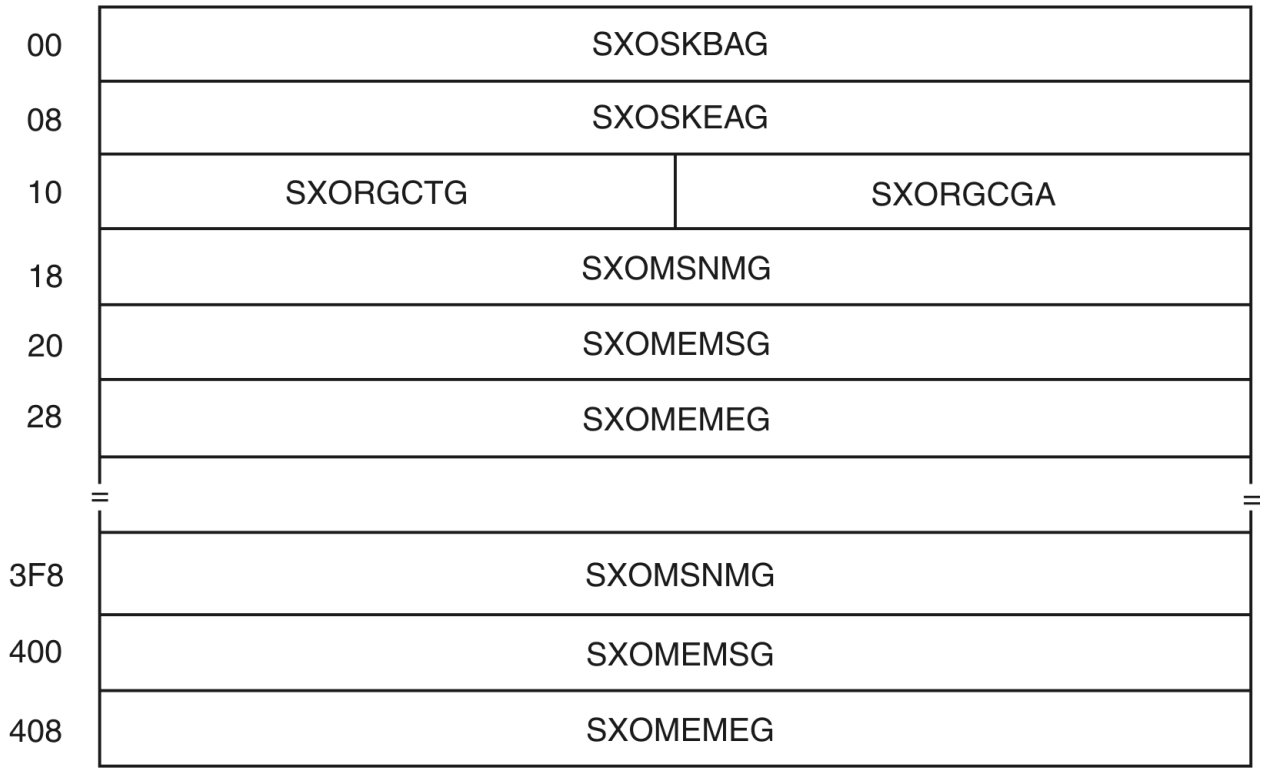


Figure 9. The Format of the User-Supplied Output Areas for a 64-Bit FINDSKEL or FINDSEGA Operation for a Segment Space

The 64-bit output areas have different names but are used for the same functions as the corresponding 31-bit output areas. See the descriptions of the 31-bit areas shown in Figure 6 on page 72.

64-Bit Area Name	31-Bit Area Name
SXOSKBAG	SXOSKBA
SXOSKEAG	SXOSKEA
SXORGCTG	SXORGCT
SXORGCGA	SXORGCTA
SXOMSNMG	SXOMSSNM
SXOMEMSG	SXOMEMST
SXOMEMEG	SXOMEMEN

Usage Notes

1. When the host-primary address space of your virtual machine is considered a shareable address space, you can successfully use only the FINDSEG, FINDSPACE, FINDSKEL FINDSEGA, or FINDNSSA operations of DIAGNOSE code X'64'. A request for any of the other operations of this DIAGNOSE code will result in return code X'0CB'.

   The host-primary address space of your virtual machine becomes shareable if your virtual machine uses the PERMIT function of the ADRSPACE macro to grant another virtual machine access to the address space. The host-primary address space remains in the shareable state until your virtual machine subsequently invokes the ISOLATE function of the ADRSPACE macro, or until a subsystem reset operation is performed on your virtual machine (for example through the SYSTEM RESET

command). For more information on ADRSPACE, see “ADRSPACE — Address Space Services” on page 797.

2. A segment space has a range that may span one or more architected 1 megabyte segments. The beginning address of a segment space is rounded down to the nearest megabyte boundary of the member having the lowest page value. The ending address of a segment space is rounded up to the last address of the last page within the megabyte occupied by the member having the highest page value.
3. Each member saved segment has a range that may span one or more pages. The beginning address of a member saved segment is determined by its lowest page value. The ending address is determined by the last address of the highest page value.
4. If a load with an overlay arrangement occurs, the old segment space is indeed purged, and the new segment space is loaded with the duplicate members.
5. When a member of a segment space is loaded, all members of the segment space are made accessible to the virtual machine.

   Note: However, to access other member saved segments predictably, the DIAGNOSE code X'64' programming interface or the CMS SEGMENT programming interface must be used.
6. When a LOAD*xxx* is issued, the following search order occurs to resolve the load request:
   a. The user's loaded saved segments are searched for the specified name.
   b. Other user's saved segments are searched for the specified name.
   c. The closed class A SDF files are searched for the specified name.
      - If the specified name is a member of one or more segment spaces:
        – The first segment space for which the user is authorized is made accessible to the user's virtual machine, and the specified member is loaded.
      - If the specified name is a DCSS and the user is authorized for the DCSS, it is loaded.
      - If the specified name is a segment space:
        – If the user is authorized for the segment space, it is made accessible to the user's virtual machine and the entire segment space is loaded.
7. A FIND*xxx* request for a saved segment of a specified name that is both not currently loaded to the user's virtual machine and in pending purge status, results in a not-found condition.
8. A PURGESEG purges only a saved segment previously loaded by a corresponding LOAD*xxx* function with the same name.
   - If other members of the same segment space are still loaded, then the virtual machine still has access to the member that was purged.

     Note: An attempt to access a member saved segment after it has been purged cause unpredictable results.
9. If a PURGESEG was issued for the last member of a segment space loaded into a user's virtual machine, the segment space is removed from the user's virtual machine address space. The virtual machine has no accessibility to any area of the segment space.
10. Addresses returned through FINDSPACE are always rounded to megabyte boundaries.
11. The beginning address returned through a LOAD*xxx*, FINDSEG, FINDSEGA, and FINDSKEL is on a page boundary. The ending address is rounded to the page boundary containing the highest page definition.
12. It is the guest's responsibility to serialize writes to writable shared storage.
13. When a DCSS is loaded nonshared, any writable shared storage appears as it was when the DCSS was saved. Changes made to shared pages by other users who have the DCSS loaded shared, are not copied.

14. You may not be authorized to issue subcodes X'00', X'04', X'10', X'20', and X'24' of this DIAGNOSE code if an external security manager is installed on your system. For additional information, contact your security administrator.
15. Subcodes X'00', X'04', X'0C', X'10', and X'18' cannot be used on a DCSS that includes addresses above 2047 MB, because these subcodes support only 31-bit output addresses. If one of these subcodes is used on a DCSS above 2047 MB, the DIAGNOSE code exits with return code X'02C', indicating that the saved segment does not exist.
16. The initial load of a large DCSS can appreciably delay processing of subsequent DIAGNOSE code X'64' requests and the following commands: DEFSEG, SAVESEG, DEFSYS, SAVESYS, INDICATE NSS, IPL of an NSS, PURGE NSS, and QUERY NSS.
17. LOADSHR, LOADNSHR, LOADNOLY, and PURGESEG will fail if executed when VMRELOCATE is in progress.

Responses

Condition Codes for a Normal Exit: Upon a normal exit condition, you receive a condition code of either 0 or 1. Refer to Table 4 on page 77 for the exact results. In the headings for the last two columns of this table, user buffer refers to data in the user's output buffer for SEGEXT functions.

Table 4. Normal Exit Results with the SEGEXT Function

Condition Code	Meaning	Contents of Rx or User Buffer	Contents of Ry or User Buffer
0	(FIND) The saved segment is already loaded.	The actual starting address at which the saved segment is loaded in Rx.	The actual ending address of the saved segment in Ry.
0	(FINDSPACE) If FINDSPACE was issued with the name of the segment space, the segment space exists and it is already loaded or has one of its members already loaded. If FINDSPACE was issued with the name of the member or a DCSS, the saved segment exits and is already loaded.	No change	The ending address is rounded up to the segment (megabyte) boundary containing the highest page value specified for a member on the DEFSEG command.
0	(FINDSEGA) If FINDSEGA was issued with the name of the segment space, the segment space exists and it is already loaded or has one of its members already loaded. If FINDSEGA was issued with the name of the member or a DCSS, the saved segment exits and is already loaded.	No change	The ending address is rounded up to the segment (megabyte) boundary containing the highest page value specified for a member on the DEFSEG command.

Table 4. Normal Exit Results with the SEGEXT Function (continued)

Condition Code	Meaning	Contents of Rx or User Buffer	Contents of Ry or User Buffer
0	(LOAD) The saved segment loaded outside the user's defined virtual storage area.	The actual starting address at which the saved segment was loaded.	The actual ending address of the saved segment.
0	(PURGE) The saved segment is purged.	No change	No change
0	(FINDNSSA) Output area contains NSS configuration information.	No change	No change
1	(FIND) The saved segment exists but is not loaded.	The actual starting address at which the saved segment would be loaded.	The actual ending address of the saved segment in Ry.
1	(FINDSKEL and/or FINDSEGA) The skeleton or active segment exists but is not loaded.	No change	The actual ending address of the skeleton or active segment at which the saved segment would be loaded.
1	(FINDSPACE) The saved segment exists but is not loaded. If it is issued with a segment space name, neither the segment space nor its members are loaded.	No change	The ending address is rounded up to the segment (megabyte) boundary containing the highest page value specified for a member on the DEFSEG command.
1	(LOAD) The saved segment loaded within the user's defined virtual storage area. The member saved segment was outside of the virtual machine's storage, but the segment space containing the member is inside the virtual machine's storage area.	The actual starting address at which the saved segment was loaded.	The actual ending address of the storage released before the saved segment was loaded, or the ending address of the virtual machine.
1	(PURGE) The saved segment does not exist.	No change	No change

Condition Codes for an Exit with Error: Upon an exit error condition, you receive a condition code of 2. In addition, Ry contains a return code (in hexadecimal) indicating the exact error. Refer to Table 5 on page 78 for return code messages.

Note: In this table, return codes are given in both hexadecimal and decimal forms, as the decimal number corresponds to a CP message number unless otherwise noted.

Table 5. Results of Exit with Error from DIAGNOSE code X'64'

Condition Code	Return Code in Ry	Meaning
2 (FINDNSSA)	44 (X'02C')	An NSS is not IPLed.

Table 5. Results of Exit with Error from DIAGNOSE code X'64' (continued)

Condition Code	Return Code in Ry	Meaning
2 (FINDSEGA)	44 (X'02C')	The saved segment does not exist or is part of a segment space that is incomplete.
2	53 (X'035')	The user is not in the CP directory.
2	174 (X'0AE')	Paging I/O error
2	203 (X'0CB')	An operation other than FINDSEG, FINDSPACE, FINDSKEL or FINDSEGA was requested when the host-primary address space was in the shareable state. This return code does not correspond to a CP message number.
2	449 (X'1C1')	The user is not authorized.
2	475 (X'1DB')	A fatal I/O error occurred reading the CP directory.
2	1015 (X'3F7')	Insufficient storage is available to satisfy your request.
2	1351 (X'547')	The saved segment would overlay the existing segment. Rx contains the address of the first segment that would be overlaid.
2	1352 (X'548')	An unacceptable condition occurred.
2	1357 (X'54D')	Reserved. Not used by z/VM.
2	1358 (X'54E')	An attempt was made to load a CP-owned DCSS in nonshared mode.
2	1367 (X'557')	The user attempted to load a member saved segment in a mode (NONSHARE) different than the mode (SHR) established for the segment space to which it belongs.
2	6874 (X'1ADA')	This result is returned if the DEFine STORage CONFIGuration command is used to define the storage configuration for the virtual machine's base address space and more than one configuration extent is specified. Saved systems and segments cannot be loaded into storage that is greater than or equal to the starting address of the second storage configuration extent.

Note: A return code from a load function is always associated with the item that would have produced a successful load. In an example where three choices are possible:

- M1 is a member of segment spaces S1, S2, S3
- S1 is a restricted segment space
- S2 is a segment space with pending purge status
- S3 is a segment space that will have a paging problem

A load of M1 is attempted by a user not authorized for S1. The return code associated with the loading of M1 using segment space S3 would have been returned. If no other options (no S2 or S3) were available, the unauthorized return code would have been given. If S1 and S2 were options, and S2 is in pending purge, then the return associated with the S2 status would have been given.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'64' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • Ry is not a valid subcode. • Rx is not aligned on a doubleword boundary or the buffer crosses a page boundary. • The output area is not aligned on a doubleword boundary or it crosses a page boundary.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to: • Fetch the DCSS name (for functions other than X'0018' and X'0038') • Store into the SEGEXT output area (for functions X'0018' and X'0038' only) • Fetch and store into the SEGEXT parameter list (for functions X'0018' and X'0038' only).

DIAGNOSE Code X'70' – Time-of-Day Clock Accounting Interface

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'70' to request timing information from CP. Each time the virtual machine is run, CP provides the accumulated CPU time the virtual machine has used (VTIME) and a time-of-day (TOD) time stamp to be used in subsequent accounting calculations. Programs that are running in the virtual machine may use the timing information to calculate the amount of processor time used by each job.

XC virtual machines in access register mode cannot execute DIAGNOSE code X'70'.

Note: For more information, see *IBM ESA/390 Principles of Operation*.

Entry Values:

Rx
is the guest real address of a 16-byte storage area in the host-primary address space to be used as a communication area (CA). The storage area must be aligned on a doubleword boundary, must not cross a 4K-byte boundary, and must be in the virtual machine's real storage, preferably in page zero. Page zero is preferred because page zero is in storage for the user, and the communication-area page must be locked. Thus when page zero is used, CP does not have to lock an additional page.

Key-controlled protection and low-address protection do not apply to accesses to the communication area.

If Rx contains the value X'FFFFFFFF', the Time-of-Day clock accounting interface for the issuing virtual CPU is reset. If the interface was not enabled, no action is taken.

Ry
Not used.

After a virtual machine issues a DIAGNOSE code X'70' to enable the TOD clock accounting interface, CP updates the communication area each time the virtual machine is run. The first 8 bytes of the communication area contain the total CPU time accumulated to date by the virtual CPU while running in interpretive-execution mode (CA.VTIME) up to the guest TOD clock time (CA.TOD) in the next 8-byte field. The difference between any current value of the guest TOD clock and the value of the guest TOD clock stored in the communication area (CA.TOD) represents the current value of the delta VTIME accumulated by the guest in addition to the VTIME stored in the communication area (CA.VTIME). Programs running in the virtual machine should not change the communication area.

DIAGNOSE code X'70' may not be issued in access-register mode in an XC virtual machine.

Usage Notes

1. In a virtual machine with multiple virtual CPUs, the guest - typically a system control program (SCP) - must use a separate communication area for each of its CPUs. If the guest runs in an attached-processor (AP) or a multiprocessor (MP) configuration, the SCP has to issue DIAGNOSE code X'70' from each virtual CPU, identifying a different communication area.
2. When the wait-state interpretation capability is specified for the guest by CP, the guest remains dispatched, even when it enters the PSW enabled-wait state. In this case, the time spent dispatched in wait state is included in the virtual CPU time reported for the guest in the accounting communication area (CA.VTIME).
3. To avoid receiving widely differing results, care should be taken when using the information returned by DIAGNOSE code X'70' to analyze program performance of small portions of code. Processor performance characteristics, such as the cache storage size, influence the information returned. For example, the number of references to data which is not currently in cache may affect the amount of virtual time reported by DIAGNOSE code X'70'. Cache storage size and other processor performance characteristics may vary between processor models. Refer to the processor characteristics documentation for your hardware.
4. CP uses the real processor's CPU Timer value to maintain the communication area VTIME value. Although the CA values are accurate, when calculating consumed CPU time between CP updates to the Communication Area, it is important to remember that ESA/390 architecture does not require that the CPU Timer value be decremented each time the CPU's TOD clock advances. The CPU Timer may not count the time during which the CPU is not executing instructions. Thus, the difference between a current value of the guest TOD clock and CA.TOD may not be entirely reflected in CA.VTIME the next time the CA values are updated. This could result in calculated VTIME values that are slightly larger than actual consumed processor time at that moment. With extremely frequent sampling, the VTIME that is computed with the following algorithm may appear to regress slightly.

   Programmers who need to calculate an exact value should consider using the guest CPU Timer instead of DIAGNOSE code X'70'. See *IBM ESA/390 Principles of Operation* for details on CPU Timer and CPU TOD Clock operation.

Tutorial Notes: The following notes are tutorial information, not requirements of the DIAGNOSE code X'70' interface:

1. To use the information that CP has stored in the communication area for accounting purposes, perform the following steps:
   a. Preserve the TOD value in the communication area (CA.TOD) by storing a copy of it in your storage or registers.
   b. Get the current TOD by issuing the STORE CLOCK (STCK) instruction.
   c. Subtract the TOD in the communication area from the current TOD obtained in step “1.b” on page 81. This difference is the amount of processor time the virtual machine has used since it was last run (delta.VTIME).
   d. Add the accumulated processor time that is stored in the communication area (CA.VTIME) to the result obtained in step “1.c” on page 81. The result is the total amount of processor time the virtual machine has used up to the present time.
   e. Ensure that the TOD value stored in the communication area (CA.TOD) has not changed since step “1.a” on page 81 was performed, that is, the TOD in the communication area (CA.TOD) must be equal to the copy obtained in step “1.a” on page 81. If it has changed, repeat the procedure from step “1.a” on page 81.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'70' is given incorrect data:

Problem Encountered	Cause
Special-operation exception	DIAGNOSE code X'70' cannot run in an XC virtual machine that is in access register mode.
Specification exception	Any of the following: • The communication area is not aligned on a doubleword boundary. • The communication area crosses a page boundary. • The TOD clock accounting interface is already active for the virtual CPU using a different communication area.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store into the communication area; key-controlled protection and low-address protection do not apply.

DIAGNOSE Code X'74' – Saving and Loading an Image Library File

Privilege Class: A, B, C, or E

Addressing Mode: 24-bit or 31-bit

Entry Values:

Rx
Contains the first 4 characters of the image library file name. Rx cannot be register 15.

Rx+1
Contains the last 4 characters of the image library file name, where the name is left-justified and padded with blanks.

Ry
Contains the guest real address of the start of the data area. The area must start on a page boundary. Accesses to the data area are not subject to key-controlled protection or low-address protection.

Ry cannot be register 15.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the image library file. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the image library file is in the host-primary address space.

Ry+1
Bits 0-7 contain a subcode for either loading or saving of the image library file. Bits 8-31 contain the length of the image library data in bytes. CP rounds the length up to a multiple of 4K bytes, and saves or loads the resulting number of bytes. Partial pages are not saved or loaded.

The following subcodes are bits 0 through 7 of the Ry+1 register.

Subcode	Function	Description
X'00'	LOAD	Loads an image library into guest real storage from an image library file.
X'04'	SAVE	Saves an image library from the guest's real storage area into an image library file.

Usage Notes

1. If the number of bytes to save (in Ry+1) is specified as zero, the image library is not created. If an image library with the same name already existed, it will be unchanged.
2. Using VMRELOCATE for a virtual machine that might be using this function is not recommended and may have undesired results. If a virtual machine reads an Imagelib and is then relocated to another system, any subsequent save of that Imagelib will be written out to a different system, not the one from which it was read.

Responses

Condition Codes for a Normal Exit: Upon a normal exit condition, you receive a condition code of 0 or 1, indicating that the image library file was saved or loaded successfully. You also receive a return code in Ry. Refer to the following chart for the meaning:

Function and Condition Code	Return Code in Ry	Meaning
LOAD, CC=0	0 (X'00')	An image library file was loaded successfully.
SAVE, CC=0	0 (X'00')	The new image was saved successfully.
SAVE, CC=1	0 (X'00')	An image file of the same name has been replaced.

Condition Codes for an Exit with Error: Upon finding an error condition, you receive a condition code of 3, indicating that the image library file was not saved or loaded successfully. You will also receive a return code in Ry. Refer to the following chart for the meaning:

Function and Condition Code	Return Code in Ry	Meaning
LOAD, CC=3	4 (X'04')	Image library file not found
LOAD, CC=3	20 (X'14')	The image length specification is greater than the actual size of the image. The residual byte count is in Ry+1.
LOAD, CC=3	24 (X'18')	Paging or spooling error
LOAD, CC=3	28 (X'1C')	The image length specification is less than the actual size of the image. The residual byte count is in Ry+1.
LOAD, CC=3	36 (X'24')	Abnormal termination occurred; an NSI001 soft abend generated.
LOAD, CC=3	40 (X'28')	Invalid file name specified
SAVE, CC=3	8 (X'08')	Image library file currently active
SAVE, CC=3	24 (X'18')	Paging or spooling error
SAVE, CC=3	32 (X'20')	Spool space full
SAVE, CC=3	36 (X'24')	An abnormal termination occurred; an NSI001 soft abend generated
SAVE, CC=3	40 (X'28')	Invalid file name specified
SAVE, CC=3	44 (X'2C')	No spool file IDs available

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'74' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • R15 is specified as Rx or Ry. • The address in Ry is not on a page boundary. • The register specified for Rx is the same as Ry. • The Rx and Ry registers are not at least two registers apart.
Privileged-operation exception	Any of the following:. • The guest does not have the correct privilege class (A, B, C, or E). • The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to: • Fetch the data area (SAVE function); Key-controlled protection does not apply. • Store into the data area (LOAD function); Key-controlled protection and low-address protection do not apply.

DIAGNOSE Code X'7C' – Logical Device Support Facility

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'7C' to create, and communicate with, logical 3270 terminals. DIAGNOSE code X'7C' provides the following functions:

- The INITIATE function opens a logical communication path between the calling virtual machine that issues the DIAGNOSE code (the DIAGNOSE code X'7C' guest) and CP. It causes a logical device to be created and the VM logo to be directed to it.
- The ACCEPT function collects output data that CP has directed to a logical device. It is invoked after the DIAGNOSE code X'7C' guest is notified through an external interrupt that output data is to be processed.
- The STATUS function notifies CP of the completion and ending status of an ACCEPT function for logical printers.
- The PRESENT function passes input data to CP. The location of the data is described by an address or a complemented address. If a complemented address is used, it is the address of a list that describes a data stream occupying multiple data buffers.
- The TERMINATE function notifies CP to delete a specific logical device. If the logical device is the console of a virtual machine, the virtual machine is placed in FORCE DISCONNECT state. If the logical device is dialed to a virtual machine, the dialed connection is reset. If the logical printer is attached to a virtual machine, it is detached from that virtual machine. If an input or output operation is being processed, it is terminated with a unit check and intervention required.

Entry Values:

Rx

Is any user-specified register except R15 that contains the logical device address used to coordinate CP and virtual machine operations.

If a specific address is requested for the INITIATE function, the low-order two bytes of Rx specify the device address. Addresses in the range X'0000' to X'nnnnn-1' are valid, where nnnnn is the hexadecimal representation of the maximum number of logical devices as specified on the SET

MAXLDEV command. The system default for nnnnn-1 is 4095, and thus the default range of valid addresses is X'0000' to X'0FFF'.

Rx+1

For the INITIATE function, Rx+1 contains the following information about the pseudo device to be created:

- The first byte indicates optional features:
  - Bit 0 — 3270 extended features are to be supported (not valid for 3277s or 328Xs).
  - Bit 1 — the ACCEPT function must be followed by STATUS function (must be on for logical printers).
  - Bit 2 — the specific device address is requested.
  - Bit 3 — an IP address is supplied in Ry+1.
  - Bit 4 — Ry+1 points to a 16-byte field containing an IPv6 address. If bit 3 is on, bit 4 is ignored.

    Rule: The 16-byte field must not cross a page boundary.
- The second byte contains the model number: X'y0' through X'y5' (the first four bits (y) of the model number are ignored).
- The third byte contains the device class and must be X'40' (graphic).
- The fourth byte indicates the device type:
  - X'01' - 3278 or 3279 (3270 family displays except the 3277)
  - X'02' - 328X (3270 family printers)
  - X'04' - 3277 display

The following constitute valid model, class and type input. (The first four bits, (y), of the model number are ignored. Also, X'y0' is accepted, but treated as model X'y2'.)

```
3277    y04004
3277    y24004
3278/9  y24001
3278/9  y34001
3278/9  y44001
3278/9  y54001
328X    y04002
328X    y14002
328X    y24002
328X    y34002
328X    y44002
```

For the ACCEPT function, Rx+1 is a register that contains the address of a data buffer. Zero may not be used as the data-buffer address.

For the PRESENT function in buffer format, Rx+1 is a register that contains the address of the data buffer.

For the PRESENT function in a list format, Rx+1 is a complemented address of a list of buffer entries. For more information on this format see “PRESENT: DIAGNOSE code X'7C' subcode X'00000003'” on page 90. If this form of PRESENT ends with condition code 0, return code 3, an external interrupt is reflected when processing of the data is complete.

For a PRESENT from a printer, if Rx+1 is zero, an asynchronous device end interrupt is reflected. For a PRESENT from a non-printer device, Rx+1 may not be zero.

For the STATUS function, Rx+1 contains the sense data in the low order byte of the register.

Ry

Is any user-specified register except R15 that contains one of the following subcodes:

Subcode
Function

X'00000001'
INITIATE

X'00000002'
ACCEPT

X'00000003'
PRESENT

X'00000004'
TERMINATE

X'00000005'
TERMINATE (All)

X'00000006'
STATUS

Ry+1
For the INITIATE function, Ry+1 is a register that may contain the IP address associated with the logical device. This information is recorded and is used in appropriate command responses, operator messages, accounting records, and access control interface parameter lists.

For the ACCEPT and PRESENT functions, Ry+1 is a register that contains the length of the data buffer. For the BUFFER form of the PRESENT function, the buffer length cannot exceed 4096 bytes. For ACCEPT and for the BUFFER form of PRESENT, the length must be greater than zero. For the LIST form of the PRESENT function, this register is not used to contain the length; bits 1-31 of Ry+1 are ignored in this case. The contents of Ry+1 are also ignored when PRESENT is issued with zero in Rx+1 to generate an asynchronous device end from a printer.

PRESENT
When bit 0 is set to 1, the data is from a Read-Buffer request; when bit 0 is set to 0, the data is from a Read-Modified request.

ACCEPT
When bit 0 is set to 1, move partial data if the buffer is too short (Partial ACCEPT); when bit 0 is set to 0, move all data; if all data cannot be moved, the issuer receives a condition code of 1, and Ry +1=3; the size of the buffer needed is returned in Ry.

DIAGNOSE code X'7C' may not be issued in access-register mode in an XC virtual machine. All addresses passed on DIAGNOSE code X'7C' are guest absolute addresses in the host-primary address space. Key-controlled protection and low-address protection do not apply to storage references by DIAGNOSE code X'7C'.

Exit Values: On a successful return from an INITIATE function, Rx contains the logical device address assigned to the newly created logical device. This logical device address is returned by the INITIATE function, and must be specified by the user for an ACCEPT, PRESENT, TERMINATE, or STATUS function.

On return from an ACCEPT function, Ry contains the length of the data transferred. The maximum buffer length for an ACCEPT is X'7FFFFFFF'.

On completion of any of the DIAGNOSE functions, Ry+1 contains the return code.

Usage Notes

1. Even if the input is an address, z/VM uses all 32 bits of the Rx and Ry input registers.
2. DIAGNOSE code X'7C' ACCEPT sets a return code of 0 whether the CCW is command chained. It also issues an external interrupt subcode, X'02', for each new command chained CCW.

Responses

Condition Codes and Return Codes: One of the following condition codes is returned from the DIAGNOSE code X'7C' call. The codes are:

0 — Function completed with no errors
1 — Error condition

2 — Busy condition
3 — Device addressing error

In addition to the condition codes, a return code that indicates the specific cause of the problem is returned in Ry+1 as follows:

Condition Code	Return Code in Ry +1	Meaning
0	0 (X'00')	Normal completion
0	1 (X'01')	(ACCEPT) Another ACCEPT is required for another Write data stream.
0	2 (X'02')	(ACCEPT) Another ACCEPT is required for the next segment of the current data stream.
0	3 (X'03')	(PRESENT) An external interrupt is presented when processing of the data is finished. Note that the issuer may receive an external interrupt other than the one that indicates that the PRESENT is completed. If this occurs, then the issuer will not receive the interrupt specifically indicating that the PRESENT is complete.
0	4 (X'04')	(ACCEPT) A STATUS function must be issued to end the ACCEPT.
1	1 (X'01')	An invalid function code is in register Ry.
1	2 (X'02')	• (ACCEPT) no data available • (STATUS) The logical device is not waiting for a status function.
1	3 (X'03')	• (ACCEPT) The buffer was too short. No data was transferred. • Another ACCEPT is required to retrieve the data. • The required data length is in register Ry, unless data-chained CCWs are being used. If data-chained CCWs are being used, the byte count returned in Ry only contains the amount of data up to the CCW that exhausted the ACCEPT buffer and may not be the total data amount.
1	4 (X'04')	(ACCEPT, PRESENT) The buffer specification is in error: • (PRESENT) The length is greater than 4096 bytes. • (ACCEPT) The length is zero. • (PRESENT) The length is zero (length is not checked for an asynchronous device end request on a logical printer). • The address is not in the user's address space. • A paging I/O error occurred.
1	9 (X'09')	(INITIATE) The maximum logical devices have already been created.
1	10 (X'0A')	(ACCEPT, PRESENT) The logical channel detected an error.
1	11 (X'0B')	User not authorized for DIAGNOSE code X'7C'.
2	1 (X'01')	(PRESENT) CP has pending data that must be accepted first. The PRESENT function was not performed.

Condition Code	Return Code in Ry +1	Meaning
2	2 (X'02')	(PRESENT) The previous DIAGNOSE code X'7C' is not completed. The current PRESENT function was not performed. An external interrupt is reflected to the virtual machine.
2	3 (X'03')	(PRESENT) CP has an active Read-Buffer request outstanding, and this PRESENT is for a Read-Modified request. The PRESENT function was not performed.
2	4 (X'04')	(PRESENT) This PRESENT indicates Read-Buffer request data, and the READ is for a Read-Modified request. The PRESENT function was not performed.
2	5 (X'05')	(PRESENT) CP is waiting for a STATUS request. The PRESENT function is not performed.
3	2 (X'02')	The specified logical device does not exist.
3	3 (X'03')	• (INITIATE) The device class, type, or model is invalid. • (STATUS) The STATUS is not valid for this logical device; the original INITIATE did not have bit 1 on in Rx+1.
3	4 (X'04')	(INITIATE) The specific device address is invalid.
3	5 (X'05')	(INITIATE) For a specific device request, the device was already created.
3	6 (X'06')	(INITIATE) The IPv6 address pointer is not valid or the data crosses a page boundary.

Program Exceptions: These program exceptions can occur if the DIAGNOSE X'7C' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Either of • An incorrect privilege class • Rx or Ry specified as register 15.
Privileged-operation exception	The virtual machine is in the problem state.
Special-operation exception	DIAGNOSE code X'7C' cannot run in an XC virtual machine that is in access register mode.

Logical Device External Interrupt Code X'2402'

The logical device support uses a special external interrupt code to notify the DIAGNOSE code X'7C' guest of a change in status for a specific logical device. The external interrupt code is X'2402'. This interrupt causes a fullword of data to be stored at location X'80' in the DIAGNOSE code X'7C' guest's virtual machine. The interrupt is masked on and off by bit 22 of control register zero.

The format of the stored fullword is:

Address
Contents

80-81
Logical device number

82
Flag byte

83
External interrupt reason code

The **flag byte** contains the following status flags:

Bit
Meaning

0
Data from the last PRESENT was discarded by the system (the subsequent I/O was a Write instead of a Read).

1
A data transfer error occurred on the previous PRESENT (LIST form of the PRESENT).

The logical device external interrupt reason codes stored in location X'83' are:

Code
Meaning

01
CP is terminating the logical device.

02
A Write has been issued to the device (an ACCEPT must be done).

03
A previous PRESENT is now finished. (User received CC=0 and RC=3, or CC=2 and RC=2 after a PRESENT)

04
A Read Buffer command has been issued to the device.

05
A Read Modified command has been issued to the device.

Logical Device Support Facility Functions

The following logical device functions manage communications and the transfer of data between CP and the DIAGNOSE code X'7C' guest.

INITIATE: DIAGNOSE code X'7C' subcode X'00000001'

The INITIATE function opens a logical communications path between the DIAGNOSE code X'7C' guest and the z/VM Control Program. It causes a logical device to be created, initialized with device, model, and type information, enabled, and the z/VM logo to be directed to it. During the enabling process, if the device supports extended features, a WSFQ (Write Structured Field Query) is issued to the device to determine its characteristics, and the Reply information used to update the device characteristics of the logical device, and also saved to be used to respond to future DIAGNOSE code X'8C' requests issued by the logical device user's virtual machine.

Rx+1 must contain the model number in byte 2, and the device class and type in bytes 3 and 4.

Rx is used only if a specific logical device is requested: bit 2 of Rx+1 is on. If so, Rx contains the logical device number.

If a specific address is not requested, then CP assigns the next available address to the new logical device. When more than one logical device will be created with specific device addresses requested, it is recommended that these devices be created in the range of X'0F80' to X'0FFF'. This is because logical devices will not be created from this range unless there is a specific request from this range or there are no other logical device addresses available. Reserving this range for specific logical device requests minimizes the possibility of finding a device address already in use.

Ry+1 is used only if an IP address is supplied: bit 3 of Rx+1 is on. If so, Ry+1 contains the IP address to be associated with the logical device.

On completion of the INITIATE, the newly assigned logical device address is placed in Rx. This value is used on subsequent DIAGNOSE operations to indicate the logical device being used. This address is also provided with any external interrupts for the device so that the DIAGNOSE code X'7C' guest can associate the interrupt with a specific logical device.

ACCEPT: DIAGNOSE code X'7C' subcode X'00000002'

The ACCEPT function collects data that CP has directed to a logical device. It is invoked after the DIAGNOSE code X'7C' guest is notified through external interrupt that output data is to be processed. Upon invocation, Rx+1 must contain the data buffer address and Ry+1 the buffer length. If the data buffer supplied was too short to contain the data, the function returns the required buffer size in Ry and no data is moved. This action can be overridden by setting an indicator in the length register (bit zero in Ry+1 set to 1) when the function is invoked. In this case, the data **is** moved to the short buffer and a condition code of 0 and a return code of 2 is sent. The system moves the next portion of the data on the next ACCEPT. Upon successful completion of function processing, the data length is returned in Ry, and the data buffer contains the CCW OP code in its first byte (only the first time if several operations are required to collect all the data from that CCW) and data in the remaining buffer space.

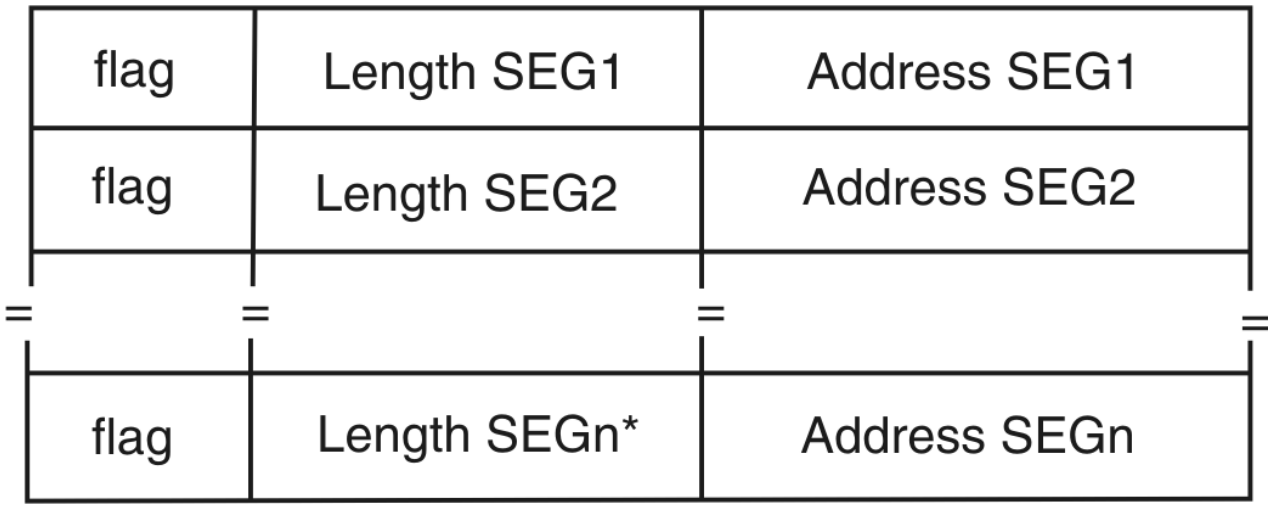
STATUS: DIAGNOSE code X'7C' subcode X'00000006'

The STATUS function allows status to be returned to CP after an ACCEPT function is performed. It must be used with a logical 328x printer to indicate when the printer has completed the printout and the ending status of the printer.

PRESENT: DIAGNOSE code X'7C' subcode X'00000003'

The PRESENT function passes input data to CP. The location of the data is described by an address or a complemented[1] address in Rx+1. If the register contains an address, it is the address of a data buffer 4096 bytes or less in length. In this case, Ry+1 contains the length of that data buffer. If Rx+1 contains a complemented address, it is the address of a list that describes a data stream occupying multiple data buffers and/or greater than 4096 bytes in length. In this case, Ry+1 is not used to describe the data length. However, in either case, a high-order bit of 1 in Ry+1 indicates whether the data is the result of a Read Buffer or Read Modified CCW.

If a list describes the data, the list must be in the format:



The list must start on a fullword boundary. Each entry consists of a flag byte, a 3-byte unsigned length field and a 4-byte buffer address field that describe the length and location of sequential segments of a data stream. Bit X'80' set to 1 in the flag byte marks the last entry; other bits are reserved and should be zeros.

A single entry list may be used to describe a single data buffer greater than 4096 bytes in length. Neither the list nor the data may be modified before transfer of the data has completed.

[1] Refers to the two's complement of the address.

A PRESENT may be either solicited or unsolicited. If solicited, then a Read Buffer or Read Modified was directed by CP to the logical device and the DIAGNOSE code X'7C' guest is notified through an external interrupt of that fact. When the guest responds with a DIAGNOSE code X'7C' PRESENT, the input data is received immediately and the DIAGNOSE completes with a condition code of 0 (zero), and a return code of 0 (zero).

If unsolicited, processing differs depending on whether the PRESENT is buffer format or list format. If buffer format, then the input data is moved to a CP buffer awaiting a Read from CP, and the DIAGNOSE completes with a condition code of 0 (zero) and a return code of zero (0). When eventually a Read is processed by CP, the data saved in the CP buffer satisfies the Read. If list format, the data is left in its input list format, and the DIAGNOSE completes with a condition code of zero (0) and a return code of 3. Later, when a Read is processed by CP, and the input data received (that is, moved from the DIAGNOSE code X'7C' input buffers), the DIAGNOSE X'7C' guest is notified through an external interrupt that the previous PRESENT is now finished.

TERMINATE: DIAGNOSE code X'7C' subcode X'00000004'

The TERMINATE function causes CP to delete a specific logical device. If the logical device is the console of a virtual machine, the virtual machine is placed in Force Disconnect state. If the logical device is dialed, the connection is reset; if attached, it is detached from that virtual machine. If an input or output operation is being processed, it is terminated with a unit check and intervention required. The logical device is deleted, and its logical device address is made available again.

TERMINATE (All): DIAGNOSE code X'7C' subcode X'00000005'

The TERMINATE (all) function notifies CP to terminate all logical devices created by the DIAGNOSE code X'7C' guest.

DIAGNOSE Code X'84' – Directory Update-in-Place

Privilege Class: B

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'84' enables a class B user to replace certain data in any user, identity, or subconfiguration stanza of the CP object directory. The user must specify the stanza and can replace the following data:

- The logon password
- The default virtual machine storage size
- The maximum virtual machine storage size
- Privilege classes
- Logical editing symbols
- The initial program load (IPL) system
- IPL parameter data
- The account number
- The distribution code
- User options
- The minidisk access mode
- The minidisk read, write, or multiple password
- The minidisk device type, allocation definition, and volume serial.
- Options of the SCREEN directory control statement
- XAUTOLOG user IDs
- The type of virtual machine (XA, ESA, XC, Z)
- The maximum number of virtual processors

- The maximum number of spool files
- Virtual processor ID and VECTOR and CRYPTO option values
- User's default date format setting.

See the individual operation descriptions in Table 6 on page 93 to understand when the results of each operation take effect.

DIAGNOSE code X'84' can neither add new entries to, nor delete existing entries from, the object directory. It can only replace existing object directory data. This DIAGNOSE instruction has no effect on definition control statements in the CP source directory.

In SSI-enabled directories, the changes to identity and subconfiguration object directory entries are applied as if the changes are made to the statements in the corresponding stanza in the source directory. Because settings specified in a subconfiguration stanza can override settings in the associated identity stanza, changes made using DIAGNOSE X'84' to an identity object directory entry might not appear to have any affect if overridden by settings specified in the subconfiguration stanza.

Object directories for SSI-ready directories do not contain separate entries for identity and subconfiguration stanzas specified in the source directory. Rather, when the object directory is built, the identity stanza and the single associated subconfiguration stanza are merged into a single entry that appears as if it was created from a single user stanza. Therefore, changes to the object directory for identity and subconfiguration stanzas are made specifying the user ID from the IDENTITY statement.

For more information on the directory entries, see *z/VM: CP Planning and Administration*.

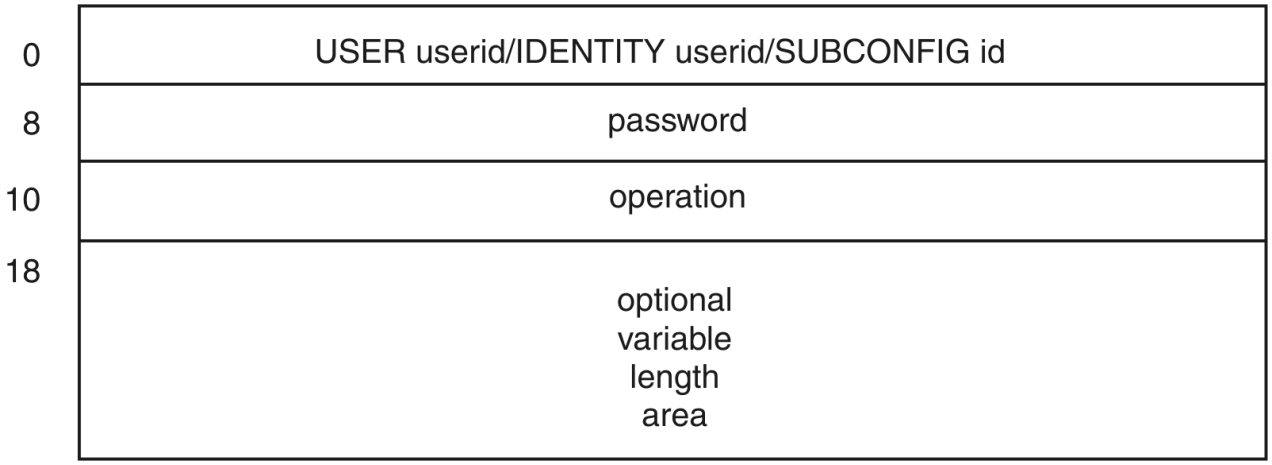
Entry Values:

Rx
must contain the guest real address of a variable-length parameter list.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

Ry
must specify, in bytes, the length of the parameter list.

The parameter list must contain at least a fixed-length area 24 bytes long, followed by an optional variable length area of up to 223 bytes and a 4-byte fence (X'FF') or two sequential blanks. Data in the fixed-length area identifies the directory stanza to be updated, the password of the user ID associated with the stanza, and the data field to be replaced in the directory stanza. All entries in the parameter list must contain unpacked, EBCDIC data.



The parameter list is organized as follows:

Fixed-length area

USER *userid*/IDENTITY *userid*/SUBCONFIG *id*
is the USER or IDENTITY user ID, or the SUBCONFIG ID whose stanza is updated. This is a 1- to 8-character, left-justified value and must be padded with blanks with no imbedded blanks.

password
is the current CP password of the USER, IDENTITY, or SUBCONFIG whose stanza is updated. This is a 1- to 8-character, left-justified value and must be padded with blanks with no imbedded blanks. For a subconfiguration stanza, specify the password of the associated IDENTITY.

If this field is blank, the update-in-place function is processed in *test mode* and the directory is not updated. Test mode lets you check the syntax of the directory operations without modifying the CP user directory or any operational controls.

operation
is a 1- to 8-character, left-justified value that identifies the data in the stanza that is to be replaced. Valid values and the data that each identifies for replacement are defined in the description of the variable-length area that follows.

Variable-length area

Table 6 on page 93 shows, for each value of the operation field, the data that must be in the variable-length area of the parameter list, and the format and characteristics of the data. Table 6 on page 93 also shows specific information concerning all DIAGNOSE code X'84' operations.

Table 6. DIAGNOSE Code X'84' Operations

Operation	Description
ACCOUNT	Replaces all of the user's accounting numbers, up to eight, to be effective at the next user logon. The ACCOUNT operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. *aaaaaaa*1[*aaaaaaa*2[*aaaaaaa*3...[*aaaaaaa*8]]] ***aaaaaaa*1...*aaaaaaa*8** is up to eight account numbers, 1- to 8-character values, left-justified and padded with blanks. These replace all account numbers, so include the current numbers along with any changes, remembering that the first in the list is the primary or logon account number. The list is terminated by a blank account number or X'FFFFFFFF'.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
CPU	Replaces attributes of a user's virtual processor, to be effective at the next user logon. The CPU operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. *pp* [VECTOR|NOVECTOR] [CPUID *bbbbbb*][CRYPTO|NOCRYPTO] ***pp*** is a 2-digit hexadecimal virtual processor address between X'00' and X'3F' followed by a blank and one of the following: **VECTOR|NOVECTOR** designates the specified processor as having or not having VECTOR capability. **CPUID *bbbbbb*** assigns the value following as the processor's CPUID where *bbbbbb* is a valid hexadecimal number. **CRYPTO|NOCRYPTO** designates the specified processor as having or not having CRYPTO capability. ***Usage Notes:*** 1. This operation can be used only to alter virtual processors defined by a CPU directory statement. 2. If you use DIAGNOSE code X'84' to add a VECTOR designation to a CPU, it will have no effect if the user has specified NOVF on the OPTION directory control statement in the directory stanza. 3. You can add a CRYPTO designation to a virtual CPU only if the user has a CRYPTO directory statement in the directory stanza.
DATEFMT	Replaces the user's default date format setting, to be effective at the next user logon. [SHOrtdate|FULldate|ISOdate|SYSdefault] **SHOrtdate|FULldate|ISOdate|SYSdefault** sets the virtual machine's default date format to the specified setting.
DISTRIB	Replaces the user's distribution identification value, to be effective at the next user logon. The DISTRIB operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. *dddddddd* ***dddddddd*** is the distribution identification word, a 1- to 8-character value, left-justified and padded with blanks.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
EDITCHAR	Replaces the user's logical editing symbols for terminal and command analysis, to be effective at the next user logon. The EDITCHAR operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. *symbols* ***symbols*** is an 8-byte value containing logical editing symbols. The first four bytes, counting from the left, are line edit symbols. The first or high-order byte is the line-end symbol (#), the second byte is the line-delete symbol (¢), the third byte is the character-delete symbol (@), and the fourth byte is the escape-character symbol ("). All existing symbols in the directory are replaced. Therefore, specify existing symbols that are to be retained as well as symbols that are to be changed. Unspecified symbols must contain blanks. The last four bytes of the 8-byte value are reserved for future IBM use. ***Usage Note:*** You cannot use any of the letters A through Z, the numbers 0 through 9, or the bytes X'0E' (shift out) or X'0F' (shift in) as a logical line edit symbol.
IACCOUNT	Forces the creation of an accounting record for the current account number and replaces the primary account number in the directory stanza in the object directory, the new account number being effective immediately. The IACCOUNT operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. *aaaaaaaa* ***aaaaaaaa*** is a 1- to 8-character account number, left-justified and padded with blanks. The number is assumed to be a valid value in the customer accounting scheme. ***Usage Note:*** If the new account number specified is not one of the alternate account numbers presently in the directory stanza of the object directory, then the primary account number is simply replaced and is not preserved as a valid alternate account number.
IPL	Replaces the user's automatic IPL statement, to be effective at the next user logon. *name*|*aaaa* [PARM [*ddddd...ddddd*] ***name|aaaa*** is a 1- to 8-character system name or virtual device number, left-justified and padded with blanks. **PARM** is a keyword indicating that data is to follow. ***ddddd...ddddd*** is up to 48 characters of variable data. ***Usage Note:*** All existing values are replaced in the directory stanza; therefore, specify values that are to be retained as well as values that are to be changed. Trailing blanks are not truncated, but passed.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
LOGPASS	Replaces the password of the specified user ID, to be effective at the next user logon. *password* The LOGPASS operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. ***password*** is a new 1- to 8-byte logon password, left-justified and padded with blanks, starting in column 1 of the variable length area.
MACHINE	Replaces the user's virtual machine data, to be effective at the next user logon. The MACHINE operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. [XA|ESA|XC|Z] [MAXCPU *nn*] **XA|ESA|XC|Z** sets the default virtual machine type designation. **MAXCPU *nn*** sets the maximum number of virtual processors that the virtual machine can have. The variable *nn* is a number between 1 and 64. ***Usage Notes:*** 1. Use either or both of the above option values in any order, separated by a single blank. 2. The list is terminated by two consecutive blanks or the string X'FFFFFFFF'.
MAXSTOR	Replaces the user's maximum virtual machine storage size. The new maximum storage size is enforced at the time of the next DEFINE STORAGE or user logon. Changing the maximum storage size does not cause any change to the currently defined storage size of the target user even if the new maximum is smaller than the user's currently defined storage size. *nnnnnnnnu* ***nnnnnnnnu*** is the maximum virtual machine storage size, a 1- to 8-byte left-justified decimal value followed by a 1-character storage unit suffix. See the STORAGE operation for the list of storage unit suffixes, maximum input values, and usage notes.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
MDISK	Replaces the default access mode and password definitions for the specified MDISK address, which must already exist in the directory stanza, the change to be effective immediately. The MDISK has two formats of parameter lists based on the length of the device address and the access mode of the device definition. Format 1 is the preferred format. Format 2 is for VM/SP, VM/SP HPO, and VM/ESA (370 Feature) compatibility. Format 1: {VDEV *aaaa mmmm* [*readpass* [*writepas* [*multipas*]]]} Format 2: {*aaammm*[*readpass*][*writepas*][*multipas*]} **VDEV** indicates that this is a format 1 parameter list. ***aaaa/aaa*** is a 1- to 4-hexadecimal digit minidisk address. If less than four digits, the value will be padded to the left with 0's. ***mmmm/mmm*** The default access mode of the device definition. Valid access modes are R, RR, W, RW, M, MR, MW. These access modes may be concatenated with optional suffix letters. For Format 1 the allowed suffix letters are V, S, and E. They can be used in the following combinations: V, S, E, VS, or VE. When concatenated with a mode, some of the possible combinations would be RV, RRS, WE, MVE, or MRVS. For Format 2, the only suffix letter allowed is V. The mode is replaced completely so you must specify it exactly as you want it to appear. ***readpass writepas multipas*** The passwords are 1- to 8-character values that enable the user to LINK to a desired DASD in the appropriate mode. All of the passwords are replaced, so you must supply all of the values to be retained as well as those that are changed. Valid values for access mode and for passwords are defined in *z/VM: CP Planning and Administration*. ***Usage Notes:*** • Format 1 — A positional set of alphanumeric character strings, each value of which is separated from the previous by a single blank. The line is terminated by two or more blanks or a fence of X'FFFFFFFF'. The format is identified by the value *VDEV* starting in column 25, followed by a blank. The rest of the elements of the operation's invocation are in the following list. They are positional and separated by a single blank: – A hexadecimal 1- to 4-digit minidisk address. If less than four digits, the address is padded to the left with zeros. – A 1- to 4-character access mode – A 1- to 8-character read password – A 1- to 8-character write password – A 1- to 8-character multiple password • Format 2 — A 30-byte field, column-dependent format. All values must be left-justified and padded with blanks. The contents of the field are: – Bytes 1 through 3, counting from the left, specify a minidisk address. This is the minidisk whose mode and passwords are to be changed. – Bytes 4 through 6 specify the access mode. – Bytes 7 through 14 specify the read password. – Bytes 15 through 22 specify the write password. – Bytes 23 through 30 specify the multiple password.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
OPTIONS	Replaces the user's current set of OPTIONS definitions. The new options become effective the next time the user logs on. [Acct] [Cpuid *bbbbbb*] [Lang *Langid*] ** Values supported for compatibility ** [Realtimer] [Ecmode] [Isam] [Svcoff] [BMX] [AFfinity *xx*] **Acct** sets the account flag to enable the cutting of account records. The OPTION ACCT operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. **Cpuid *bbbbbb*** sets the default cpuid for the user ID to *bbbbbb*, where *bbbbbb* is a valid hexadecimal number. **Lang *langid*** sets the system message language ID value to *langid*. **Values supported for compatibility** are recognized, validated and ignored when used in combination with the valid ones. ***Usage Notes:*** 1. The user options field is an 80-byte, left-justified value padded with blanks. Specify each option as a character string with a blank character between options. 2. The options field must be followed by the value X'FFFFFFFF' or by at least two blanks. 3. For a description of each option and a list of valid values, see *z/VM: CP Planning and Administration*.
PRIORITY	Is accepted and ignored for compatibility with VM/SP HPO Release 5 and later. *pp* The PRIORITY operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. ***pp*** HPO dispatching priority is ignored.
PRIVLEGE	Replaces the user's current privilege class definition with a new set of classes. The PRIVLEGE operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. *c*[*ccccccccccccccccccccccccccccccc*] ***ccccccccccccccccccccccccccccccc*** is the privilege class parameter, a 32-byte value, where each byte represents a privilege class. Valid values for each byte are A through Z, and 1 through 6. All existing classes in the directory stanza are replaced. Therefore, specify existing classes that are to be retained, as well as classes that are to be changed. The data must be left-justified and padded with blanks. This field can be all blanks, in which case the specified virtual machine has the default classes (defined on the PRIV_CLASSES statement in the system configuration file) each time the user logs on. The new privilege classes are effective the next time the user logs on.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
RMDISK	Alters the minidisk extents and the volume serial number for a minidisk that currently exists in the user directory. The RMDISK update-in-place takes effect **immediately** for **all** subsequent *new* LINKs to the specified minidisk and for ALL *new* minidisks established by virtue of subsequent LOGONs. LINKs to the specified minidisk that were established prior to the RMDISK operation continue to use the *old* extents that were in effect at the time of that LINK. In this case, a DETACH followed by a re-LINK is performed (or LOGOFF followed by LOGON, which is in effect the same thing), is necessary to access the new extents. *aaaa ddddddddd rrrrrrrrr ccccccccc volser* ***aaaa*** is a 1- to 4-hexadecimal digit minidisk address. If less than four digits, the value is padded to the left with zeros. The hexadecimal address **must** be a currently existing minidisk for the specified user ID. ***ddddddddd*** is the device type of the real DASD. Select the appropriate value from the following: 3350, 3370, 3375, 3380, 3390, 9332, 9335, 9336, 9345 or FB-512. The 9345 must be formatted as an ECKD™ device. ***rrrrrrrrr*** is the cylinder (CKD/ECKD) or block (FBA) relocation factor identifying the beginning of the new allocation. It is the caller's responsibility to specify a cylinder or block location factor that is valid for the device type that has been specified. This field must be a 1-10 character decimal value. See the table of maximum minidisk sizes in the documentation on the MDISK directory statement in *z/VM: CP Planning and Administration*. ***ccccccccc*** is a decimal count of the number of cylinders (CKD/ECKD) or blocks (FBA) to be allocated or END which is defined as the remaining cylinders or blocks of the volume. It is the caller's responsibility to specify a cylinder or block count value that is valid for the device type that has been specified. This field must be a 1-10 character decimal value or END. See the table of maximum minidisk sizes in the documentation on the MDISK directory statement in *z/VM: CP Planning and Administration*. ***volser*** is a 1- to 6-character new volume serial identification. Any 1- to 6-character EBCDIC value, other than blank(s), is accepted. ***Usage Notes:*** 1. The variable-length area **must** contain **all** of the positional fields, each field separated from its predecessor by at least one blank. Trailing data, if any, that follows the **last** positional field (and its blank delimiter) is ignored. 2. All other fields associated with the minidisk (default link mode, passwords, minidisk and DASD options) remain unchanged. 3. When coding the device type value, a 3380C is treated as a 3380 if you are coming from a VM/SP, VM/SP HPO, or VM/ESA (370 Feature) system.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
SCREEN	Replaces the user's SCREEN color and highlight control options, to be effective at the next user logon. *ccccccchhhhhhhh...ccccccchhhhhhhh* ***ccccccchhhhhhhh*** are the display screen options, specified in an 80-byte area composed of ten doubleword fields. The ten fields are paired into five sets corresponding to the five display areas of the SCREEN. You must specify these areas in the following order: 1. CP output 2. VM output 3. Input redisplay 4. Input area 5. Status area. Each of the five doubleword sets has a color field and an extended highlight field (see the SCREEN option description in *z/VM: CP Planning and Administration* for valid color and extended highlight values). Within each doubleword set you must specify the color first followed by the extended highlight value. You must specify all fields, including those you do not want to change. Each of the options you specify must also be left-justified in its respective 8-byte field.
SPOOLF	Replaces the user's spool file characteristics. The new maximum number of spool files will be effective the next time the user logs on. MAXSPOOL *nnnn* **MAXSPOOL *nnnn*** sets the new maximum spool file number where *nnnn* is the maximum spool file number for this user. The number is between 1 and 9999.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
STORAGE	Replaces the user's default virtual machine storage size, to be effective at the next user logon. ***nnnnnnnnu*** is the virtual machine storage size, a 1- to 8-byte left-justified decimal value followed by a 1-character storage unit suffix. The value specified can be any value less than or equal to the user's current maximum storage value in the directory. **Storage units / Suffix (*u*)** **Maximum input value (*nnnnnnnn*)** **Kilobytes / K** 2096128 **Megabytes / M** 99999999 **Gigabytes / G** 99999999 **Terabytes / T** 16777216 **Petabytes / P** 16384 **Exabytes / E** 16 ***Usage Notes:*** 1. The K suffix is provided for upward compatibility only; a K specification is rounded up to a MB value. The maximum specification is 2096128K (2047 MB). 2. The maximum input value of 99999999 for the M or G suffix is not a size limit but the physical limit of the parameter (8 digits plus suffix). If the maximum input value for one of these suffixes does not allow you to define the amount of storage you want, you need to use a larger storage unit. 3. The maximum size you can specify is 16E (or 16384P or 16777216T), although the actual maximum size supported may be restricted by the model of the server where the directory is used. 4. An XC virtual machine can address up to 2047 MB of storage in its base address space. 5. Giving many virtual machines very large storage sizes might affect real storage availability. For each virtual machine, CP creates one or more segment tables in host real storage to represent the virtual machine storage: • For a virtual machine less than or equal to 512 MB, one frame is allocated for the segment table. • For a virtual machine larger than 512 MB but less than or equal to 1 GB, two contiguous frames are allocated for the segment table. • For a virtual machine larger than 1 GB but less than or equal to 1.5 GB, three contiguous frames are allocated for the segment table. • For a virtual machine larger than 1.5 GB but less than or equal to 2 GB, four contiguous frames are allocated for the segment table. • For a virtual machine larger than 2 GB, multiple segment tables are created, plus one or more higher level (region) tables are created to identify the segment tables. If needed, multiple levels of region tables are created. Each region table occupies 1-4 contiguous frames.

Table 6. DIAGNOSE Code X'84' Operations (continued)

Operation	Description
TACCOUNT	Forces the creation of an accounting record for the current account number. Then replaces the *active* account number with the new one specified in the parameter list. The user's definition in the object directory is left unchanged. The account number is temporary, terminated by a LOGOFF, by another TACCOUNT operation, an IACCOUNT operation, or a SET ACCOUNT command. *aaaaaaaa* ***aaaaaaaa*** is the account number, a 1- to 8-character value, left-justified and padded with blanks. The number is assumed to be a valid value in the customer accounting scheme.
XAUTOLOG	Replaces the list of class G user IDs authorized to XAUTOLOG this user, to be effective immediately. The XAUTOLOG operation can be performed on a user or identity stanza, but is not allowed on a subconfiguration stanza. [*uuuuuuu*1 [*uuuuuuu*2 ... [*uuuuuuu*8]]] **[*uuuuuuu*1 [*uuuuuuu*2 ... [*uuuuuuu*8]]]** lists up to eight user IDs, each a 1- to 8-character value, separated by single blanks. The list is terminated by two consecutive blanks or the string X'FFFFFFFF'. All XAUTOLOG user IDs are replaced, so specify user IDs that are to be retained as well as those that are to be changed.
XSTORE	Accepted for compatibility only. Expanded storage (XSTORE) is not supported. Specifying this operation has no effect.

Exit Values:

Ry
Contains 0 unless the condition code is 1, in which case the value is the appropriate return code shown in the following list.

Usage Note

Users defined with the LBYONLY operand in the password field of their USER statement in the CP directory may be restricted from performing functions that require password validation. A user ID defined with the LBYONLY operand cannot be the target of DIAGNOSE X'84' operations unless the virtual machine issuing the diagnose has the D84NOPAS option in its directory stanza and the operation specified is not LOGPASS or MDISK.

Responses

Condition Codes: Upon return from DIAGNOSE code X'84', CP sets one of the following condition codes:

Condition Code	Meaning
0	The object directory was updated.
1	DIAGNOSE code X'84' has detected an error. The directory is unchanged. The return code defines the error.
3	DIAGNOSE code X'84' has encountered a paging error while trying to access or update an object directory page. The directory remains unchanged. The return code defines the error.

Abend Codes:

Abend Code	Meaning
UDU001	An error has been detected in one of the directory blocks for the user.

Return Codes (in Ry):

Return Code	Meaning
0 (X'00')	The directory update was successful.
26 (X'1A')	The specified device address was not found.
28 (X'1C')	The value in the OPERATION field of the parameter list is invalid. Operations not allowed for subconfiguration stanzas are treated as invalid operations.
30 (X'1E')	The specified user ID could not be found.
31 (X'1F')	The password specified in the fixed-length area of the parameter list does not match the current password of the user ID being updated.
40 (X'28')	The storage size requested exceeds the maximum defined for this virtual machine.
41 (X'29')	The storage size requested exceeds the maximum allowed for the specified storage unit.
42 (X'2A')	The conversion of the storage size from alphanumeric to binary failed with a syntax error.
43 (X'2B')	The value specified for the virtual machine storage size is invalid.
53 (X'35')	The specified privilege classes contain an invalid character.
70 (X'46')	An option value specified to the OPTIONS operation is invalid.
72 (X'48')	The option value contains a syntax error or an invalid character.
80 (X'50')	The parameter list contains an invalid minidisk address.
81 (X'51')	The parameter list specifies an invalid access mode for a minidisk.
91 (X'5B')	No attributes were found on the SCREEN operation.
92 (X'5C')	Invalid attributes were found on the SCREEN operation.
101 (X'65')	The parameter list is too large, greater than the maximum of 255 bytes.
102 (X'66')	The length of the parameter list is less than 24 bytes.
110 (X'6E')	No parameter data currently exists in the directory.
111 (X'6F')	The parameter length is invalid.
121 (X'79')	The virtual address is invalid.
122 (X'7A')	This user has no minidisk defined at this virtual address.
123 (X'7B')	The specified device type is invalid.
124 (X'7C')	The specified starting cylinder or block is invalid.
125 (X'7D')	The specified number of cylinders or blocks is invalid.
126 (X'7E')	The specified volume serial number is invalid.
230 (X'E6')	An invalid option was specified for the SPOOLF operation.
231 (X'E7')	The MAXSPOOL value was missing.
232 (X'E8')	The MAXSPOOL value equals zero.

233 (X'E9')	The MAXSPOOL value contains an invalid EBCDIC value.
234 (X'EA')	The MAXSPOOL value was greater than 9999.
240 (X'F0')	The CPU specified was not found in the directory for this user.
241 (X'F1')	An invalid option was specified for the CPU operation.
242 (X'F2')	The specified CPU address is invalid.
243 (X'F3')	The CPUID value contains an invalid EBCDIC value.
244 (X'F4')	The CPUID value was greater than six bytes long.
245 (X'F5')	The CPUID value was missing.
246 (X'F6')	CRYPTO was specified for the CPU operation but the user does not have a CRYPTO directory control statement in their user definition.
250 (X'FA')	Too many user IDs were specified for the XAUTOLOG operation.
251 (X'FB')	An invalid user ID was specified for the XAUTOLOG operation.
260 (X'104')	The MAXCPU value was missing.
261 (X'105')	An invalid option was specified for the MACHINE operation.
262 (X'106')	Conflicting values were specified for the MACHINE operation.
263 (X'107')	The MAXCPU value contains an invalid EBCDIC value. It was not in the range of 1 to 64.
264 (X'108')	The XC virtual machine type conflicts with the V=R/V=F option for this virtual machine. V=R and V=F are not supported.
290 (X'122')	An option is missing for the DATEFMT operation.
291 (X'123')	An invalid option was specified for the DATEFMT operation.
292 (X'124')	More than one option was specified for the DATEFMT operation.
410 (X'19A')	An error occurred while writing a directory page out to a direct access device. (See following note.)
475 (X'1DB')	An error occurred while reading a directory page. (See following note.)
550 (X'226')	An invalid symbol was specified for the EDITCHAR operation.
1507 (X'5E3')	The active CP directory is invalid.

Note: To update the directory, use the service program (DIRECTXA) described in *z/VM: CP Planning and Administration*. For more information on formatting the DIRECTXA utility, see *z/VM: CP Commands and Utilities Reference*.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'84' is given incorrect input data:

Problem Encountered	Cause
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state. • The user does not have the appropriate privilege class.
Access exception (See "Access Exceptions" on page 8.)	An error occurred trying to fetch the parameter list.

DIAGNOSE Code X'88' – Validate User Authorization/Link Minidisk

Privilege Class: G

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'88' is used by a service virtual machine to authenticate a client agent's access to a target virtual machine's minidisks and to gain access to those minidisks on the client's behalf. If an external security manager (ESM) is not controlling the use of this diagnose instruction, the issuing virtual machine must have the DIAG88 option in its User Directory entry. When under ESM control, special authorization is required to use this function. Contact your administrator to obtain any needed permissions. This capability should be granted only to trusted programs.

An application may verify its ability to use DIAGNOSE code X'88' through use of subcode negative 1 (-1). When an application has the ability to use DIAGNOSE code X'88', it is assumed that the issuing virtual machine will first use subcode X'00' or subcode X'08' to validate that a client is permitted to access the resources of a target virtual machine. After performing this validation, the issuing virtual machine may then use subcode X'04' to obtain access to the target machine's minidisks on behalf of the client.

An agent may be granted access to a target machine's minidisks if the agent's logon password is known and either the agent is the target or is authorized in the User Directory to access the target via LOGONBY. In other words, if a client knows how to log on to the target virtual machine, either directly or using LOGONBY, it may be granted access to that machine's minidisk resources.

CMS provides simplified programming access to DIAGNOSE code X'88' through use of the DMSPASS callable service. For more information, see DMSPASS callable service in *z/VM: CMS Callable Services Reference*.

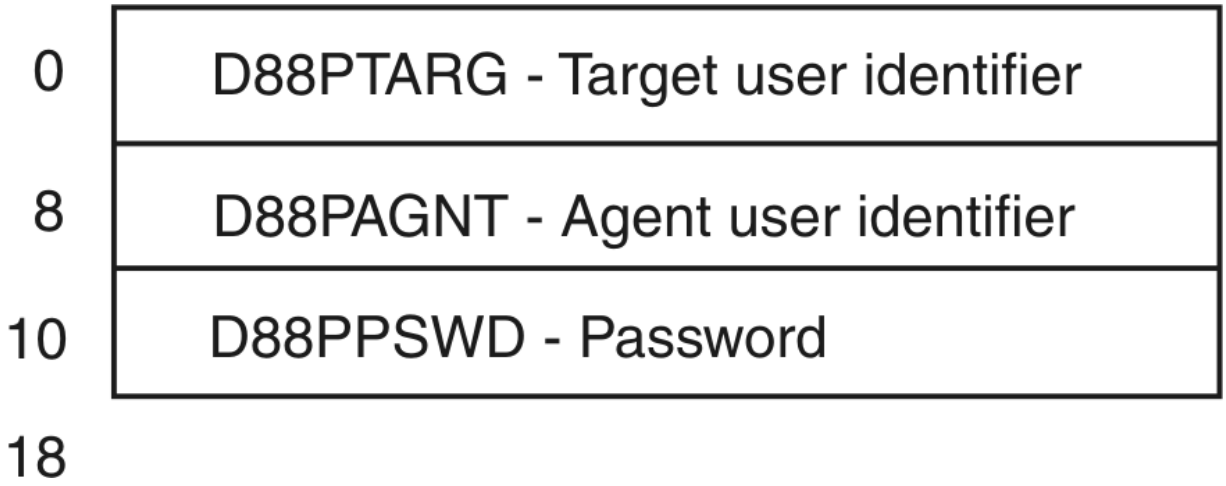
Entry Values for Subcode X'00':

Rx
X'00000000'

Ry
Pointer to the parameter list, D88PARM0. D88PARM0 COPY is provided in the HCPGPI macro library.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

D88PARM0 DSECT
The parameter list is in this format:



D88PTARG
is the user ID of the target virtual machine whose resources are to be accessed.

D88PAGNT
is the user ID of the end user requesting access to the target user's resources.

D88PPSWD
is the password of the end user requesting access to the target user's resources.

Subcode -1 – Verify Authorization to Use DIAGNOSE Code X'88'

Subcode -1 provides an application with the ability to invoke DIAGNOSE code X'88', without any additional parameters, to ensure the virtual machine has all needed authorizations to use DIAGNOSE code X'88', including those of the ESM. To use this functionality, invoke the CMS callable service DMSPASS, specifying the domain parameter as -1.

Entry Values for Subcode -1:

Rx
X'FFFFFFFF '

Ry
Is not used.

Ay
Is not used.

Subcode X'00' – Validate User Authorization

Subcode X'00' verifies that the supplied password is valid (for LOGON) for the designated agent. If the first character of the password is a blank, the verification is bypassed. If the target and agent user identifiers are different, this function ensures that the agent is authorized for LOGONBY to the target. Subcode X'00' is provided for compatibility with previous releases of z/VM. Applications should use subcode X'08' instead.

Subcode X'04' – Link Minidisk

Subcode X'04' links to a minidisk owned by the target virtual machine.

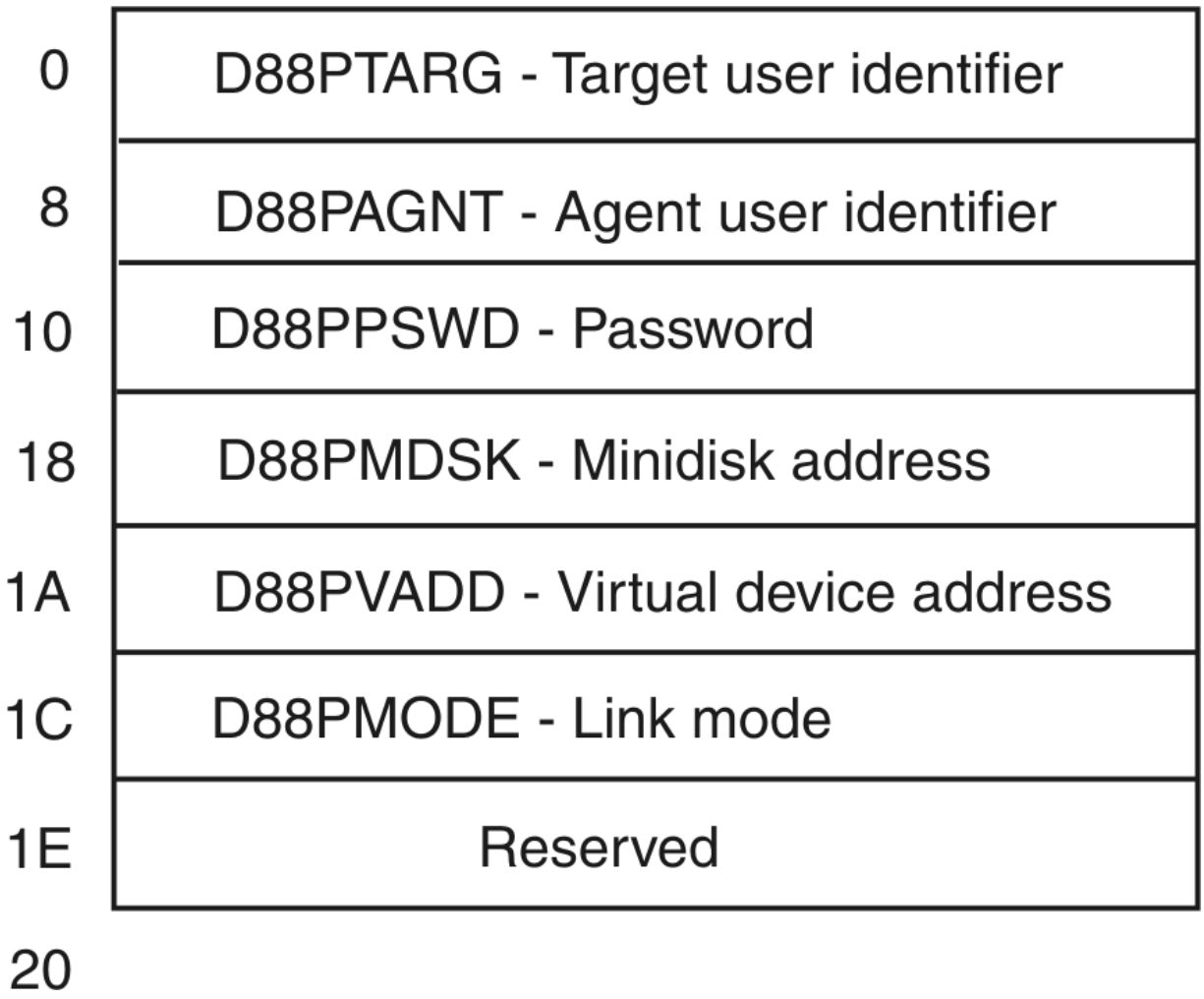
Entry Values for Subcode X'04':

Rx
X'00000004'

Ry
Pointer to the parameter list, D88PARM0.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

D88PARM0 DSECT
The parameter list is in this format:



D88PTARG
is the user ID of the target virtual machine whose minidisk is to be linked.

D88PAGNT
is the user ID of the end user requesting the access to the target user's minidisk.

D88PPSWD
is the minidisk password.

D88PMDSK
is the address of the target virtual machine minidisk to be linked.

D88PVADD
is the virtual device address in the requesting virtual machine's configuration at which the link is to be established.

D88PMODE
is the mode in which the link is to be established. This may be any of the valid LINK modes (R, RR, W, WR, M, MR, or MW). In addition, the first character may be specified as X, in which case the link is established as follows:

- If a password is supplied, the link is established in the highest mode associated with that password.
- If public access is permitted (ALL is the password defined in the User Directory), the link is established in the highest mode with such access.
- For the minidisk owner, the link is established in the mode associated with the minidisk definition in the User Directory.
- Otherwise, the link is established as read only (RR).

Exit Values: Return codes are set for DIAGNOSE code X'88'.

Subcode X'08' – Validate User Authorization

Subcode X'08' verifies the supplied password is valid (for LOGON) for the designated agent. If the target and agent user identifiers are different, this function ensures that the agent is authorized for LOGONBY to the target.

Subcode X'08' is similar to subcode X'00' with the exception that subcode X'08' handles password phrases and all necessary calls to an ESM. The caller does not have to worry about case, valid characters, or lengths.

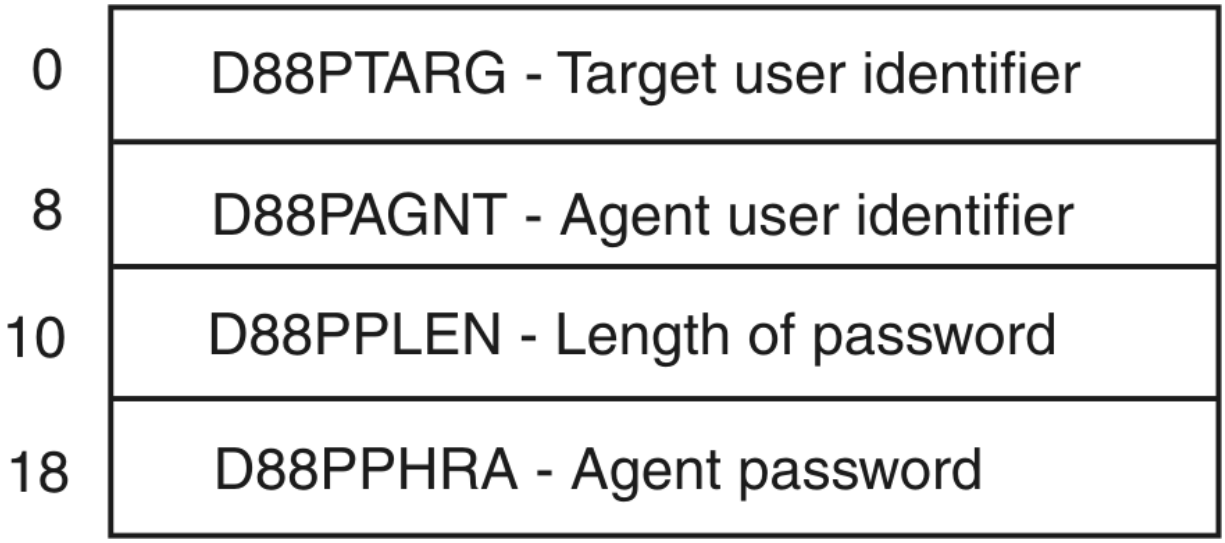
Entry Values for Subcode X'08':

Rx
X'00000008'

Ry
Pointer to the parameter list, D88PARM0.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Ry designates general register 0, Ay is not examined. The ALET is assumed to be 0, which indicates the host-primary address space.

D88PARM0 DSECT
The parameter list is in this format:



D88PTARG
is the user ID of the target virtual machine whose identity the agent would like to assume after authentication. If this field is not null or blank and is not the same as D88PAGNT, the agent's authorization to use LOGONBY to access the target user ID is verified. If D88PTARG is null or blank, it is assumed to be the same as D88PAGNT.

D88PAGNT
is the user ID of the agent whose password is to be verified.

D88PPLEN
is the length, in bytes, of the password; the maximum value is 200. Applications should be coded to allow up to 200 characters (including blanks), but should also be aware that an ESM may impose further restrictions on the password length. For example, RACF® supports passwords up to 100 characters in length.

D88PPHRA
is the password of the agent user ID. The password field is ignored if the agent user ID is defined in the CP directory as NOPASS.

Exit Values: Return codes are set for DIAGNOSE code X'88'.

Responses

Return Codes:

Return codes for subcode -1 are as follows:

Return Code in Rx	Meaning
0 (X'00')	The issuer is authorized to use this diagnose.
8 (X'08')	The issuer is not authorized to use this diagnose.
40 (X'28')	ESM required, but is not available.

Return codes for subcode X'00' are as follows:

Return Code in Rx	Meaning
0 (X'00')	Access granted.
8 (X'08')	Access denied.
24 (X'18')	Paging I/O error.

Return codes for subcode X'04' are as follows:

Return Code in Rx	Meaning
0 (X'00')	Minidisk linked read/write.
4 (X'04')	Minidisk linked read-only.
8 (X'08')	Access denied.
12 (X'0C')	Minidisk password required.
16 (X'10')	Minidisk password incorrect.
20 (X'14')	Link failed. Ry = LINK error message number.
24 (X'18')	Paging I/O error.

Return codes for subcode X'08' are as follows:

Return Code in Rx	Meaning
0 (X'00')	Agent is authenticated and is authorized to act as target.
8 (X'08')	Agent password not valid. This return code is also used when the agent user ID is defined in the CP directory with a password of NOLOG or AUTOONLY.
24 (X'18')	Paging I/O error.
28 (X'1C')	Agent authenticated but is not authorized to act as target.
32 (X'20')	Agent password has expired.
36 (X'24')	ESM active, but does not support password phrases.
40 (X'28')	ESM required, but is not available.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'88' is given incorrect data:

Problem Encountered	Cause
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state. • The issuer does not have class G privileges. • The issuer does not have the DIAG88 User Directory option. • The ESM is required, but is not available (subcodes X'00' and X'04' only).
Specification exception	One of the following: • Rx and Ry are the same register • The Rx register does not contain a valid subcode. • The address of the parameter list specified in Rx is not on a doubleword boundary. • A password length is zero or greater than 200. • The agent user ID is blank.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the parameter list.

DIAGNOSE Code X'8C' – Access 3270 Display Device Information

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'8C' to allow a virtual machine to obtain certain 3270 display device information instead of issuing a Write Structured Field Query command. If the characteristics of the display are altered dynamically, the data returned by DIAGNOSE code X'8C' does not reflect the changes. (This is because the information is obtained only at power-on time, or when the device becomes enabled for CP's use.)

Entry Values:

Rx
Is the guest real address of a user-provided data buffer; the buffer must start on a doubleword boundary. Rx cannot be register 15.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the data buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the data buffer is in the host-primary address space.

Ry
Is the length of the user-provided data buffer. This value must be greater than zero.

Rx+1
Must contain one of the following:

- The virtual device number of the 3270 display device for which information is requested
- The value negative 1 (-1). Specify -1 when the device is a virtual console whose device address is unknown to your virtual machine.

Exit Values:

Rx+1
Contains a return code–see the Responses section for a description.

Ry
Contains the residual count if the length specified on entry is greater than the amount of data received.

The data returned in the user-specified data buffer by DIAGNOSE code X'8C' is in the following format:

Byte 0	Byte 1	Bytes 2 and 3	Bytes 4 and 5	Bytes 6 through nnn
Flags	Number of partitions	Screen width, in cells (hexadecimal)	Screen height, in cells (hexadecimal)	Write Structured Field Query reply data

The flags are defined as follows:

X'80'—Extended color present
X'40'—Extended highlighting present
X'20'—Programmable symbol sets (PSS) present
X'02'—3270 emulation
X'01'—14-bit addressing allowed.

Usage Notes

1. The data returned in bytes 6 through *nnn* is the data returned from the Write Structured Field (WSF) Query, issued by CP. This data is stripped of the AID value X'88'. (The AID value signals the beginning of the WSF read-partition query reply data. Refer to your device operations manual for a more detailed description).
2. If the write structured field is not supported by the display, only six bytes of information are available: the flags and number of partitions, which contain zeros, and screen width and height, which contain their correct values.
3. If a paging error occurs when the pageable buffer that contains the query reply data is being paged in, up to six bytes of data are returned in the buffer specified by Rx, and if more than six bytes were requested, the residual count is returned in Ry. If less than six bytes were requested, that number of bytes is returned.
4. The amount of data returned by DIAGNOSE code X'8C' can be computed as follows:
   - If Ry(after) equals Ry(before)
   - Then the data length equals Ry(after)
   - Else the data length equals Ry(before) - Ry(after).

Responses

Return Codes: Upon completion of DIAGNOSE code X'8C', one of the following return codes is placed in Rx+1:

Return Code in Rx +1	Meaning
0 (X'00')	Successful completion
2 (X'02')	The specified device does not exist. It is not a display-type device, or the user is running in disconnected mode.
4 (X'04')	An I/O error has occurred (CP was unable to page in the guest's buffer).

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'8C' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • The length specified in Ry is negative or zero. • The virtual device number specified does not exist. • The buffer address is not on a doubleword boundary. • The Rx specified is register 15.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store into the data buffer.

DIAGNOSE Code X'90' – Read Symbol Table

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'90' to find the address of a specific symbol contained in the symbol table for diagnosis, monitoring, or tuning activities only.

The loader creates a symbol table at load time that contains all the external symbols in the system. This symbol table does not have a maximum size.

Entry Values:

Rx
Contains the length of the symbol. Trailing blanks are not counted. The value contained in Rx must be greater than or equal to 6, and less than or equal to 8; otherwise, you will get a specification error.

Ry
Contains the host logical storage address of the symbol.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the symbol. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the symbol is in the host-primary address space.

Usage Note

If the CSECT at the symbol is greater than 4 KB in length, the frames backing the host logical storage pages are not necessarily contiguous in host real storage.

Responses

Condition Codes: After executing DIAGNOSE code X'90', you receive a condition code indicating the success or failure of the instruction as follows:

Condition Code	Status	Rx Contains	Ry Contains
0	Resident symbol found	The length of the symbol if it is a CSECT (zero if the symbol is an entry point only).	The host logical storage address of the symbol in the system execution space.

Condition Code	Status	Rx Contains	Ry Contains
2	Symbol not found, or paging error	Unchanged	Unchanged

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'90' is given incorrect input data:

Problem Encountered	Cause
Specification exception	The length of the symbol (specified in Rx) is less than 6 or greater than 8.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the symbol.

DIAGNOSE Code X'94' – VMDUMP and Symptom Record Service

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'94' to:

- Request a dump of guest real storage to a system data file, in VMDUMP format, that can be processed by the DUMPLOAD utility for non-CP dumps, and viewed by the VM Dump Tool VMDUMPTL command for CP dumps.
- Request that CP process a symptom record. A copy of the completed symptom record is placed in the dump file (if a dump was requested), and copies of the completed symptom record are available to authorized symptom record recording virtual machines. Symptom records should be used to record suspected errors in IBM software or in IBM hardware.

Entry Values:

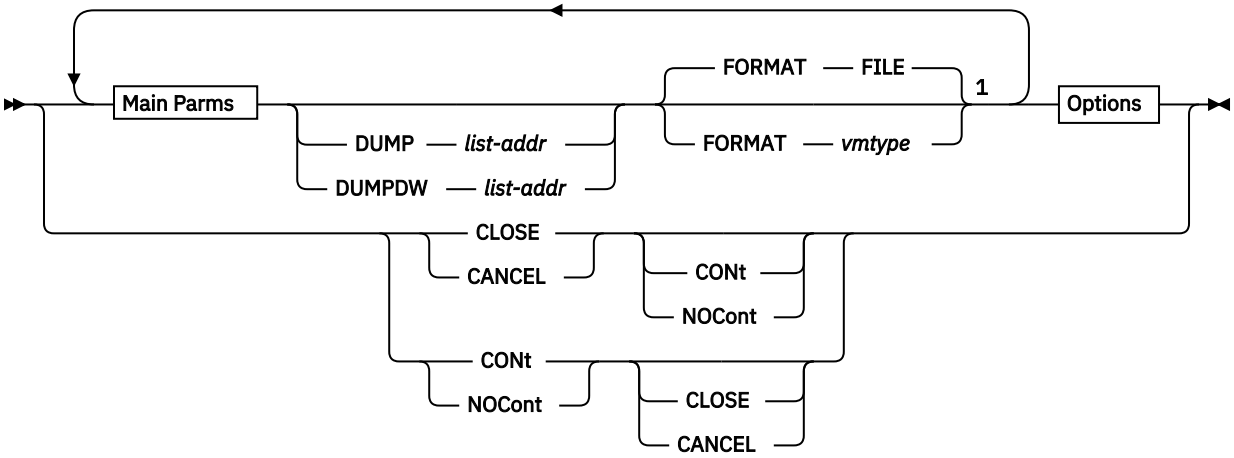
Rx
Is the address of the parameter string in guest real storage.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter string. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter string is in the host-primary address space.

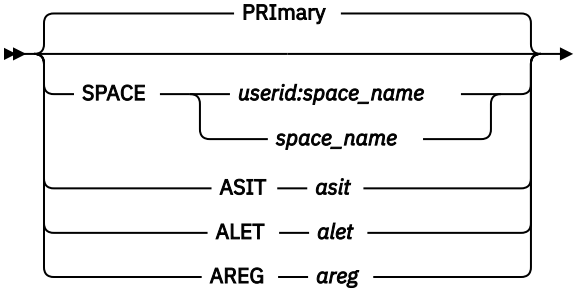
Ry
Is the length of the parameter string. When a request to process a symptom record accompanies a dump request, Ry should not be register 15.

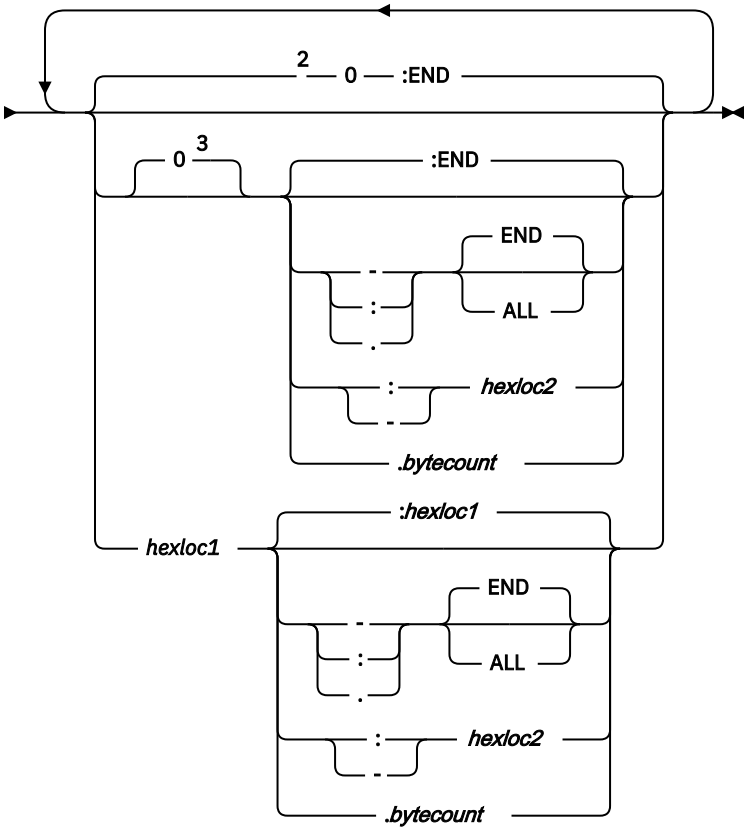
Supported Parameters

The parameters supported by DIAGNOSE code X'94' are in the following syntax diagram. If you are unfamiliar with reading syntax diagrams, see “Syntax, Message, and Response Conventions” on page xxxv. The parameter string must be no longer than 240 bytes, and must not be fetch-protected from the invoking program. It is a character string with the same format as the operands of the VMDUMP command, with the exception of the parameters described, which are unique to DIAGNOSE code X'94'.



Main Parms

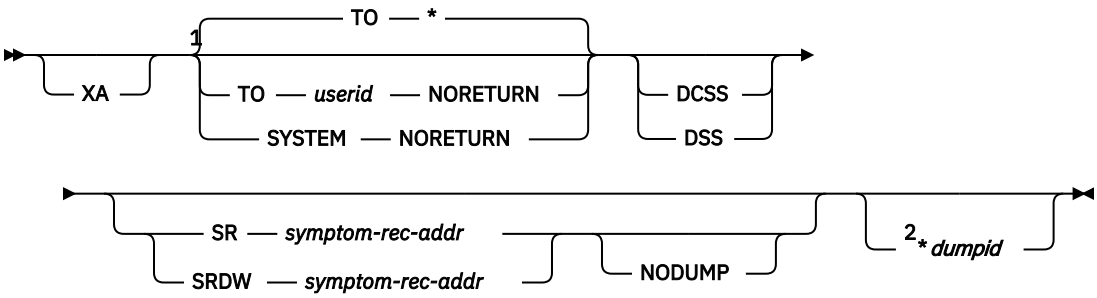




Notes:

[1] On an individual request, the address space qualifiers, ranges, dump address list, format, and optional parameters can be specified in any order. Address ranges and the format apply to the address space last specified on the request.
[2] When DCSS or DSS is specified and no address range operand is specified, the default range is ignored and no part of the address space is dumped. Also, if the address space qualifier specified refers to an address space other than the primary space, the default range is 0-ALL.
[3] This path is followed only when the starting address is zero *by default*. If zero is specified explicitly, the path starting with *hexloc1* is followed.

Options



Notes:

[1] The user who is to receive the dump files can be specified, even by default, only once and only on the first request for the dump.
[2] If **dumpid* is specified, it *must* be last.

Parameters Unique to DIAGNOSE code X'94'

DUMP *list-addr*
indicates that the dump address ranges are in hexadecimal form within a separate dump address list. *List-addr* is the address of the dump address list in the same address space as the parameter list. It is expressed as a 4-byte hexadecimal address immediately following the DUMP keyword and a single blank. See "Dump Address List" on page 116 for a complete description of the dump address list.

DUMPDW *list-addr*
indicates that the dump address ranges are in hexadecimal form within a separate dump address list. *List-addr* is the address of the dump address list in the same address space as the parameter list. It is expressed as an 8-byte hexadecimal address immediately following the DUMPDW keyword and a single blank. See "Dump Address List" on page 116 for a complete description of the dump address list.

NORETURN
indicates that the invoking virtual machine may not transfer the spool file back to itself from the dump receiver, using the TRANSFER or CHANGE command. This parameter must be used only with the SYSTEM or TO *userid* (in other words, not your own user ID) parameter designating the authorized receiver of the dump file. This facility can be used to restrict access to dumps of shared virtual machine storage to certain authorized users.

SR *symptom-rec-addr*
indicates that a symptom record is to be processed by CP. *Symptom-rec-addr* is the address of the symptom record in the same address space as the parameter string. It is expressed as a 4-byte hexadecimal address immediately following the SR keyword and a single blank. This parameter may be used to record suspected software errors in IBM products. Symptom records should not be used to record end-user errors. Errors that are presented through DIAGNOSE code X'94' using the SR keyword are those that need the attention of the personnel who are responsible for first-level problem determination at an installation.

SRDW *symptom-rec-addr*
indicates that a symptom record is to be processed by CP. *Symptom-rec-addr* is the address of the symptom record in the same address space as the parameter string. It is expressed as an 8-byte hexadecimal address immediately following the SRDW keyword and a single blank. This parameter may be used to record suspected software errors in IBM products. Symptom records should not be used to record end-user errors. Errors that are presented through DIAGNOSE code X'94' using the SRDW keyword are those that need the attention of the personnel who are responsible for first-level problem determination at an installation.

NODUMP
indicates that the symptom record is to be made available to the authorized symptom record recording virtual machine, but no virtual machine dump is to be taken. This parameter may only be used with the SR parameter indicating the address of a symptom record to be processed.

The remaining parameters are exactly as specified for the VMDUMP command in *z/VM: CP Commands and Utilities Reference*.

Dump Address List

A virtual machine program that issues DIAGNOSE code X'94' can provide dump address lists containing address ranges to be dumped. These lists are in hexadecimal format and the maximum number of dump address lists is 2,049. When the DUMP or DUMPDW keyword is used, you must store the 4 or 8 byte address of the first list in the DIAGNOSE code X'94' parameter list described above, following the DUMP keyword and separated from it by a blank. When the DUMPDW keyword is used, the address of the first list is an 8-byte address. The first word of each list contains a pointer to the next list.

The dump address list can be in one of four formats based on whether an address space qualifier is specified. Figure 10 on page 116 shows the 31-bit base format (without an address space qualifier), and Figure 11 on page 116 shows the 31-bit extended format (with an address space qualifier). Figure 12 on page 116 shows the 64-bit base format, and Figure 13 on page 117 shows the 64-bit extended format.

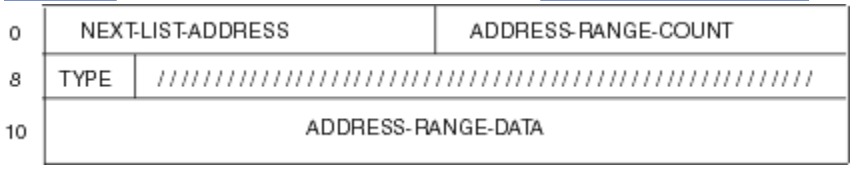


Figure 10. 31-bit Base format dump address list (without an address space qualifier)

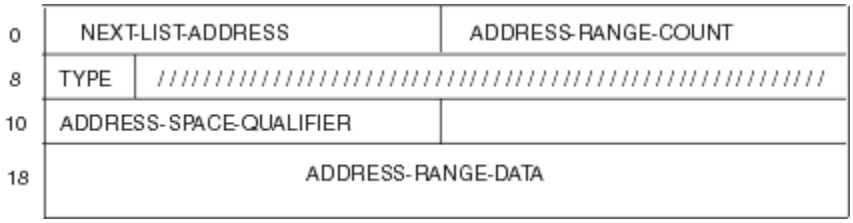


Figure 11. 31-bit Extended format dump address list (with an address space qualifier)

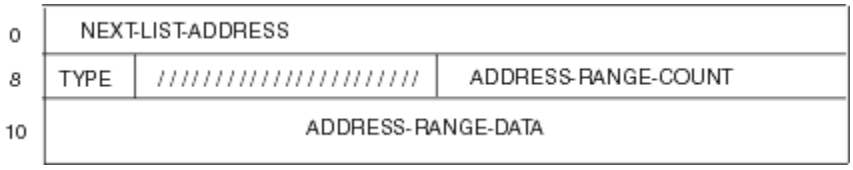


Figure 12. 64-bit Base format dump address list (without an address space qualifier)

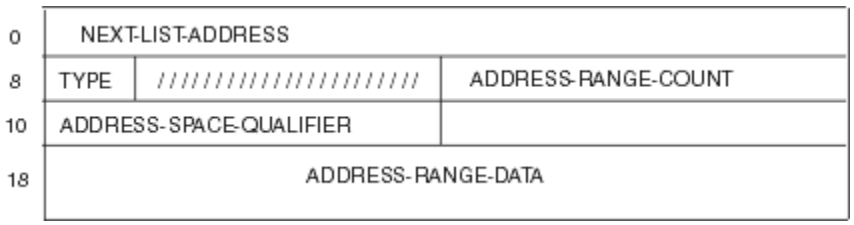


Figure 13. 64-bit Extended format dump address list (with an address space qualifier)

NEXT-LIST-ADDRESS
is the fullword or doubleword address of the next dump address list or a fullword containing binary zeros if no further list exists.

ADDRESS-RANGE-COUNT
is the fullword count of the number of address ranges included in this dump address list. It must be greater than zero.

TYPE
is a one-byte flag indicating:

- whether the address ranges in the list are specified in address/length or address/address form (X'80')
- whether the address space qualifier is specified (X'40')
- whether the addresses are 31-bit or 64-bit (X'20').

Bits 3-7 are reserved and must contain binary zeros.

```
1... .... (X'80')  0 = address/length
                    1 = address/address
.1.. .... (X'40')  0 = no address space qualifier specified
                    1 = address space qualifier specified
..1. .... (X'20')  0 = 31-bit addresses
                    1 = 64-bit addresses
```

address/length
Ranges are specified as the starting address and length.

For 31-bit addresses, the length is specified as a 4 byte value. For 64-bit addresses, the length is specified as an 8-byte value.

address/address
Ranges are specified as the starting address and ending address.

ADDRESS-SPACE-QUALIFIER
is the 4-byte ALET specifying the address space with which each address range in the dump address list is associated.

Notes:

1. The values represented by this format of an ALET and the format specified on the DIAGNOSE invocation (for example, for DCSS) can be the same. However, the format is different - in the dump address list, the ALET is a 4-byte value; on the invocation, the ALET is the 8-digit hexadecimal representation of the 4-byte value.
2. Only the primary address space can be specified when in z/Architecture mode.

ADDRESS-RANGE-DATA
consists of pairs of fullwords or doublewords containing the starting address of each range followed by either the length or the ending address, depending on the setting of the TYPE flag. The number of range entries is contained in the ADDRESS-RANGE-COUNT field.

These ranges point to data in the address space identified by the address space qualifier. If there is no address space qualifier, the ranges point to data in the primary address space. The ranges are not affected by any address space qualifier specified in the parameter string.

Usage Notes Regarding Dumping a Virtual Machine

1. The requested dump file contains only storage that the user is authorized by storage key to access. Any block of storage that is fetch protected with a key other than that of the invoking program is not dumped. A virtual machine program that executes in key zero can dump any part of virtual storage. Shared segments which comprise named saved systems and discontiguous saved segments are also dumped, subject to the same protection rule. Only second-level storage (that is, guest real storage) is dumped. Operating systems that execute in translate mode create third-level (guest virtual) storage.

   Note: In addition to storage key authorization, the virtual machine must be permitted to access the requested address space.
2. You cannot specify inline address ranges and DUMP or DUMPDW address list ranges that refer to the same space on the same invocation of DIAGNOSE code X'94'.
3. The maximum address that VMDUMP can dump is X'7FFFEFFF' for ESA/390 machines, and X'7FFFFFFFFF' (512 Gigabytes - 1) for z/Architecture machines.
4. VMDUMP dumps of z/Architecture guests on z/VM Version 5.2 are not compatible with VMDUMP dumps of z/Architecture guests of previous releases.
5. The absence of discontiguous saved segments when DCSS is specified results in an error only if no other area of storage was requested.
6. The dump address list must contain at least one address range.
7. The DUMP, DUMPDW and NORETURN parameters may not be specified on the same invocation of DIAGNOSE code X'94' with CLOSE, CANCEL, CONT, or NOCONT. See the VMDUMP command documentation for a list of other parameters with the same restriction.
8. When multiple DIAGNOSE code X'94' invocations are used in continuous output mode, the NORETURN parameter is only accepted on the first request which opens the file. Once successfully specified, it may be repeated on later requests. However, if it is not specified on the first request, subsequent requests (affecting the same VMDUMP file) that specify NORETURN cause an error.
9. When multiple DIAGNOSE code X'94' invocations are used in continuous output mode, the DUMP parameter may be specified on any request to dump storage.
10. You may not be authorized to issue this DIAGNOSE code if an external security manager is installed on your system. This restriction applies only to the VMDUMP TO command. For additional information, contact your security administrator.
11. Long-running DIAGNOSE code X'94' processing can be halted by the user by entering #CP CPHX. Another user can halt a user during DIAGNOSE code X'94' processing by using the CPHX or FORCE command. A TERM BRKKEY (normally set to PA1 key) will not halt DIAGNOSE code X'94' processing.
12. For more information on the VMDUMP command, see the Usage Notes in *z/VM: CP Commands and Utilities Reference*.

Usage Notes Regarding Dump Address Lists

1. The dump address list must contain at least one address range.
2. If the dump address list does not contain address space qualifiers then DIAGNOSE code X'94' processing will consider the address ranges in the dump address list to apply to the primary address space.
3. Because address space qualifiers are within the dump address list any address space qualifiers specified on the same DIAGNOSE code X'94' invocation with a DUMP *list-addr* option will not affect the addresses in the dump address list. For example, if the DIAGNOSE code X'94' parameter string contained

   ```
   'ASIT 12A456B81B345F78 FORMAT CMS DUMP list-addr'
   ```

   the address space qualifier ASIT 12A456B81B345F78 would apply only to the FORMAT CMS option. The addresses in the dump address list would either default to the primary address space or be qualified by the address-space-qualifier(s) within the dump address list.

You cannot specify dump ranges for the same address space in both the parameter list and the dump address list.

Usage Notes Regarding Symptom Records

1. DIAGNOSE code X'94' processing always creates and records a symptom record when taking a dump. If the guest uses the SR or SRDW keyword, CP updates that symptom record and records if for the guest.
2. A symptom record is processed only if it resides within storage that the user is authorized by storage key to access. If the storage within which the symptom record resides is fetch-protected with a key other than that of the invoking program, the symptom record is not processed; if a dump was requested along with symptom record processing, CP generates a symptom record to be placed in the virtual machine dump. This record contains only CP environment information. A symptom record supplied by a virtual machine that executes in key zero is always processed, regardless of where in guest real storage the symptom record resides.
3. The program that issues the DIAGNOSE code X'94' must initialize the required fields within the symptom record with information describing a problem before it issues DIAGNOSE code X'94'.
4. CP updates the symptom record to contain the following information:
   - CP environment data in section 1
   - The architectural level of the CP symptom record processor (C'10') in section 2
   - Return and reason codes in section 2.1.

   If the invoking program is running with a PSW key that is neither the storage key of the symptom record nor zero, the system records the symptom record but does not update the invoker's copy in virtual machine storage with CP environment data.
5. Symptom records which are over 3500 decimal bytes in length are truncated before being placed in a dump or made available to symptom record recording virtual machines.
6. All fields designated by an RS are updated within the caller's symptom record by CP. The conventions for initializing the contents of the RS (required from system service) fields are:
   a. The customer-assigned system node name (ADSRID) is the value specified on the SYSTEM_IDENTIFIER statement in the system configuration file for the CPU on which CP is running while processing the DIAGNOSE code X'94' request.
   b. The ADSRTOD field is in Coordinated Universal Time (UTC). The ADSRGMT field contains the clock zone differential that may be of value to users who want to convert to local. If the STCK instruction completes with a nonzero condition code, the clock value (ADSRTIME) is hex zeros, and the printable time/date fields are blanks.
7. When the SR or SRDW parameter is coded, registers Rx+1 and Ry+1 reflect the results of symptom record processing in the form of hexadecimal return codes and reason codes. If the symptom record storage within the virtual machine can be written to, the ADSRRET and ADSRREA fields in section 2.1 contain these same results. The return code and reason code value reflect the most severe level of error that occurred. The reason code value contains the value for the first error found at the severity level indicated by the return code.

   If the symptom strings in sections 3 and 4 contain other than alphanumeric or national characters, pound signs, slashes, or blanks, a return code of 4 and a reason code of X'0E08' are generated.
8. The SDB syntax of the symptoms in sections 3 and 4 is not validated.
9. If a DIAGNOSE code X'94' is issued with the SR or SRDW keyword in the parameter list, and section 1 of the symptom record is already initialized, the CP environment sections (RS fields) are not updated within the symptom record. An unmodified copy of the symptom record is sent to each virtual machine connected to the CP Symptom Recording system service (*SYMPTOM). *SYMPTOM is documented in Chapter 22, “Symptom System Service (*SYMPTOM),” on page 773.
10. The SR, SRDW and NODUMP parameters may not be specified on the same invocation of DIAGNOSE code X'94' with CLOSE, CANCEL, CONT, or NOCONT.

11. When multiple DIAGNOSE code X'94' invocations are used in continuous output mode, the SR or SRDW parameter can be specified on any one invocation to supply the symptom record in the dump. When a program issues a DIAGNOSE code X'94' request only to add a symptom record and does not want to dump any areas of guest storage, it should include a dummy address range of zero. This is required because, if no address range or DUMP/DUMPDW address list is specified, all of guest storage is dumped. Because page zero is always included in VMDUMP files, a request such as *0 SR addr* does not result in any added output.
12. Input symptom record storage

    Unused space is not compressed from the symptom record. The symptom record sections are written in the sequence and at the displacements in which they are presented. No unused symptom record storage should be passed by means of the DIAGNOSE code X'94' instruction. For example, there should not be preassigned, unused space at the end of symptom record sections 3, 4, or 5. The length for that section (contained in section 2) should only be the length of the *used* portion of the section when the DIAGNOSE code X'94' is issued; otherwise, the common storage that is allocated to the service routine and the CMS symptom record file contain wasted space.
13. Symptom record recording considerations

    When a DIAGNOSE code X'94' is issued with a parameter list containing an SR or SRDW keyword, CP sends copies of the symptom record asynchronously to authorized virtual machines that are connected to the CP Symptom Recording system service (*SYMPTOM). Control is returned to the virtual machine that issued the DIAGNOSE code X'94' instruction after a copy of the symptom record is placed on the recording queue. For more information on *SYMPTOM, see Chapter 22, “Symptom System Service (*SYMPTOM),” on page 773.

Responses

Condition Codes and Return Codes: Upon completion of DIAGNOSE code X'94', control is returned to the invoker with a condition code set to indicate the status of both input parameter list processing and dump processing. A return code in Ry indicates the results of processing the input parameter string and the dump request (if present).

If symptom record processing was requested, return and reason codes in registers Ry+1 and Rx+1, respectively, indicate the result of symptom record processing (condition codes are not used to reflect the status of symptom record processing).

The condition codes are listed in Table 7 on page 120.

Table 7. DIAGNOSE Code X'94' Condition Codes

Condition Code	Meaning
0	The function has completed successfully. All requested ranges have been dumped.
1	The function completed with error(s). Portions of the requested address ranges have been dumped. Ry contains a return code which indicates the reason for the error.
2	The function failed. No dump has been created. Ry contains a return code which indicates the reason for the failure.

The return codes in Ry are listed in Table 8 on page 121.

Table 8. DIAGNOSE Code X'94' Return Codes

Condition Code	Return Code	Meaning
0,1	0 (X'00')	Successful completion. • CC = 0 indicates that all requested areas were dumped. • CC = 1 indicates that the DCSS operand was specified without any ranges or dump address list, and no DCSSs are loaded.
2	4 (X'04')	Parameter list exceeds 240 bytes.
1,2	8 (X'08')	System I/O error reading a guest page. • CC = 1 indicates that at least one requested page could not be dumped. • CC = 2 indicates that the parameter list or dump address list could not be accessed.
1,2	12 (X'0C')	Storage access violates guest fetch protection. • CC = 1 indicate that at least one requested page could not be dumped. • CC = 2 indicates that the parameter list or dump address list could not be accessed.
2	16 (X'10')	Invalid dump address range. More than 2,049 dump address lists, ending address less than start, or range length of 0 given, or too many ranges in the address list. (A single list must fit within a page of storage.)
2	20 (X'14')	Conflicting options.
2	24 (X'18')	Userid missing or longer than eight characters.
2	28 (X'1C')	hexloc missing or invalid hexadecimal value. If dump address list is used, number of ranges specified was zero.
2	32 (X'20')	Required parameter missing or invalid.
2	36 (X'24')	User ID not in directory.
2	40 (X'28')	Spooling error.
2	44 (X'2C')	hexloc exceeds storage size.
2	48 (X'30')	Return code unused but reserved for compatibility with VM/SP, VM/SP HPO, and VM/ESA (370 Feature)
2	52 (X'34')	Invalid address points to storage outside of user area.
2	56 (X'38')	Soft abend occurred during processing.
1, 2	60 (X'3C')	Unaddressable range given. At least one range given was above guest storage and did not fall within a discontiguous saved segment. The unaddressable ranges are not dumped. • CC=1 indicates the valid addresses given are dumped. • CC=2 indicates that no valid ranges were given.
2	64 (X'40')	For an XC virtual machine in access-register mode: ALET in the parameter list is missing or invalid

Table 8. DIAGNOSE Code X'94' Return Codes (continued)

Condition Code	Return Code	Meaning
2	68 (X'44')	For an XC virtual machine in access-register mode: Access register is missing or invalid.
2	72 (X'48')	For an XC virtual machine in access-register mode: ASIT in the parameter list is missing or invalid.
2	76 (X'4C')	For an XC virtual machine in access-register mode: Address-space identifier in the parameter list is missing or invalid.
1	80 (X'50')	For an XC virtual machine in access-register mode: An address space has become inaccessible to your virtual machine. One or more pages associated with that address space were not dumped.
2	84 (X'54')	You have used an operand that is not valid for your virtual machine mode or architecture mode.
1	92 (X'5C')	CP cannot add the requested information to the current virtual machine dump. Either more than 200 address spaces have been requested, or there have been more DIAGNOSE X'94' invocations requesting data from a single address space than allowed (900 invocations are allowed for ESA/390, ESA/XA and ESA/XC. 500 invocations are allowed for z/Architecture.
2	96 (X'60')	Unable to access the address space due to an ALEN-translation exception condition.
2	100 (X'64')	Unable to access the address space due to an ALET-specification exception condition.
2	104 (X'68')	Unable to access the address space with the specified address space identifier (addressing-capability exception condition).
2	108 (X'6C')	Authorization request failed.
2	112 (X'70')	Unable to access the address space with the specified ASIT (addressing-capability exception condition).
2	116 (X'74')	The parameter list or dump address list is in a space that does not exist or that you are not authorized to access.
2	120 (X'78')	The space designated by the ALET parameter or the address space qualifier in the dump address list cannot be accessed due to an addressing-capability exception condition.
2	124 (X'7C')	The space designated by the AREG parameter cannot be accessed due to an addressing-capability exception condition.
1	132(X'84')	Processing halted by CPHX or FORCE command.

Upon completion of DIAGNOSE code X'94', if the SR or SRDW parameter was specified, the return codes shown in Table 9 on page 123 are placed in Ry+1:

Table 9. DIAGNOSE X'94' Symptom Record Processing Return Codes

Return Code	Meaning
0 (X'00')	Successful completion. The symptom record was recorded and updated (if necessary) in virtual machine storage.
4 (X'04')	Error(s) detected within the DIAGNOSE code X'94' request. The entire input symptom record was recorded.
8 (X'08')	Error(s) detected within the DIAGNOSE code X'94' request. A partial symptom record was recorded.
18 (X'12')	An error was detected within the DIAGNOSE code X'94' request. None of the symptom record was recorded.
22 (X'16')	An error was detected within the DIAGNOSE code X'94' service. None of the symptom record was recorded.

The reason code shown in Table 10 on page 123 may be returned in Rx+1 when Ry+1 contains a return code value of X'00':

Table 10. DIAGNOSE X'94' Reason Codes for Return Code X'00'

Reason Code	Meaning
X'0000'	Successful completion

The reason codes shown in Table 11 on page 123 may be returned in Rx+1 when Ry+1 contains a return code value of X'04':

Table 11. DIAGNOSE X'94' Reason Codes for Return Code X'04'

Reason Code	Meaning
X'0164'	The input symptom record was copied, but the key of the storage containing the symptom record did not allow the symptom record to be updated.
X'0E08'	The input symptom record was processed, but invalid characters were found in section 3 or 4.

The reason codes shown in Table 12 on page 123 may be returned in Rx+1 when Ry+1 contains a return code value of X'08':

Table 12. DIAGNOSE X'94' Reason Codes for Return Code X'08'

Reason Code	Meaning
X'0158'	The total length of the input symptom record exceeds 3500 decimal bytes.
X'015C'	Optional portions of the input symptom record were not in accessible storage. The resulting record includes all accessible entries of the input symptom record.

The reason codes shown in Table 13 on page 123 may be returned in Rx+1 when Ry+1 contains a return code value of X'12':

Table 13. DIAGNOSE X'94' Reason Codes for Return Code X'12'

Reason Code	Meaning
X'0104'	The first two bytes of the symptom record do not contain the characters *SR*.
X'0108'	The input symptom record does not contain the required entries of section 2.
X'010C'	The input symptom record does not contain the required entries of section 2.1.

Table 13. DIAGNOSE X'94' Reason Codes for Return Code X'12' (continued)

Reason Code	Meaning
X'0114'	The input symptom record does not contain the required entries of section 3.
X'012C'	Required portions of the input symptom record are not in accessible storage.
X'0E04'	The input symptom record is a duplicate; a continuous mode dump is being processed, and a symptom record has already been placed in the dump file.

DIAGNOSE Code X'98' – Real I/O

Privilege Class: Any

Addressing Mode: 31-bit

Using DIAGNOSE code X'98', a virtual machine can lock and unlock virtual pages and can execute its own real channel programs.

The subfunctions of DIAGNOSE code X'98' are LOCK, UNLOCK, and SSCH-Real. The requested subfunction is identified by a code in the Rx register. The LOCK subfunction locks a single 4 KB page of virtual machine storage in host real storage. In addition, it returns to the virtual machine the absolute storage address of the frame used to lock the guest page. The UNLOCK subfunction unlocks a single 4 KB page of virtual machine storage that was previously locked by DIAGNOSE code X'98'. In addition, multiple 4 KB pages can be locked or unlocked by invoking the Block Diagnose X'98' Request. This allows multiple occurrences of a Diagnose X'98' subfunction to be executed with only a single invocation of the Diagnose instruction. The SSCH-Real subfunction initiates execution of a real channel program for XA, ESA, XC, and Z virtual machines.

DIAGNOSE code X'98' allows a virtual machine to bypass CCW translation and initiate execution of real channel programs. Execution of this DIAGNOSE bypasses most software and hardware protection mechanisms. For example, the virtual machine is able to read from, or write to, **any** frame in storage that does not have a storage key of 0. If the virtual machine is running a program with an error in it, system (that is, other users) security and integrity may be compromised. The ability to execute DIAGNOSE code X'98' is controlled by the directory OPTION statement, and should be granted only to the most trusted programs after careful consideration of the possible security and integrity exposures.

LOCK Subfunction

This subfunction locks a selected page of virtual machine absolute storage into host real storage below 2 GB, thus excluding the page from future paging activity. Locking pages can enhance the efficiency of a particular virtual machine by keeping frequently-used pages resident in real storage. When the LOCK subfunction is issued, it operates exactly like the LOCK command, obtaining a real storage page from the normal paging area.

Entry Values:

Rx
contains X'00000000' the subfunction code

Ry
contains the guest absolute address to be translated and locked. Ry cannot be register 15.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the page to be locked. This address space must be owned by the virtual machine issuing DIAGNOSE code X'98'. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the page to be locked is in the host-primary address space.

Exit Values:

Ry+1
contains one of the following:

- The host absolute address (set by CP with condition code 0)
- A return code (set by CP with condition code 3):

Return Code	Meaning
1 (X'01')	Reserved.
2 (X'02')	This is an invalid guest address.
3 (X'03')	Page cannot be locked.
5 (X'05')	ALET-specification exception condition for an XC virtual machine in access-register mode: Ay contains an ALET that cannot be translated.
6 (X'06')	ALEN-translation exception condition for an XC virtual machine in access-register mode: Ay contains an ALET that cannot be translated.
7 (X'07')	For an XC virtual machine in access-register mode, the issuer of this DIAGNOSE is not the owner of the address space or the issuer is the owner but the address space has been destroyed.

UNLOCK Subfunction

This subfunction unlocks a page of virtual machine storage that was previously locked by DIAGNOSE code X'98'. Real storage pages become available for normal paging operations.

Entry Values:

Rx
contains X'00000004'—the subfunction code

Ry
contains the guest absolute address of the page to be unlocked

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the page to be unlocked. This address space must be owned by the virtual machine issuing DIAGNOSE code X'98'. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the page to be unlocked is in the host-primary address space.

Exit Values: If UNLOCK was successful, the condition code is 0 and Ry+1 is unchanged.

If UNLOCK was not successful, the condition code is 3 and Ry+1 contains one of the following return codes:

Return Code	Meaning
1 (X'01')	Reserved.
2 (X'02')	This is an invalid guest address
3 (X'03')	Page is already unlocked
5 (X'05')	ALET-specification exception condition for an XC virtual machine in access-register mode: Ay contains an ALET that cannot be translated.
6 (X'06')	ALEN-translation exception condition for an XC virtual machine in access-register mode: Ay contains an ALET that cannot be translated.
7 (X'07')	For an XC virtual machine in access-register mode, the issuer of this DIAGNOSE is not the owner of the address space or the issuer is the owner but the address space has been destroyed.

SSCH-Real Subfunction

This subfunction initiates execution of a real channel program that the virtual machine has built in locked pages of storage. The address of the first CCW in the ORB is a host absolute address, as are the data addresses and transfer addresses (in TIC CCWs). For more information on ORB, see the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf), *z/VM: ESA/XC Principles of Operation*, or z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), based on the architecture mode of the virtual machine. This subfunction is valid for an XA, ESA, XC, or Z virtual machine, and it operates as a SSCH instruction. Condition codes 0, 1, and 2 are compatible with normal virtual I/O operation of that instruction. Condition code 3 is extended to provide a return code to indicate error conditions unique to the Real I/O interface. The device identified by the parameter register must be defined and dedicated or a full-pack minidisk. The device cannot be read/only. To protect host storage from overlay by guest Real I/O, the subchannel key in the ORB must be nonzero.

Entry Values:

R1
contains the subsystem identification (SID) for the virtual device

Rx
contains X'0000000C'–the subfunction code

Ry
contains the guest logical address of the ORB.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the ORB. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the ORB is in the host-primary address space.

Exit Values:

Ry+1
contains the return code (set by CP with condition code 3), as follows:

Return Code	Meaning
1 (X'01')	The device is not operational or is not attached.
2 (X'02')	The device is neither dedicated nor a full-pack minidisk, or is read-only, or the device is already active with a system function such as SPXTAPE.
3 (X'03')	The subchannel key in the ORB is zero.

Block Diagnose X'98' Request

Use the Block Diagnose X'98' subfunction to allow multiple occurrences of a Diagnose X'98' subfunction to be executed with only a single invocation of the Diagnose instruction. Both the LOCK and UNLOCK subfunctions are supported with the Block Diagnose X'98' instruction.

Entry Values:

Rx
contains X'00000010'— the subfunction code to specify a Block Diagnose X'98' request.

Ry
contains the real address of a Diagnose X'98' Multiple Request Block for the function to be performed. The address must be on a word boundary; otherwise a specification exception will occur.

Ry cannot be register 15.

Exit Values:

Ry+1

A global return code reflected by the Diagnose X'98' processor, which specifies the overall results of the Block Diagnose X'98' request.

The global return code may be checked by the program to quickly determine whether all specified Diagnose X'98' subfunctions within the array have completed successfully. If the global return code is zero, there is no need to examine each individual RC field value in order to determine successful completion.

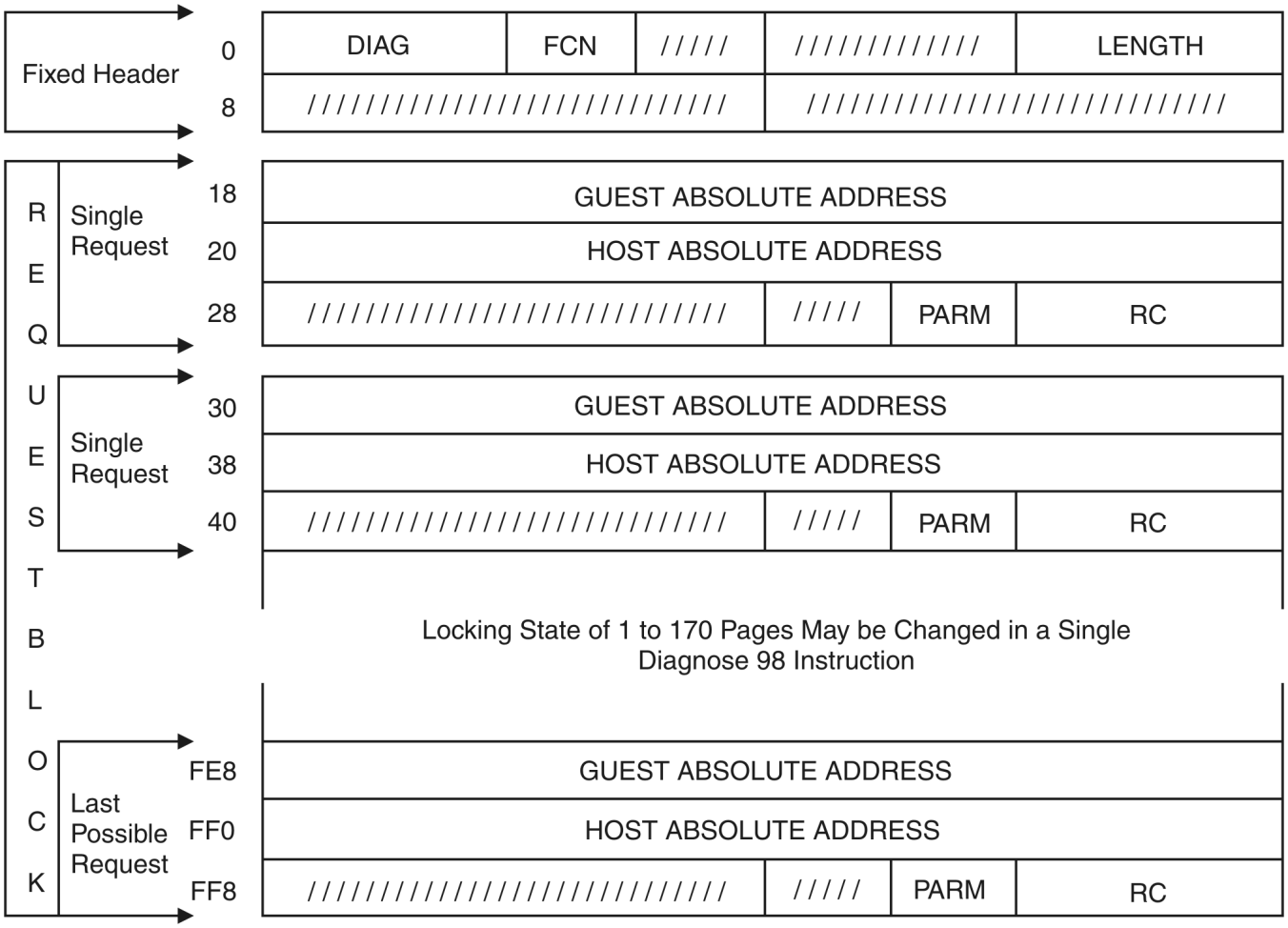
Return Code	Meaning
0 (X'00')	All requests completed successfully.
1 (X'01')	One or more requests specified in the Diagnose X'98' Multiple Request Block RC field must be checked to determine the failing requests.
2 (X'02')	Reserved.
3 (X'03')	Reserved.
4 (X'04')	Reserved. Not used by z/VM.
5 (X'05')	Invalid value specified in the Diagnose X'98' Multiple Request Block DIAG field.
6 (X'06')	Invalid value specified in the Diagnose X'98' Multiple Request Block FCN field.
7 (X'07')	Invalid value specified in the Diagnose X'98' Multiple Request Block LENGTH field.

Diagnose X'98' Multiple Request Block

The Diagnose X'98' Multiple Request Block is used by the program to specify multiple occurrences of a single Diagnose X'98' subfunction to CP for execution. The Diagnose X'98' Multiple Request Block is a self defining control block, which is divided into two sections. The first section is the Diagnose X'98' Multiple Request Block Fixed Header which is used to define the control block, provide parameters, a response area, and is global to the entire request. This fixed header section is common for all Block Diagnose X'98' requests.

The second section of this request block is variable in size, containing an array of Diagnose X'98' subfunction requests. The number of requests within the array is determined from the LENGTH and FCN fields specified in the fixed area of the request block. The data within an array entry is determined by the FCN code specified, which can be the LOCK subfunction (FCN code X'00'), the UNLOCK subfunction (FCN code X'04'), or the RELEASE AND LOCK subfunction (FCN code X'20').

The following is the layout for the entire Diagnose X'98' Multiple Request Block:



Diagnose X'98' Multiple Request Block Fixed Header: The fixed header section of the Diagnose X'98' Multiple Request Block has a length of 16 bytes for all requests. The following is a list of fields which make up the fixed header:

DIAG
Is the halfword DIAGNOSE code. It must contain the halfword X'0098'.

FCN
Specifies the Diagnose X'98' subfunction:

X'00'
LOCK

X'04'
UNLOCK

X'20'
RELEASE AND LOCK

LENGTH
Is the length in bytes of the entire Diagnose X'98' Multiple Request Block. The valid range for the length field is X'28' to X'1000'.

////////
Indicates fields reserved for future use.

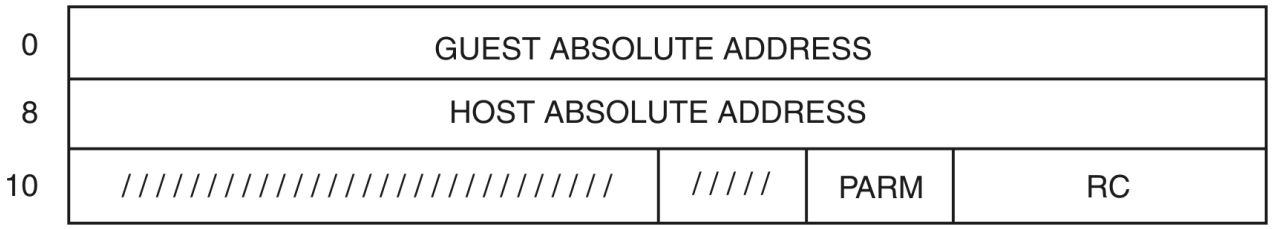
LOCK Subfunction FCN Code X'00'

The LOCK subfunction locks multiple pages of a virtual machine's absolute storage into host real storage below 2 GB, thus excluding the pages from future paging activities. When the LOCK function is issued, it is

identical to issuing multiple CP LOCK commands, obtaining real storage frames from the normal paging area.

An XC virtual machine can lock pages only within the host-primary address space. The Ay register is ignored.

The following structure is required for each individual LOCK request included in the array area of a FCN=X'00' Diagnose X'98' Multiple Request Block:



GUEST ABSOLUTE ADDRESS
Is the 8-byte guest absolute address that the program wants to lock in host storage. The 4 KB page which contains the specified address will be locked in host storage. The value specified does not need to be on a page boundary.

HOST ABSOLUTE ADDRESS
Is the 8-byte host absolute address for the program-specified GUEST ABSOLUTE ADDRESS, which is returned by CP upon a successful completion of the lock request. Zeros will be returned if the lock request was not successful.

PARM
Is not used by the LOCK subfunction. (PARM is used only by the RELEASE AND LOCK subfunction.)

RC
Is a 2-byte return code which specifies the completion status of this individual Diagnose X'98' subfunction request. The following return codes may be returned for the LOCK subfunction:

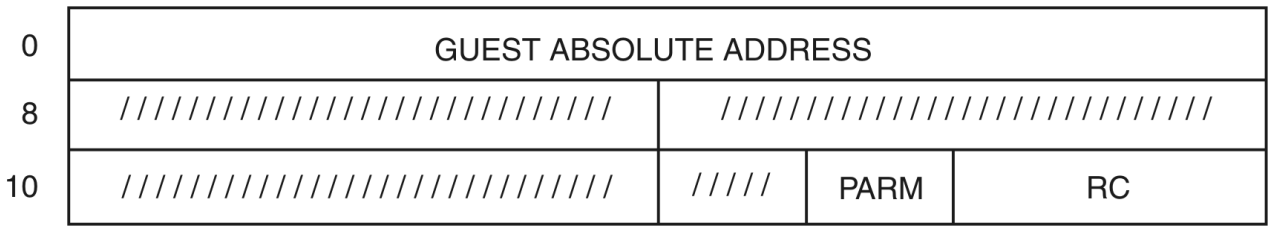
Return Code	Meaning
0 (X'00')	LOCK request completed successfully. The HOST ABSOLUTE ADDRESS field will contain the translated address.
1 (X'01')	Reserved.
2 (X'02')	Invalid guest address specified in the GUEST ABSOLUTE ADDRESS field.
3 (X'03')	Page is already locked by Diagnose X'98'.
4 (X'04')	Page is currently resident above 2 GB and cannot be moved to LOCK.
5-7 (X'05'-X'07')	Reserved for future use.
8 (X'08')	Paging error occurred while attempting to lock the page.
9-65535 (X'09'-X'FFFF')	Reserved for future use.

////////
Indicates fields reserved for future use.

UNLOCK Subfunction FCN Code X'04'

The UNLOCK subfunction unlocks multiple pages of virtual machine storage that were previously locked by Diagnose code X'98'. The real storage frames that are unlocked will become available again for normal CP paging.

The following structure is required for each individual UNLOCK request included in the array area of a FCN=X'04' Diagnose X'98' Multiple Request Block:



GUEST ABSOLUTE ADDRESS
Is the 8-byte guest absolute address that the program wants to unlock in CP storage. The 4 KB page which contains the specified address will be unlocked in CP storage. The value specified does not need to be on a page boundary.

PARM
Is not used by the UNLOCK subfunction. (PARM is used only by the RELEASE AND LOCK subfunction.)

RC
Is a 2-byte return code which specifies the completion status for this individual Diagnose X'98' subfunction request. The following return codes may be returned for the UNLOCK Subfunction:

Return Code	Meaning
0 (X'00')	UNLOCK request completed successfully.
1 (X'01')	Reserved.
2 (X'02')	Invalid guest address specified in the GUEST ABSOLUTE ADDRESS field.
3 (X'03')	Page is already unlocked.
4-7 (X'04'-X'07')	Reserved for future use.
8 (X'08')	Paging error occurred while attempting to unlock the page.
9-65535 (X'09'-X'FFFF')	Reserved for future use.

////////
Indicates fields reserved for future use.

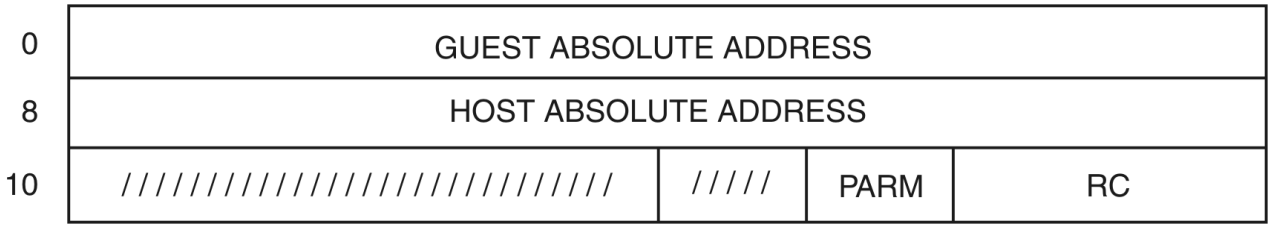
RELEASE AND LOCK Subfunction FCN Code X'20'

The RELEASE AND LOCK subfunction clears multiple pages of a virtual machine's absolute storage and locks them in real storage, based on the value specified in the PARM field, thus excluding the pages from future paging activities. A PARM value of X'01' indicates that the page can be locked anywhere in real storage, but the preference will be to lock it in storage above 2 GB. A PARM value of X'02' indicates that the page must be locked in real storage below 2 GB. If the page is already resident in storage above 2 GB, it must be moved.

When the RELEASE AND LOCK function is issued, it is identical to issuing multiple CP LOCK commands, obtaining real storage frames from the normal paging area.

An XC virtual machine can lock pages only within the host-primary address space. The Ay register is ignored.

The following structure is required for each individual RELEASE AND LOCK request included in the array area of a FCN=X'20' Diagnose X'98' Multiple Request Block:



GUEST ABSOLUTE ADDRESS
Is the 8-byte guest absolute address that the program wants to lock in host storage. The 4 KB page which contains the specified address will be locked in host storage. The value specified does not need to be on a page boundary.

HOST ABSOLUTE ADDRESS
Is the 8-byte host absolute address for the program-specified GUEST ABSOLUTE ADDRESS, which is returned by CP upon a successful completion of the lock request. Zeros will be returned if the lock request was not successful.

PARM
Specifies where in real storage the page will be locked:

X'01'
Lock the page anywhere in storage, with a preference of above 2 GB.

X'02'
Lock the page below 2 GB.

RC
Is a 2-byte return code which specifies the completion status of this individual Diagnose X'98' Subfunction request. The following return codes may be returned for the RELEASE AND LOCK subfunction:

Return Code	Meaning
0 (X'00')	RELEASE AND LOCK request completed successfully. The HOST ABSOLUTE ADDRESS field will contain the translated address.
1 (X'01')	Reserved.
2 (X'02')	Invalid guest address specified in the GUEST ABSOLUTE ADDRESS field.
3 (X'03')	Page is already locked by Diagnose X'98'.
4 (X'04')	Page is currently resident above 2 GB and could not be moved to LOCK.
5-7 (X'05'-X'07')	Reserved for future use.
8 (X'08')	Paging error occurred while attempting to lock the page.
9 (X'09')	Release of Shared Readable/Exclusive Readable is not allowed.
10 (X'0A')	Invalid PARM value specified.
11 (X'0B')	Storage could not be disassociated from the page.
12-65535 (X'0C'-X' FFFF')	Reserved for future use.

////////
Indicates fields reserved for future use.

Block Diagnose X'98' Request Condition Codes

The following are the condition codes which may be set on completion of a Block Diagnose X'98' Request:

Condition Code	Meaning
0	All requests completed successfully.
3	Non-zero global return code was returned.

Usage Notes

1. If you attempt to lock a prefix page in a virtual multiprocessor configuration, an error code of 2 is returned.

2. If you attempt to lock a page that has been mapped with the DEFINE function of the MAPMDISK macro, an error code of 2 is returned.
3. The LOCK subfunction requires an available real storage frame in the area the system uses for paging.
4. If you LOCK a page in a shared segment and do not issue an UNLOCK, the page will be unlocked when the last user of the shared segment releases the storage at LOGOFF (or uses another command which clears storage).
5. The LOCK command issued for a user ID's virtual pages operates independently of the DIAGNOSE code X'98' page locking facility. Pages locked by the LOCK command cannot be unlocked by DIAGNOSE code X'98', nor can pages locked by DIAGNOSE code X'98' be unlocked by the UNLOCK command.
6. Pages that are locked by DIAGNOSE code X'98' are not unlocked by the operation of the ADRSPACE ISOLATE and ADRSPACE PERMIT macros (see “ADRSPACE — Address Space Services” on page 797).
7. SSCH-Real does not support suspend/resume in channel programs. New I/O starts cannot be executed before previous I/O operations are concluded (early redrive is not supported).
8. At completion of an I/O operation started by DIAGNOSE code X'98', the CCW address returned to the virtual machine in the IRB on an XA, ESA, XC, or Z Test Subchannel instruction is a host absolute address.
9. The address-limit-checking facility of the Channel Subsystem is not available to users of the DIAGNOSE code X'98' SSCH-Real. The host absolute addresses used in the channel program are random addresses from the viewpoint of the virtual machine, so limit checking is meaningless. If the address-limit-checking control is set on in the ORB, it is ignored by the control program when scheduling the I/O.
10. If too many pages of real storage are locked, there may not be enough remaining frames to allow the system to operate efficiently. It is important for the system programmer to control the use of this facility.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'98' is given incorrect data:

Problem Encountered	Cause
Operand exception (z/VM SSCH-real subfunction only)	Any of the following: • Invalid subchannel ID • Invalid entries in the ORB.
Operation exception	The user does not have the correct directory authorization.
Specification exception	Any of the following: • Ry=R15, Rx=Ry, or Rx=Ry+1 • Unknown subfunction requested.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the ORB (SCCH-real subfunction).

Program Exceptions: These program exceptions can occur when issuing a Block X'98' request.

Problem Encountered	Cause
Operation exception	The virtual machine is not authorized to issue a Diagnose X'98' instruction. The virtual machine must have the OPTION DIAG98 specified in its CP system directory entry.

Problem Encountered	Cause
Specification exception	The following reasons will cause this exception to be reflected: • The subfunction code specified in Rx is not valid or supported by Diagnose X'98' . • The Diagnose X'98' Multiple Request Block specified by Ry is not on a fullword boundary. • Rx is the same register as Ry. • Rx is the same register as specified by Ry+1. • R15 was specified for the Ry register.
Access exception	The following reasons will cause this exception to be reflected: • Protection violation (key mismatch) when fetching or storing the Diagnose X'98' Multiple Request Block. • Addressing violation if the the Diagnose X'98' Multiple Request Block specified by Ry is not within the storage configuation.

DIAGNOSE Code X'9C' – Voluntary Time Slice Yield

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'9C' to notify the scheduler that a spin lock loop exists in your virtual machine. DIAGNOSE code X'9C' informs the scheduler that the remainder of the CPU time slice allocated to a virtual CPU is no longer useful and that another virtual CPU in your virtual machine should be favored for execution by the scheduler.

Entry Values:

Rx
In z/Architecture mode, bits 0-31 of register Rx are ignored, bits 48-63 contain the CPU address of the virtual CPU that should be favored by the scheduler for execution, and bits 32-47 are unused and should contain zeros. In ESA/390 mode, bits 16-31 of register Rx contain the CPU address of the virtual CPU that should be favored by the scheduler for execution and bits 0-15 are unused and should contain zeros.

Usage Notes

1. DIAGNOSE code X'9C' is useful when the operating system on one guest CPU is waiting to obtain a spin lock, and it is known which other guest CPU is holding the lock and needs to be run before the issuing guest CPU can again run productively.
2. The virtual CPU that issues DIAGNOSE code X'9C' will have its dispatching priority modified such that the target virtual CPU specified in Rx will generally run before the issuing virtual CPU is run again. If there is no other virtual CPU for this virtual machine, or the specified virtual CPU does not exist, then DIAGNOSE code X'9C' has no effect.
3. The effects of DIAGNOSE code X'9C' are temporary. After a brief time, the virtual CPU is scheduled as if the DIAGNOSE were never issued.

Responses

Program Exceptions: These program exceptions can occur when issuing DIAGNOSE code X'9C' :

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.

DIAGNOSE Code X'A0' – Obtain ACI Information

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'A0' to:

- Return the Access Control Interface (ACI) groupname for a given user ID (subcode 0)
- Determine whether an External Security Manager (ESM) is installed (subcode 8).
- Return External Security Manager (ESM) product information (subcode 72 (X'48')).

Note: Only the basic DIAGNOSE code X'A0' interface is described in this book. An external security manager (ESM) may extend that interface, for its own internal use or as an API.

DIAGNOSE code X'A0' cannot be issued in access-register mode in an XC virtual machine.

Entry Values:

Rx
For subcode 0, Rx contains the guest real address of a 16-byte doubleword aligned buffer. The first 8 bytes of this buffer contains a user ID (left-justified, followed by spaces) passed as input. The second 8 bytes are used by CP to pass the groupname back to the guest. For subcode 8 Rx is not used. For subcode 72 RX contains the guest real address of a doubleword aligned buffer to receive the ESM Product Information, this buffer should be 277 bytes in length to ensure that it will accommodate the maximum amount of data that might be returned.

Ry
Contains a right-adjusted subcode value:

- 0 — to request the return of the groupname associated with the input user ID
- 8 — to request a test to determine if an ESM is installed.
- 72 — to request ESM product information.

Note: An ESM may support additional subcode values. If you have an ESM installed, consult the ESM documentation for information on such extensions, their register usage, and applicable restrictions.

Exit Values:

Normal Exit:

Rx
Unchanged

Ry
Unchanged

- For subcode 0, the guest condition code is set to 0 on a successful request.
- For subcode 8, the guest condition code is set to 0 if the ACI program has been installed, or to 1 if it has not been installed.
- For subcode 72 returned ESM product information is in the following format:

  ESM name
  8–byte character string

ESM version
4–byte character string

ESM is active
Flag byte (bit 0 indicates ESM is active)

Vendor name
8–byte character string

ESM product information
1–byte length of the following variable length character string

Error with Exit:

Rx
Unchanged

Ry
Unchanged.

- For subcode 0, the user ID that was specified is invalid. The condition code is set to 1.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'A0' is given incorrect data:

Table 14. DIAGNOSE X'A0' Program Checks

Problem Encountered	Cause
Specification exception	Any of the following: • Invalid subcode specified (should be either 0, 8, or 72). • If you specify subcode 0 or 72, your buffer address in Rx is not on a doubleword boundary.
Privileged-operation exception	The virtual machine is in the problem state.
Special-operation exception	DIAGNOSE code X'A0' cannot run in an XC virtual machine that is in access register mode.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the user ID or store either the group name or the ESM product information.

DIAGNOSE Code X'A4' – Synchronous I/O (Standard CMS Blocksize)

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'A4' to perform synchronous input/output operations to a direct access device formatted by CMS and supported by z/VM. Using DIAGNOSE code X'A4' ensures that CP will construct the appropriate channel program for the device being accessed. DIAGNOSE code X'A4' operates in all supported virtual machine architectures. Results of the I/O operation are contained in the synchronous block I/O parameter list (HCPSBIOP) and in the condition code and return code.

Entry Values:

Rx
contains the real address of the synchronous block I/O parameter list (HCPSBIOP). The address must be on a word boundary; otherwise a specification exception will occur.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the synchronous block I/O parameter list (HCPSBIOP) and list of block

entries (see “Block Entries (SBILIST)” on page 141). If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the HCPSBIOP and SBILIST are in the host-primary address space.

Three different I/O operations can be performed through DIAGNOSE code X'A4': READ, WRITE, and FORMAT.

Operations are specified using the request-type field of the HCPSBIOP. The way to request each operation shown, see “Synchronous Block I/O Parameter List (HCPSBIOP)” on page 136.

Synchronous Block I/O Parameter List (HCPSBIOP)

The synchronous block I/O parameter list (HCPSBIOP) is the parameter list that DIAGNOSE code X'A4' uses in controlling the block I/O request. The parameters for read and write I/O operations include:

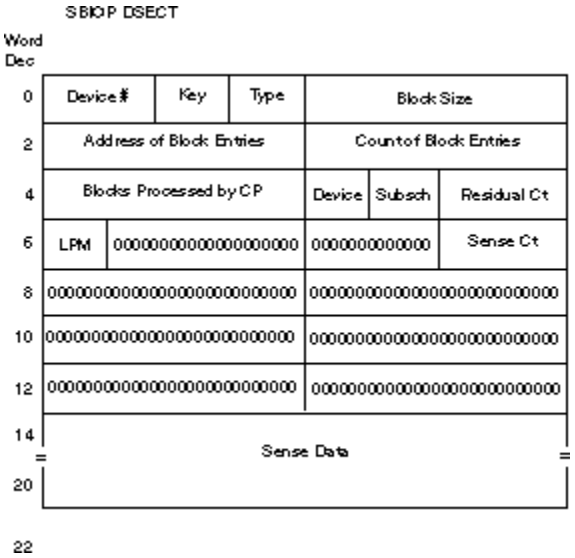
- Virtual device number
- Storage protection key
- Request type
- Block size
- Address of the block number/data address list
- Number of pairs in the block number/data address list
- Number of blocks processed by CP
- Device and subchannel status
- Residual count
- Sense data count
- Sense data.

These fields of the HCPSBIOP must be filled in:

- Virtual device number
- Storage protection key
- Request type
- Block size
- Address of the block entries list
- Number of pairs in the block entries list.

See the Responses section for the HCPSBIOP fields that are filled in upon completion of the operation.

The HCPSBIOP for the read and write request type is as follows (the HCPSBIOP COPY file is provided in the HCPGPI macro library):



The fields in this HCPSBIOP are defined as follows:

Device Number
Bits 0 through 15 of word 0 contains a virtual device number (1 to 4 digits) of the DASD to which this operation is targeted. The DASD must be fully supported. This field is filled in by the issuer of the DIAGNOSE, and is returned unchanged.

Storage Protection Key
Bits 16 through 19 of word 0 contain the subchannel key for all fetching of output data and for the storing of input data associated with the start function. This key is matched with a storage key during these storage references. Bits 20 through 23 must be zeros; otherwise an operand exception occurs. Bit 20 represents suspend control, which is not supported. The storage protection key is filled in by the issuer of the DIAGNOSE and is returned unchanged.

I/O Request Type
Bits 30 through 31 of word 0 contain the request type for this I/O operation. If bit 30 of word 0 is 1, then the I/O operation is to read data from DASD to storage. If bit 31 of word 0 is 1, then the I/O operation is to write data from storage to DASD. The user must set exactly one of bits 30 and 31 — the other must be zeros; otherwise, an operand exception occurs. Bits 24 through 29 of word 0 are reserved for future use and must be zeros; otherwise, an operand exception occurs.

In other words, the I/O request-type bits must be:

```
WRITE  = B'0000 0001'
READ   = B'0000 0010'
```

Anything else is invalid.

Block Size
Bits 0 through 31 of word 1 contain the size of the storage blocks for this request. The block size must be one of the following:

- 1024
- 2048
- 4096

If you set the block size to other than the above, processing of the DIAGNOSE terminates and a return code of 8 and a condition code of 2 are set. For a read or write request to CKD or ECKD DASD, if the block size does not match the physical block size of the DASD, the DIAGNOSE request ends with an incorrect length indication, and the results of the Read or Write are unpredictable.

Address of Block Entries
Bits 0 through 31 of word 2 designate the guest absolute address of the list of DASD block number and data address pairs for this request. This list resides in the host-primary address space and is

described in “Block Entries (SBILIST)” on page 141. All 32 bits of this field are used for the address; high-order bits beyond the address size must be zero.

The three rightmost bits of the block entries address must be zeros, specifying that the SBILIST is on a doubleword boundary; otherwise an operand exception occurs.

If the data address specifies a location protected against fetching or specifies a location outside the storage of the virtual machine, the processing of the DIAGNOSE terminates and a return code of 10 and a condition code of 2 are set.

Fetch-protection override and storage-protection override do not apply to references to the SBILIST.

Count of Block Entries
Bits 0 through 31 of word 3 contain the count of DASD block number and data address pairs in the SBILIST. For a read or write request, the maximum allowed is 500 and the minimum is 1; otherwise, the processing of the DIAGNOSE terminates and a return code of 11 and a condition code of 2 are set.

This is an unsigned binary number. The field is returned unchanged.

Blocks Processed by CP
Bits 0 through 31 of word 4 contains the number of blocks which were successfully processed by the Control Program. Blocks are processed sequentially, in the order the user provided. The data areas associated with unprocessed blocks are unpredictable.

Device Status
Bits 0 through 7 of word 5 identify the conditions in the device when the channel program ended. Each of the eight bits represents one condition, as defined in the Enterprise Systems Architecture/390 Principles of Operation. This is returned to the issuer.

Subchannel Status
Bits 8 through 15 of word 5 identify the conditions in the subchannel when the channel program ended. Each of the eight bits represents one condition, as defined in the Enterprise Systems Architecture/390 Principles of Operation. This is returned to the issuer.

Residual Count
Bits 16 through 31 of word 5 contain the residual count from the CCW in control when the channel program ended. This is returned to the issuer of the DIAGNOSE.

Logical Path Mask (LPM)
Bits 0 through 7 of word 6 contain a mask of paths that the channel subsystem can use to execute the I/O operation requested by the DIAGNOSE. If the issuer of the DIAGNOSE requires that the I/O be issued down a specific path, that path must be represented in this field. If this field contains zeros, the issuer of the DIAGNOSE does not wish the I/O operation to be restricted to a particular path or paths.

Note: A positional correspondence exists between the bit positions in the logical path mask and the channel path IDs in the subchannel.

Reserved
Bits 8 through 31 of word 6 are reserved for future use and must be zeros; otherwise, an operand exception occurs.

Bits 0 through 15 of word 7 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Sense Data Count
Bits 16 through 31 of word 7 contain the amount of sense data present if a unit check is indicated. This is returned to the issuer of the DIAGNOSE. This field is returned unchanged if unit check is not present.

Reserved
Bits 0 through 31 of words 8 through 13 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Sense Data
Bits 0 through 31 of words 14 through 21 contain the sense data (as limited by the sense data count) if a unit check is indicated. This is returned to the issuer of the DIAGNOSE. This field is returned unchanged if unit check is not present.

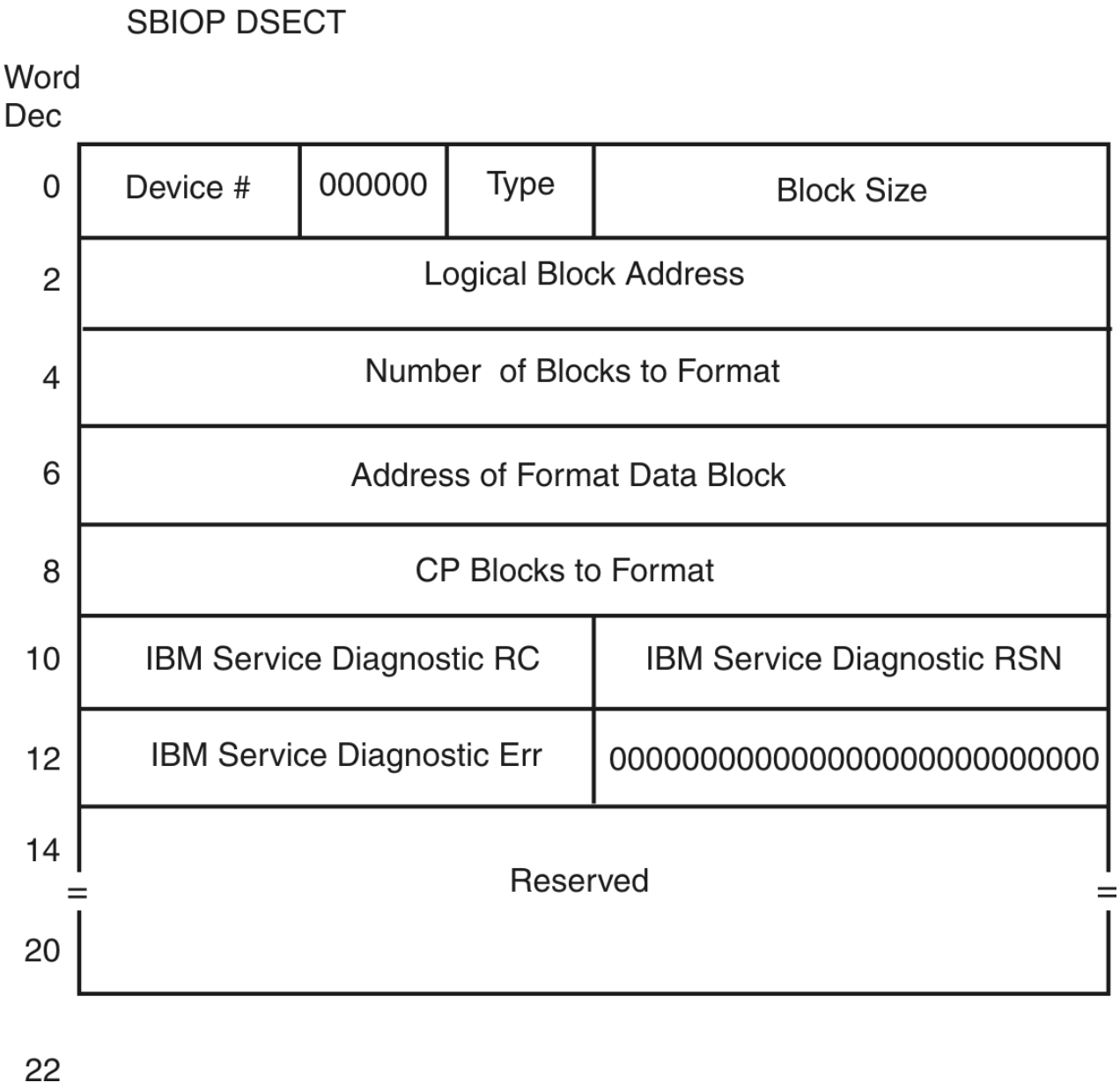
The parameters for format I/O operations include:

- Virtual device number
- Request type
- Block size
- Logical block address
- Number of blocks to format
- Address of format data block
- CP blocks to format
- IBM service diagnostic information.

These fields of the HCPSBIOP must be filled in:

- Virtual device number
- Request type
- Block size
- Logical block address.

The HCPSBIOP for the format request type is as follows (the HCPSBIOP COPY file is provided in the HCPGPI macro library):



The fields in this HCPSBIOP are defined as follows:

Device Number
Bits 0 through 15 of word 0 contains a virtual device number (1 to 4 digits) of the DASD to which this operation is targeted. The DASD must be fully supported. This field is filled in by the issuer of the DIAGNOSE, and is returned unchanged.

I/O Request Type
Bit 29 of word 0 contains the request type for this I/O operation. When bit 29 is 1, the I/O operation is to format data from DASD to storage. Bits 24 through 28, 30, and 31 of word 0 are reserved for future use and must be zeros; otherwise, an operand exception occurs.

In other words, the I/O request-type bit for format must be:

```
FORMAT = B'0000 0100'
```

Anything else is invalid.

Format is only valid for an emulated device that represents a real SCSI device associated with 2105 or 2107 attributes.

Block Size
Word 1 contains the size of the storage blocks for this request. The block size must be 512 when the device number is an emulated device that represents a real SCSI device.

If you set the block size to other than 512, processing of the DIAGNOSE terminates and a return code of 8 and a condition code of 2 are set.

Logical Block Address
Words 2 and 3 contain the logical block address where formatting will begin.

Number of Blocks to Format
Words 4 and 5 contain the request number of blocks to be formatted. When zero, formatting will continue until the end of the device is reached.

Address of Format Data Block
Words 6 and 7 contain the address of a 512-byte block which contains the data that will be written multiple times to the disk, starting at the logical block address and continuing for the number of blocks specified. When zero, zeros will be written.

CP Blocks to Format
Words 8 and 9 contain the number of blocks which CP will attempt to process when zero is specified in the Number of Blocks to Format field.

IBM Service Diagnostic Information
Words 10, 11, and 12 contain information which can be used by IBM Service when condition code 3 with return code 13 (X'0D') in register 15 is returned.

Reserved
Words 13 through 21 are reserved for future use and must be zeros; otherwise, an operand exception occurs.

Block Entries (SBILIST)

The block entries (SBILIST) describing the CMS block number and data address pairs have the following format:

The fields in the SBILIST are defined as follows:

Block Number
Bits 0 through 31 of word 0 contain the CMS block number for this request. This is provided by the issuer of the DIAGNOSE. Block numbers are assigned sequentially to CMS records, starting with zero.

Data Address
Bits 0 through 31 of word 1 designate the guest absolute address of the data for this request, in the host-primary address space. For a Write request, this is the location in storage from which the data is written to DASD. For a Read request, this is the location in storage to which the data read from DASD is placed.

If the data address specifies a location in a shared segment for a read request or specifies a location outside the storage of the virtual machine, the processing of the DIAGNOSE is terminated and return code 12, condition code 2 is set. The field in HCPSBIOP called *blocks processed by CP* indicates the number of list entries for which data transfer was successful. List entries are processed sequentially, in the order the user provided. The *blocks processed by CP* field is set to zero if the error was detected before any I/O was attempted.

Fetch-protection override, storage-protection override, and low-address protection do not apply to references to the data area.

Usage Notes

1. No I/O interrupts are returned by CP to the virtual machine; the DIAGNOSE instruction is completed only when the Read or Write commands associated with the DIAGNOSE are completed.

2. One way to figure out the block size of a CMS formatted disk automatically is to read the label record and look at the length field. The length field is in the fourth word of the record. For CKD or ECKD devices, the label record is in cylinder 0, track 0, record 3 (block number 2). For FBA devices, the label record is in the second 512 byte block (block number 1, using a blocksize of 512 bytes).
3. Use DIAGNOSE code X'A4' to perform synchronous I/O in a nonsynchronous environment.
4. Diagnose I/O operations issued to virtual Parallel Access Volume bases and aliases are randomly scheduled on any available, appropriate real base or alias device. Certain CCWs, such as Reserve and Release, require base or alias real device affinity. This is handled by CP as needed.
5. This DIAGNOSE code does not support HyperPAV alias devices.
6. This DIAGNOSE code is limited to a minidisk size of 65520 cylinders, regardless of formatted block size. If a program issues DIAGNOSE code X'A4' to a minidisk larger than 65520 cylinders, CP sets a condition code of 1 (cc=1) and places a return code of 4 in register 15.

Responses

Upon completion of a Read or Write operation, HCPSBIOP is updated with the number of blocks processed by CP, the device and subchannel status, the residual count, the sense data count and the sense data (if a unit check is indicated).

Condition Codes and Return Codes: Upon completion of DIAGNOSE code X'A4', you receive a condition code, along with a return code in register 15:

- If you receive a condition code of 0, the I/O operation was completed successfully. The return code in register 15 is 0 also.
- If you receive a condition code of 1, an error condition was detected which prevented complete execution of the channel program built for the DIAGNOSE I/O request. No I/O was performed unless the return code in Register 15 is 1.

  In addition, the return codes shown in Table 15 on page 142 are set in the guest's register 15:

  Table 15. DIAGNOSE Code X'A4' Return Codes. DIAGNOSE Code X'A4' Return Codes in the Guest's Register 15 with CC=1

Condition Code	Return Code in Register 15	Meaning
1	1 (X'01')	Device not attached (See Note below.)
1	2 (X'02')	Device is not a supported DASD.
1	3 (X'03')	Attempt to write to a read-only disk
1	4 (X'04')	A program issued a DIAGNOSE code X'A4' to a minidisk larger than 65520 cylinders.
1	5 (X'05')	Device is busy, or has an interrupt pending

  Note: This indicates that either the device has not been logically attached, or the physical path to the device has been lost. If the physical path to the device has been lost, then some portion of the channel program may have been executed. This is because CP may have used multiple real channel programs to perform the I/O operation. The *blocks processed by CP* field in HCPSBIOP must be examined to determine how much, if any, of the I/O request was completed. Device verification is required to determine why the physical path to the device was lost.
- If you receive a condition code of 2, a specification exception was detected (the HCPSBIOP was not set up correctly). No I/O was performed, except when the return code in register 15 is 16. One of the return codes shown in Table 16 on page 143 is set in the guest's register 15:

Table 16. DIAGNOSE Code X'A4' Return Codes

Condition Code	Return Code in Register 15	Meaning
2	7 (X'07')	Guest DASD block number is invalid
2	8 (X'08')	Blocksize equals zero, or is unsupported
2	9 (X'09')	The number of blocks specified exceeds the size of the mdisk
2	10 (X'0A')	The address of the SBILIST is invalid, or the SBILIST is fetch-protected
2	11 (X'0B')	The number of buffer list entries was not a positive number within the range of 1 to 500
2	12 (X'0C')	One of the buffers identified in the buffer list is not within the guest's storage (for example, its address is invalid)
2	16 (X'10')	Unsolicited status was pending at the real subchannel or device (See note below.)

Note: The channel program may not have completed, and thus the ending status may not pertain to this I/O request. The *blocks processed by CP* field in the HCPSBIOP must be examined to determine how much, if any, of the I/O request was completed. Device verification will be required to determine why unsolicited status was pending at the real subchannel or device.

- If you receive a condition code of 3, an unrecoverable I/O error occurred. If the device status field indicates a unit check, sense data is stored in the sense data field, and the amount of sense data is stored in the sense data count field. One of the return codes shown in Table 17 on page 143 is set in the guest's register 15:

Table 17. DIAGNOSE Code X'A4' Return Codes in the Guest's Register 15 with CC=3

Condition Code	Return Code in Register 15	Meaning
3	13 (X'0D')	A permanent I/O error or a soft abend occurred, or the I/O was terminated at the user's request by entering an exigent command.
3	15 (X'0F')	Paging error occurred; channel data check status is set, unless the paging error occurred while accessing the SBILIST.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'A4' is given incorrect input data:

Problem Encountered	Cause
Specification exception	The address HCPSBIOP (specified in Rx) is not on a word boundary.

Problem Encountered	Cause
Operand exception	Any of the following: • In word 0 of the HCPSBIOP, bits 20 to 23 are not all set to zero. • In word 0 of the HCPSBIOP, both bits 30 and 31 (I/O request type) are set to 0 or both are set to 1. • In the HCPSBIOP, the address of the block entries list is not on a doubleword boundary. • In the HCPSBIOP, the bits in these reserved fields are not all set to zeros: – word 6, bits 8 to 31 – word 7, bits 0 to 15 – words 8 through 13
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch or store the HCPSBIOP.

DIAGNOSE Code X'A8' – Synchronous I/O (for All Devices)

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'A8' to perform synchronous input/output operations to all fully supported devices, except channel-to-channel adapters, consoles, and graphics devices.

DIAGNOSE code X'A8' uses the synchronous general I/O parameter list (HCPSGIOP) to receive and return data.

Entry Values:

Rx
Contains the guest real address of the synchronous general I/O parameter list (HCPSGIOP). The HCPSGIOP must be on a word boundary; otherwise a specification exception occurs.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the synchronous general I/O parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

To perform general I/O operations using DIAGNOSE code X'A8', the HCPSGIOP needs to have the following fields filled in: virtual device number, storage protection key, request flag, and channel program address.

Synchronous General I/O Parameter List (HCPSGIOP)

The synchronous general I/O parameter list (HCPSGIOP) is the parameter list that DIAGNOSE code X'A8' uses in controlling the general I/O request. These parameters include:

- Virtual device number
- Storage protection key
- Request flag
- Channel program address
- CCW address at interrupt

- Device and subchannel status
- Residual count
- Sense data count
- Sense data.

The synchronous general I/O parameter list has the following format (the HCPSGIOP COPY file is provided in the HCPGPI macro library):

HCPSGIOP DSECT

Word (decimal)

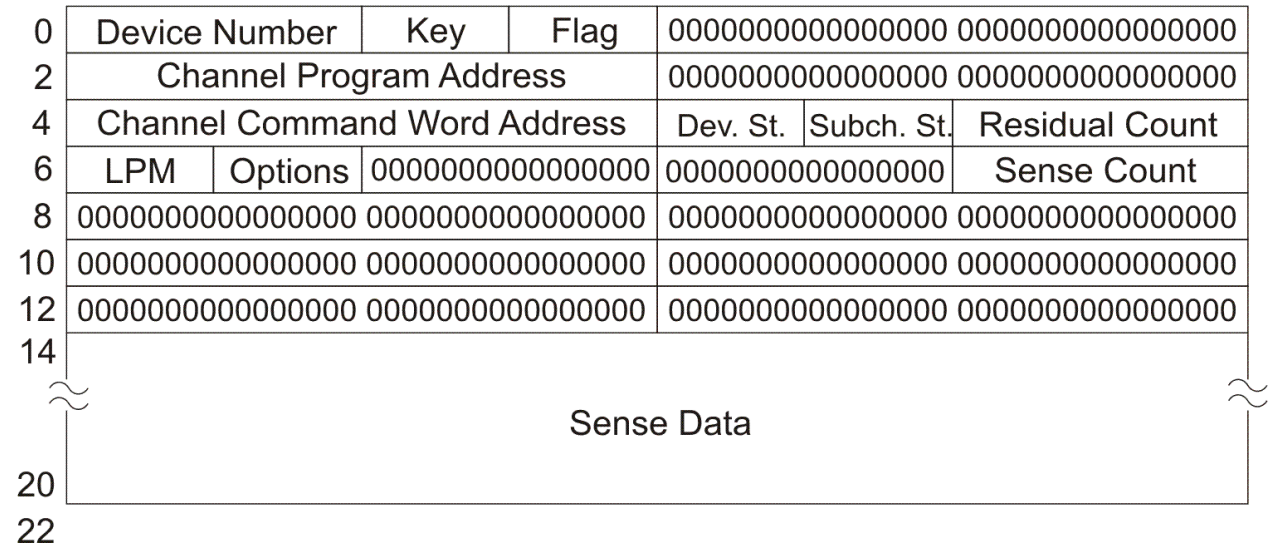


Figure 14. DIAGNOSE X'A8' Synchronous General I/O Parameter List (HCPSGIOP) Format

The fields in the HCPSGIOP are defined as follows:

Device Number
Bits 0 through 15 of word 0 contains a virtual device number (1-4 digits) of the device to which this operation is targeted. The device must be fully supported, but virtual channel-to-channel adapters, virtual console devices, and graphics devices are not allowed. This field is filled in by the issuer of the DIAGNOSE, and is returned unchanged.

Storage Protection Key
Bits 16 through 19 of word 0 contain the subchannel key for all fetching of CCWs, IDAWs and output data and for the storing of input data associated with the start function. This key is matched with a storage key during these storage references. Bits 20 through 23 must be zeros; otherwise an operand exception occurs. Bit 20 represents suspend control, which is not supported. The storage protection key is filled in by the issuer of the DIAGNOSE and is returned unchanged.

I/O Request Flag
Bits 24 through 31 of word 0 contain the request flag for this I/O operation.

Bit 24 of word 0 specifies the format of the channel-command words (CCWs) which make up the channel program designated by the channel program address field. If bit 24 is zero, format-0 CCWs are specified. If bit 24 is one, format-1 CCWs are specified. Both format CCWs are allowed in all virtual machines.

Bits 25 through 29 of word 0 are reserved for future use and must be zeros; otherwise an operand exception occurs. This field is filled in by the issuer of the DIAGNOSE and is returned unchanged.

If bit 30 (SGIDAWF2) is on, it indicates that format-2 IDAWs are used in the channel program for all CCWs that have the IDAW flag set to one. Otherwise, format-1 IDAWs are used.

If bit 31 (SGIDAW2K) is on, it indicates that format-2 IDAWs use a storage block size of 2K. Otherwise, format-2 IDAWs use a storage block size of 4K. SGIDAW2K is ignored if SGIDAWF2 is off.

Both bit 30 and bit 31 may be on if and only if bit 24 is also on. Otherwise, an operand exception is recognized.

Reserved
Bits 0 through 31 of word 1 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Channel Program Address
Bits 0 through 31 of word 2 designate the location of the first CCW in the host-primary address space. If format-0 CCWs have been specified in bit 24 of word 0, then bits 0 through 7 of word 2 must be zeros. If format-0 CCWs have been specified and bits 0 through 7 do not contain zeros, an operand exception occurs. If format-1 CCWs have been specified, then bit 0 of word 2 must be zero. If bit 0 is not zero, an operand exception occurs. This field is filled in by the issuer of the DIAGNOSE and is returned unchanged.

The three rightmost bits of the channel-program address must be zeros, specifying the CCW on a doubleword boundary; otherwise an operand exception occurs.

If the CCW address specifies a location protected against fetching or specifies a location outside the storage of the virtual machine, the processing of the DIAGNOSE is terminated and return code 13, condition code 3 is set. The subchannel status indicates a program-check condition and the CCWA = CPA + 8.

Reserved
Bits 0 through 31 of word 3 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Channel Command Word Address at Interrupt
Bits 0 through 31 of word 4 form an absolute address. The address indicated represents the CCW + 8 of the last executed CCW. This is returned to the issuer of the DIAGNOSE.

Device Status
Bits 0 through 7 of word 5 identify the conditions in the device when the channel program ended. Each of the eight bits represents one condition, as defined in the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf). This is returned to the issuer of the DIAGNOSE.

Subchannel Status
Bits 8 through 15 of word 5 identify the conditions in the subchannel when the channel program ended. Each of the eight bits represents one condition, as defined in the Enterprise Systems Architecture/390 Principles of Operation (publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf). This is returned to the issuer of the DIAGNOSE.

Residual Count
Bits 16 through 31 of word 5 contain the residual count from the CCW in control when the channel program ended. This is returned to the issuer of the DIAGNOSE.

LPM - Logical Path Mask
Bits 0 through 7 of word 6 contain a mask of paths that the channel subsystem should be permitted to use to execute the I/O operation requested by the DIAGNOSE. If the issuer of the DIAGNOSE requires that his I/O be issued down a specific path, he must represent that path in this field. If this field contains zeros, this indicates that the issuer of the DIAGNOSE does not wish the I/O operation to be restricted to a particular path or paths.

Note that there is a positional correspondence between the bit positions in the logical path mask and the channel path IDs in the subchannel.

Options
Bits 8 through 15 of word 6 contain settings for certain options for this I/O operation.

If bit 8 is on, the caller can use the 28-bit extended addressing format and access a minidisk above cylinder 65519.

Bits 9 through 15 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Reserved
Bits 16 through 31 of word 6 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Bits 0 through 15 of word 7 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Sense Data Count
Bits 16 through 31 of word 7 contain the amount of sense data present if a unit check is indicated. This is returned to the issuer of the DIAGNOSE. This field is returned unchanged if unit check is not present.

Reserved
Bits 0 through 31 of words 8 through 13 are reserved for future use and must be zeros; otherwise an operand exception occurs.

Sense Data
Bits 0 through 31 of words 14 through 21 contain the sense data (as limited by the Sense Data Count) if a unit check is indicated. This is returned to the issuer of the DIAGNOSE. This field is returned unchanged if unit check is not present.

Usage Notes

1. No I/O interruptions are returned by CP to the virtual machine. The DIAGNOSE instruction is complete only when the channel program associated with the DIAGNOSE is complete.
2. If neither DIAGNOSE code X'A4' nor DIAGNOSE code X'18' can be used, then the channel programming capability of the device being accessed must be determined and the appropriate channel program constructed.
   a. Use DIAGNOSE code X'24' or X'210' to determine if the device is in an ECKD-capable subsystem.
   b. Generate a CKD, ECKD or FBA channel program as appropriate for the device being accessed.
   c. Expand sense error analysis and recovery procedures to include the 32 byte format returned by ECKD subsystems.
3. Use DIAGNOSE code X'A8' to perform synchronous I/O in a nonsynchronous environment.
4. Diagnose I/O operations issued to virtual Parallel Access Volume bases and aliases are randomly scheduled on any available, appropriate real base or alias device. Certain CCWs, such as Reserve and Release, require base or alias real device affinity. This is handled by CP as needed.
5. DIAGNOSE code X'A8' can be used on a minidisk that resides on an Extended Address Volume (EAV).

Responses

Upon completion of DIAGNOSE code X'A8', the HCPSGIOP is returned with information in the following fields:

- Device status
- Subchannel status
- Residual count
- Address of CCW at interrupt
- Sense data count
- Sense data if a unit check is present in the device status field.

Condition Codes and Return Codes: Upon completion of DIAGNOSE code X'A8', you also receive a condition code and, if there is an error, a return code in register 15.

If you receive a condition code of 0, the I/O operation completed successfully and register 15 remains unchanged.

If you receive a condition code of 1, an error condition was detected which prevented execution of the guest virtual machine's channel program. Real I/O may not have been performed. In addition, one of the return codes shown in Table 18 on page 148 is set in the guest's register 15:

Table 18. DIAGNOSE Code X'A8' Return Codes in the Guest's Register 15 with CC=1

Condition Code	Return Code in Register 15	Meaning
1	1 (X'01')	Device not attached. **Note:** This indicates that either the device has not been logically attached, or that the physical path to the device has been lost. If the physical path to the device has been lost, then some portion of the channel program may have been executed. This is because CP may have used multiple real channel programs to perform the I/O operation.
1	2 (X'02')	Device is not supported.
1	5 (X'05')	Device is busy, or has an interrupt pending.

If you receive a condition code of 2, an exception condition was detected. Return code 16 is set in the guest's register 15, indicating that an unsolicited status was pending at the real subchannel or device.

The channel program may not have completed, and thus the ending status may not pertain to this I/O request. This is because CP may have used multiple real channel programs to perform the I/O operation. The channel command word at interrupt field in the HCPSGIOP must be examined to determine how much, if any, of the I/O request was completed. Device verification is required to determine why unsolicited status was pending at the real subchannel or device.

If you receive a condition code of 3, this indicates that an unrecoverable I/O error occurred or the I/O was terminated at the user's request by entering an exigent command. If the I/O error resulted in a unit check, then sense data is stored in the sense data field and the amount of sense data stored is in the sense data count field. Return code 13 is set in the guest's register 15, indicating that a permanent I/O error occurred.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'A8' is given incorrect input data:

Problem Encountered	Cause
Specification exception	The address of the HCPSGIOP (specified in Rx) is not on a word boundary.
Operand exception	Any of the following: • These reserved fields in the HCPSGIOP are not all set to zeros: word 0, bits 20-23, 25-31; word 3; word 6; word 7, bits 8-15; and words 8 through 13. • In the HCPSGIOP, bit 24 of word 0 (CCW format flag) is 0 and bits 0 through 7 of word 2 (channel program address) are not all zeros. • In the HCPSGIOP, bit 24 of word 0 (CCW format flag) is 1 and bit 0 of word 2 (channel program address) is not zero.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch or store the HCPSGIOP.

DIAGNOSE Code X'B0' – Access Re-IPL Data

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'B0' to obtain diagnostic information about the cause of the automatic re-IPL of your virtual machine. The data returned by the system also includes the IPL statement from the directory entry for the issuing user.

Entry Values:

Rx
Contains the guest real address of the buffer for the output data.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer for the output area. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
Contains the length of the buffer. Any nonnegative length is allowed. To avoid possible truncation of data, a buffer of at least 90 bytes is recommended.

Exit Values:

Ry
Contains a completion code, as follows:

Code
Meaning

0
Re-IPL information and IPL statement information are returned in buffer. (The IPL statement information may be the null string.)

4
No re-IPL information is available. IPL statement information is returned in the buffer. (The IPL statement information may be the null string.) Either the protected application environment is not active, or the IPL was initiated by the user.

8
No information is available. A paging or storage error has occurred.

Buffer Content: Successful completion of the request may result in output consisting of re-IPL information, an IPL statement from the directory, both types of information, or neither. Only as much information is returned as fits in the output area supplied.

Re-IPL Information

If the completion code is zero, the buffer contains re-IPL information consisting of the following:

Byte 0: Error code

- The codes and the errors to which they correspond are as follows:

Code
CP Meaning

X'01'
CP entered; Disabled wait PSW is provided in a message in the buffer

X'02'
CP entered; External interrupt loop

X'03'
CP entered; Paging error

X'04'
CP entered; Program interrupt loop

X'07'
CP entered; Complex interrupt loop

X'08'
System soft abend, abend code is provided in a message in the buffer

X'09'
CPU ... stopped; Check-stop state entered

X'0A'
Page zero damaged

X'0B'
An error occurred but CP was unable to provide a message because of a paging error.

Bytes 1-n: Variable data

- The additional variable length data, if present, depends on the particular error condition. The variable length data is shown as follows:

  Code
  Variable Data

  X'01'
  for z/Architecture mode guests, 16-byte binary disabled wait state PSW

  for non-z/Architecture mode guests, 8-byte binary disabled wait state PSW

  X'02'
  for z/Architecture mode guests, 16-byte binary external interrupt old PSW

  for non-z/Architecture mode guests, 8-byte binary external interrupt old PSW

  X'03'
  None

  X'04'
  for z/Architecture mode guests, 16-byte binary program interrupt old PSW

  for non-z/Architecture mode guests, 8-byte binary program interrupt old PSW

  X'07'
  None

  X'08'

  EBCDIC CPU number
  EBCDIC abend code

  X'09'
  EBCDIC CPU number

  X'0A'
  None

  X'0B'

  CP module name
  EBCDIC offset into module

- EBCDIC variable data items (for codes above X'07') are delimited by 1-byte binary fields. The field between multiple data items is X'00'. The field following the final data item is X'01'. For example, the buffer contents returned for a soft abend is the following (assuming the buffer is large enough):
  - X'08'
  - CPU number
  - X'00'

- Abend code
- X'01'.

IPL Statement Information

If the completion code is 0, the IPL statement information immediately follows the re-IPL information in the buffer. If the completion code is 4, the IPL statement information is placed at the start of the buffer.

Byte 0: Length of IPL directory statement

This data is present if a re-IPL occurred. It is zero if the user has no IPL statement in the directory, or if the output area is too small to contain at least one byte of text. If the entire IPL statement does not fit in the area, this length is that of the text truncated to fit in the available space.

Bytes 1-m: Text of IPL directory statement

This data is present if the preceding length byte is nonzero.

Usage Notes

1. If Ry specifies a length of zero on entry, the contents of Rx are not examined. Ry returns with the value zero if an automatic re-IPL has occurred; otherwise, Ry contains the value four.
2. CP does not guarantee the validity of the IPL statement content for the virtual machine environment issuing DIAGNOSE code X'B0'. It is the responsibility of the program issuing DIAGNOSE code X'B0' to use the information contained in it appropriately.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'B0' is given incorrect data:

Problem Encountered	Cause
Specification exception	The buffer length is negative.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to store the re-IPL data.

DIAGNOSE Code X'B4' – Read/Write/Erase the Virtual Printer XAB

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'B4' to read, write, and erase the external attribute buffer (XAB) for a spooled printer device. A copy of this external attribute buffer is added to each spool file created on the device. Specific information about the XAB may be found in “External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'” on page 979.

Entry Values:

Rx
Contains the buffer address

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
Bytes 0 and 1 contain the length of the buffer in bytes (0=ERASE). Bytes 2 and 3 contain the device number of the virtual printer.

Ry+1
Bytes 2 and 3 contain subcode X'0000' (Read) or X'0004' (Write/Erase).

Exit Values:

READ - Subcode X'0000'
CC=0 - Successful

Ry
Bytes 0 and 1 contain the length of the XAB read, 0 if there is no XAB. Bytes 2 and 3 are unchanged.

Ry+1
Byte 0 is X'00'. Bytes 1 through 3 are unchanged.

READ - Subcode X'0000'
CC=2 - Unsuccessful

Ry
Bytes 0 and 1 contain the length of the XAB if it is too large for the buffer. Bytes 2 and 3 are unchanged.

Ry+1
Byte 0 contains the return code; see Table 19 on page 152. Bytes 1 through 3 are unchanged.

WRITE/ERASE - Subcode X'0004'
CC=0 - Successful

Ry+1
Byte 0 contains X'00'. Bytes 1 through 3 are unchanged.

WRITE/ERASE - Subcode X'0004'
CC=2 - Unsuccessful

Ry+1
Byte 0 contains the return code; see Table 19 on page 152. Bytes 1 through 3 are unchanged.

Responses

Condition Codes for Normal Exit: The condition code for a normal exit is 0, indicating that the reading, writing, or erasing of the XAB was successful.

Condition Codes and Return Codes for Exit with Error: The condition code for an error exit is 2, indicating that the reading, writing, or erasing of the XAB was unsuccessful. The return code is shown in Table 19 on page 152.

Table 19. DIAGNOSE Code X'B4' Return Codes

Return Code in Ry +1	Meaning
4 (X'04')	The device does not exist, or is not a spooled virtual printer.
8 (X'08')	The buffer is too small. (returned for READ, subcode X'0000', only).
12 (X'0C')	The buffer length is invalid.
20 (X'14')	CP paging or I/O error
28 (X'1C')	Invalid subcode
32 (X'20')	The buffer address is zero or negative.
36 (X'24')	Not used
40 (X'28')	No XAB is to be erased (returned for Write/Erase, subcode X'0004', only).

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'B4' is given incorrect data:

Problem Encountered	Cause
Specification exception	Ry is R15.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch from (WRITE function) or store to (READ function) the buffer.

DIAGNOSE Code X'B8' – Spool File XAB Manipulation

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'B8' to read, write, or erase the external attribute buffer (XAB) of a spool file on the printer queue, or to read the XAB of a file on the reader queue. It also can be used to set user HOLD status or user NOHOLD status.

Note: The external attribute buffer (XAB) is a control block that contains data the user creates to specify additional information about a print file. Each print file has its own XAB, and CP has the facilities to maintain the XABs. For more information on the XAB, refer to “External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'” on page 979.

Entry Values:

Rx
Bytes 0 through 3 contain the buffer address

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
Bytes 0 and 1 contain the length of the buffer in bytes (0=ERASE). Bytes 2 and 3 contain the spool file ID.

Ry+1
Byte 1 contains flag bits which have the following meanings:

Flag
Meaning

X'01'
Place the spool file in user HOLD status

X'02'
Place the spool file in user NOHOLD status

X'04'
Look for the spool file on the reader queue (READ function only).

Flags X'01' and X'02' may not be specified together. If neither is specified, the file's HOLD status is left unchanged.

Bytes 2 and 3 of Ry+1 contain subcode X'0000' (READ) or X'0004' (WRITE/ERASE).

Exit Values:

READ - Subcode X'0000'
CC=0 - Successful

Ry
Bytes 0 and 1 contain the length of the XAB Read; 0 if there is no XAB

Ry+1
Byte 0 contains X'00'

READ - Subcode X'0000'
CC=2 - Unsuccessful

Ry
Bytes 0 and 1 contain the length of the XAB if it is too large for the buffer

Ry+1
Byte 0 contains the return code, see Table 20 on page 154.

WRITE/ERASE - Subcode X'0004'
CC=0 - Successful

Ry+1
Byte 0 contains X'00'

WRITE/ERASE - Subcode X'0004'
CC=2 - Unsuccessful

Ry+1
Byte 0 contains the return code, see Table 20 on page 154.

Usage Note

You may not be authorized to issue this DIAGNOSE code if an external security manager is installed on your system. For additional information, contact your security administrator.

Responses

Condition Codes for Normal Exit: A normal exit condition is a condition code of 0, indicating that the reading, writing, or erasing of the XAB of a spool file on the printer queue (or the reading of the XAB of a file on the reader queue) was successful.

Condition Codes and Return Codes for Exit with Error: An error exit condition is a condition code of 2, indicating that the reading, writing, or erasing of the XAB of a spool file on the printer queue (or the reading of the XAB of a spool file on the reader queue) was unsuccessful. The return code is shown in Table 20 on page 154.

Table 20. Return Codes

Return Code in Ry +1	Meaning
4 (X'04')	The spool ID is invalid or does not exist.
8 (X'08')	The length of the XAB buffer is greater than the user buffer. (See note)
12 (X'0C')	The buffer length is invalid.
20 (X'14')	CP paging or I/O error
24 (X'18')	The specified process flag is invalid.
28 (X'1C')	Invalid subcode
32 (X'20')	The buffer address is zero or negative.
36 (X'24')	Not used
40 (X'28')	No XAB is to be erased (returned for WRITE/ERASE, subcode X'0004', only).
44 (X'2C')	The XAB is not available because the file is marked for purge or has been converted.

Note: Any special processing requested by the flag byte is completed.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'B8' is given incorrect data:

Problem Encountered	Cause
Specification exception	Ry is R15.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See"Access Exceptions" on page 8.	An error occurred trying to fetch from (WRITE function) or store into (READ function) the buffer.

DIAGNOSE Code X'BC' – Open and Query Spool File Characteristics

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'BC' opens a spool file (if it is not already open) for a spooled reader device and returns all information about the spool file to a user-specified buffer. CP selects the spool file according to the rules governing the CLASS and HOLD status. For these rules, refer to *z/VM: CP Commands and Utilities Reference* and *z/VM: Virtual Machine Operation*. This means CP opens only spool files with the same class designation as the virtual reader.

Entry Values: Set the input registers up as follows when invoking DIAGNOSE code X'BC':

Rx
contains the guest real address of a buffer. Rx cannot be register 15.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Rx+1
contains the length of the buffer, in bytes.

Ry
Bytes 0 and 1 contain the subcode. Bytes 2 and 3 contain the virtual device number of the spooled reader device. Ry cannot be register 15.

Subcode
Meaning

X'0000'
This subcode provides information in character format only. This subcode is compatible with VM/SP, VM/SP HPO, and VM/ESA (370 Feature).

X'0004'
This subcode provides information in character format, where appropriate, and binary format for numeric information. This format is not compatible with VM/SP, VM/SP HPO, and VM/ESA (370 Feature).

Exit Values:

Subcode X'0000': Depending on the specified buffer length, the user's buffer contains as much of the following information shown as possible: (Bytes means character length, in bytes.)

Bytes
Description

4
Spool file ID (EBCDIC)

8
File originator

1
Class

3
Type: RDR, PRT, PUN, CON

8
Number of records (EBCDIC)

3
Number of copies (EBCDIC)

12
File name

12
File type

8
Date: mm/dd/yy

8
Time

8
Distribution

4
Status -- 'NONE'

8
FORM -- User forms

8
Destination

4
Flash name

3
Flash count (EBCDIC)

4
FCB -- Forms control buffer

4
CMOD -- Character modification

1
Character modification count (EBCDIC)

3
Load 3800 -- 'ANY' 'BEG' 'NO'

16
CHARS -- Character Arrangement Tables

8
SIZE -- Number of pages

8
SECLABEL - Security label of file.

10
Full year date: mm/dd/yyyy

10
ISO date: yyyy-mm-dd

Subcode X'0004': Depending on the specified buffer length, the user's buffer contains as much of the following information as possible. (Bytes means character length, in bytes.) Note that CP returns character information, where appropriate, and binary format for numeric data.

Bytes	Description
1	Control block update level identifier
1	Maximum length of data available in doublewords
1	Spool file CLASS
1	*** RESERVED FOR IBM USE ***
1	Copy count
1	Page copy count
1	Flash count
1	Modify number
2	Spool file ID (in hexadecimal)
2	Logical Record length
4	Record count
4	Number of spool data blocks
4	*** RESERVED FOR IBM USE ****
4	Type: RDR, PRT, PUN, CONS
4	Spool file ID in EBCDIC
8	File owner
8	File originator
8	File name
8	File type
8	Date: mm/dd/yy
8	Time: hh-mm-ss
8	Distribution code

8
Destination value

8
User form name

8
Operator form name

4
FCB name

4
3800 Load CCWs: 'NO '|'BEG '|'ANY '

4
Flash name

4
Modify name

16
Character set names

8
SECLABEL - Security label of file.

10
Full year date: mm/dd/yyyy

10
ISO date: yyyy-mm-dd

Usage Note

If an external security manager is installed, the user may not receive all the information about the file. If the external security manager denies access to the file, the only fields of the response that will contain information are the spool file ID, user ID, class, date, time and status fields. All other fields will contain asterisks.

Responses

Condition Codes and Return Codes: Upon completion, DIAGNOSE code X'BC' returns the following condition codes (found in the user's PSW) and return codes (found in Ry+1):

Condition Code	Return Code in Ry +1	Meaning
0	0 (X'00')	Data transfer successful, file opened
1	0 (X'00')	Data transfer successful, file already opened
2	N/A	No file is found
3	4 (X'04')	The device address is invalid
3	8 (X'08')	The device type is invalid
3	18 (X'12')	Device busy, not ready, or a real reader
3	22 (X'16')	A paging I/O error is received.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'BC' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • The user's buffer length is less than or equal to 0 • Either Rx or Ry is specified as R15 • There is any overlap of Rx, Rx+1, Ry, or Ry+1 • The subcode specification is not X'00' or X'04'.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See"Access Exceptions" on page 8.)	An error occurred trying to store into the buffer.

DIAGNOSE Code X'C8' – Set Language

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'C8' initiates the SET of the national language for the virtual machine session. It sets only the CP language. CP uses this language to issue most CP system messages.

For CMS applications, the preferred user interface is the CMS SET LANGUAGE command.

For more information on the languages supported on your z/VM system, see *z/VM: Installation Guide*.

Entry Values:

Rx and Rx+1
Language identifier specifying the language to be set for the virtual machine and the *langid* specified in the message repository. The language identifier is specified using the 1- to 5-character *langid*, left-justified (that is, the first four characters in Rx and fifth character in the high-order byte of Rx+1). If the language identifier is less than five characters, it should be padded on the right with blanks. Rx cannot be register 15.

Ry
X'00xxxxxx'–the function code (X'00' in the high-order byte, with the remaining bytes unused).

Exit Values: When processing of the SET function completes, the first five bytes of the register pair Rx, Rx +1 contain the language identifier for the language that has been set for CP messages (this language may or may not be the same as the language set before the DIAGNOSE code was issued), left-justified and padded on the right with blanks as required.

Responses

Return Codes: The low-order byte of register Ry contains one of the following return codes (the contents of the remaining bytes of Ry are unpredictable):

Return Code in Ry +1	Meaning
0 (X'00')	The language requested has been set.
20 (X'14')	A paging error occurred while an attempt was being made to read the first page of the requested message repository. The current language used for CP messages is unchanged.
28 (X'1C')	No message repository could be found for the specified language identifier. The current language used for CP messages is unchanged.
32 (X'20')	The selected repository does not appear to be a valid message repository.

Return Code in Ry +1	Meaning
36 (X'24')	Unrecoverable error, causing a soft abend. The current language used for CP messages is unchanged.
40 (X'28')	The NLS lock could not be obtained. This is a temporary error. Retry the command.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'C8' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • Rx or Rx+1 is the same register as Ry • Rx is register 15. • The function code is invalid.
Privileged-operation exception	The virtual machine is in the problem state.

DIAGNOSE Code X'CC' – Save Message Repository

Privilege Class: E

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'CC' to initiate the SAVE function for the CP message repository. If the SAVE operation completes successfully, then DIAGNOSE code X'C8' can be used to set that language.

The preferred user interface is the CMS LANGGEN command.

Entry Values:

Rx and Rx+1
The language identifiers specifying the language to be saved and the language identifiers specified in the message repository. The language identifiers are specified using the 1- to 5-character language identifier, left-justified (that is, the first four characters in Rx and the fifth character in the high-order byte of Rx+1). If the language identifiers are less than five characters, they should be padded on the right with blanks. Rx cannot be register 15.

Ry
X'00xxxxxx' – the function code (X'00' in the high-order byte, with the remaining bytes unused). Ry cannot be register 15.

Ry+1
The guest real address where the compiled CP message repository that is to be saved has been loaded. This address must be on a page boundary.

Key-controlled protection does not apply to accesses to the message repository data.

Ay+1
Is used only by XC virtual machines in access-register mode. Ay+1 contains the ALET for the address space containing the CP message repository.

Responses

Return Codes : When processing of the SAVE function is completed, the low-order byte of Ry contains the return code (the contents of the remaining bytes being unpredictable).

Return Code in Ry	Meaning
0 (X'00')	The entire CP message repository is successfully saved.

Return Code in Ry	Meaning
16 (X'10')	The repository cannot be saved because of insufficient resources (spool file IDs or spool space).
24 (X'18')	A storage or paging I/O error occurred.
28 (X'1C')	The language identifier specified does not match the language identifier in the repository at the specified virtual address.
32 (X'20')	The selected repository does not appear to be a valid message repository.
40 (X'28')	An unrecoverable error has occurred. A soft abend has been taken.
44 (X'2C')	No system spool file ID is available.
48 (X'30')	An invalid file name was specified. The language identifier was left blank or contains more than one name.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'CC' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • Rx is the same register as Ry or Ry+1 • Rx+1 is the same register as Ry • Rx or Ry is register 15 • Function code is invalid • Repository buffer address is not on a 4K-byte boundary.
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state. • The user does not have the privilege class required to issue the DIAGNOSE instruction.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the message repository data. (Key-controlled protection does not apply.)

DIAGNOSE Code X'D0' – Volume Serial Support

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'D0' to pass CP the virtual device number and the volume serial (VOLSER) for a tape device. CP will include the VOLSER in error recording records it creates for that drive.

Entry Values:

Rx
Contains the address of the tape volume serial passed to the DIAGNOSE interface. The tape volume serial should be six bytes long.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the tape volume serial. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the tape volume serial is in the host-primary address space.

Ry
Contains the virtual device number passed to the DIAGNOSE interface.

Exit Values: Return and condition codes are returned for DIAGNOSE code X'D0'.

Usage Note

The volume serial of a tape volume is recorded in the OBR and MDR records whenever DIAGNOSE code X'D0' has been successfully executed for the tape volume. The VOLSER is maintained until the tape volume is unloaded. If a tape is manually unloaded, CP clears the VOLSER field following the logging of the required OBR record to avoid the possibility of logging future OBR/MDR records with an incorrect VOLSER.

Responses

Condition Codes and Return Codes:

Condition Code	Return Code in Ry	Meaning
0	The original contents of Ry.	DIAGNOSE completed successfully.
1	1 (X'01')	Addressing exception condition.
1	3 (X'03')	Paging or storage error.
1	4 (X'04')	VOLSER fetch-protected.
1	5 (X'05')	Invalid virtual device number or device not dedicated
1	6 (X'06')	The VOLSER function does not support this device.
1	7 (X'07')	ALET-specification exception condition: For an XC virtual machine in access-register mode, Ax contains an ALET that has an unexpected bit setting.
1	8 (X'08')	ALEN-translation exception condition: For an XC virtual machine in access-register mode, Ax contains an ALET that cannot be translated.
1	9 (X'09')	Addressing-capability exception condition: For an XC virtual machine in access-register mode, Ax contains an ALET that designates an address space for which your virtual machine's access has been revoked.

DIAGNOSE Code X'D4' – Set Alternate User ID

Privilege Class: B

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'D4' is used by a master virtual machine when scheduling work on one of its worker virtual machines on behalf of an end user. The end user's user ID is considered to be the *alternate user ID*.

CP uses the alternate user ID in the following ways:

- Placed in the IPVMID field of the APPC/VM connection pending interrupt data when the worker issues and APPCVM CONNECT. (See Part 3, “The Advanced Program-to-Program Communication/VM,” on page 385 for more information.)
- Used as a spool file origin ID for spool files created by the worker. It establishes the end user's user ID as the originator of spool files created while the worker machine is processing the end user's request. Special files such as VMDUMP files are not affected as they rightly belong to the worker machine. When

the worker virtual machine is finished with the end user's request, the master virtual machine can set a new alternate user ID for the next job, or cancel alternate user ID processing for the worker.

Notes:

1. The parameter list may cross a page boundary.
2. Invoking DIAGNOSE code X'D4' does not change the user ID for existing IUCV/APPC connections.
3. If an external security manager is installed and security label checking is enabled, but no SECLABEL is supplied (for either subcode 0 or 4), then the worker's VMDBK is updated with the alternate user's default SECLABEL.
4. If security label checking is not enabled, the DD4ALTSC value supplied with a subcode 4 is ignored.

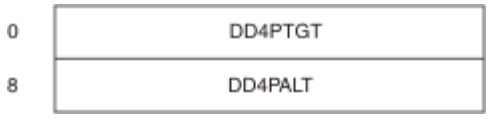
Entry Values for Subcode X'00':

Rx
X'00'

Ry
Pointer to the parameter list, DD4PARM0. DD4PARM0 COPY is provided in the HCPGPI macro library.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

DD4PARM0 DSECT
The parameter list is in this format:



DD4PTGT
is the user ID of the worker virtual machine which will run with an alternate user ID. If less than eight characters, it must be padded on the right with blanks.

DD4PALT
is the user ID of the end user requesting the work from the worker virtual machine. This user ID will appear as the originator of the spool files. If specified and less than eight characters, it must be padded on the right with blanks. If set to zero (not specified), then the alternate user ID and alternate SECLABEL functions are set off (reset to zero).

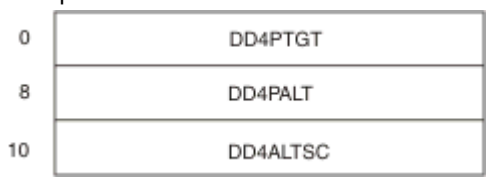
Entry Values for Subcode X'04':

Rx
X'04'

Ry
Pointer to the parameter list, DD4PARM4.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

DD4PARM4 DSECT
The parameter list is in this format:



DD4PTGT
is the user ID of the worker virtual machine which will run with an alternate user ID. If less than eight characters, it must be padded on the right with blanks.

DD4PALT
is the user ID of the end user requesting the work from the worker virtual machine. This user ID will appear as the originator of the spool files. If specified and less than eight characters, it must be padded on the right with blanks. If set to zero (not specified), then the alternate user ID and alternate SECLABEL functions are set off (reset to zero).

DD4ALTSC
is the SECLABEL of the end user requesting the work from the worker virtual machine. The worker virtual machine will acquire this SECLABEL value. The SECLABEL is a 1- to 8-character value. If specified and less than eight characters, it must be padded on the right with blanks.

Exit Values: Return codes are returned for DIAGNOSE code X'D4'.

Responses

Return Codes: Return codes are returned as follows:

Return Code in Rx	Meaning
0 (X'00')	Successful completion
4 (X'04')	Paging or storage error on the parameter list
8 (X'08')	Worker virtual machine not found
12 (X'0C')	ESM authorization denied
16 (X'10')	Alternate user ID currently set through APPC/VM

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'D4' is given incorrect data:

Problem Encountered	Cause
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state. • The issuer does not have class B privileges.
Specification exception	One of the following: • Rx and Ry are the same register • The Rx register does not contain a valid subcode.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the parameter list.

DIAGNOSE Code X'D8' – Read Spool File Blocks on System Queues

Privilege Class: D

Addressing Mode: 24-bit or 31-bit

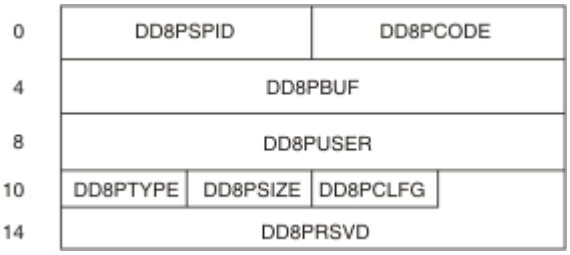
Use DIAGNOSE code X'D8' to read information from the spool file descriptor block of any file in the system. This DIAGNOSE code allows a program to request the first or next spool file block on the specified queue: reader, printer, or punch. Either files for a particular user, or any file within the system, can be specified. The descriptor block information is returned in either DIAGNOSE code X'14' compatibility (old) format (SFBLOKs), or in z/VM format (SPFBKs).

Entry Values:

Rx
Points to the parameter list, DD8PARM0. DD8PARM0 COPY is provided in the HCPGPI macro library. The parameter list must be on a doubleword boundary.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing DD8PARM0 and the output buffer for the spool file descriptor blocks. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, DD8PARM0 and the output buffer are in the host-primary address space.

DD8PARM0 DSECT
The parameter list, in this format:



DD8PSPID
is the spool file ID of the previous file when reading the *next* file. It is zero if the first file is to be read.

DD8PCODE

X'0000'
Return SFBLOK format (see Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969)

X'0004'
Return SPFBK format (see Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969)

DD8PBUF
is the address of the storage buffer in the same address space as the parameter list. The buffer must start on a doubleword boundary and must reside in the same host address space as the parameter list.

DD8PUSER
is the owner of the file if the selection is by the user. This field must be either zero or blank when requesting the first file on a queue.

DD8PTYPE
is the queue to be searched:

X'80'
Printer files

X'40'
Punch files

X'20'
Reader files

DD8PSIZE
is the size of the user buffer in double words. If it is zero, the first 20 doublewords are returned. If the address in DD8PBUF plus the length is greater than the size of the address space, the buffer is wrapped to the beginning of the address space.

DD8PCFLG
means choose system or individual user files.

X'00'
Get the next file for the user.

X'80'
Get the next file regardless of the user.

DD8PRSVD
is reserved for IBM use and must be zero.

Responses

Condition Codes: Condition codes are returned as follows:

Condition Code	Meaning
0	Successful completion; the spool file block has been copied to the user's buffer.
1	No files are available after the input file.
2	The starting file is not found.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'D8' is given incorrect data:

Problem Encountered	Cause
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state. • The issuer of the command does not have class D privileges.
Specification exception	Any of the following: • The parameter list or buffer is not doubleword-aligned. • The reserved fields are not set to zeros. • The subfunction code is not defined. • The spool file ID is specified without the corresponding user ID. • The user ID was not specified, but the request was for a particular user. • The queue specified is not valid.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the parameter list or to store into the spool file block buffer.

DIAGNOSE Code X'DC' – Control Application Monitor Record Collection

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'DC' to control the collection of monitor records for an application. Data is collected by CP from buffers declared using this Diagnose function and may be used to analyze application performance.

Entry Values: To issue DIAGNOSE code X'DC', the user's directory must contain an OPTION APPLMON statement.

Rx
Specifies the guest real address of the parameter list.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list and the product ID. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list and product ID are in the host-primary address space.

Ry
Used for responses, no entry value required

Rx and Ry can be the same register.

There are two different formats for the parameter list based on the addressing mode of the caller. For 24-bit and 31-bit addressing mode callers there is a 16 byte parameter list:

DIAG
is a 2-byte field containing the DIAGNOSE code X'00DC'.

FCN
is a 1-byte field specifying the function code of the DIAGNOSE:

X'00'
START INTERVAL RECORDING

Declare the buffer for CP monitoring. Application data is collected from this buffer by CP at each sample interval when application data monitoring is enabled by the MONITOR SAMPLE ENABLE APPLDATA command. For more information on this command, see *z/VM: CP Commands and Utilities Reference*. The application may place data in the buffer at any time. The application may declare more than one buffer from the same virtual machine. Each START request for a given buffer address and length is treated independently.

X'01'
STOP INTERVAL OR CONFIGURATION RECORDINGS

Delete the buffer from CP monitoring. Application data is no longer collected from this buffer. Each STOP request for a given buffer address and length is treated independently. If this function is issued for an interval recording buffer then if the APPLDATA domain is enabled for event monitoring, CP generates an event monitor record, indicating that collection of application data has stopped.

X'02'
GENERATE EVENT RECORD

Collect application data from the buffer immediately as an event record, if the APPLDATA domain is enabled for event recording by the MONITOR EVENT ENABLE APPLDATA command.

X'03'
START CONFIGURATION RECORDING

Collect application data from the buffer immediately as an event record, if the APPLDATA domain is enabled for event recording by the MONITOR EVENT ENABLE APPLDATA command. In addition, collect application data from the buffer as a sample configuration record whenever monitor configuration data is generated.

LEN
is a 1-byte field specifying the byte length of the parameter list.

The minimum length is X'10'. A length beyond X'10' is allowed, but only the first 16 bytes are used by CP.

PRODUCT-ID-ADDRESS
is a 4-byte guest real address in the same address space as the parameter list. This address is treated as a 31-bit address if the guest PSW is in 31- or 64-bit addressing mode. It is treated as a 24-bit

address in 24-bit addressing mode. It points to a 16-byte field that identifies the product that is generating the data. The contents of this field are fetched at the time DIAGNOSE code X'DC' START is issued, and are saved for inclusion in subsequent monitor records.

A suggested format for the product ID is *ppppppppffnvvrrmm*, where *ppppppp* is the product number or unique name, *ff* is the function of the product, *n* is the record number of the product, *vv* is the version, *rr* is the release, and *mm* is the modification level.

CP does not check or interpret this data; it is included in all monitor records generated for this buffer. This data may be useful for identification purposes.

Note: This field is not specified with function code X'01'.

////...
is a 2-byte reserved field.

BUFFER-LEN
is a 2-byte field specifying the length of the application data buffer.

The minimum value for BUFFER-LEN is 1; the maximum value is 4012.

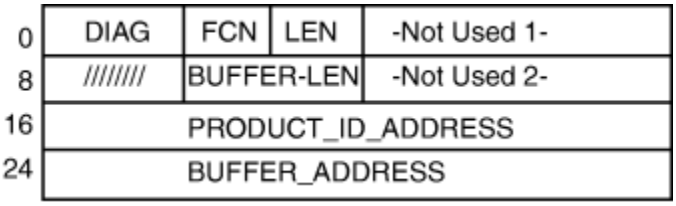
Note: The maximum figure is derived to provide minimal space in a 4KB page for the static portion (APLSDT_APHDR) of the monitor record, MRAPLSDT, the end-of-frame monitor record, MRMTREOF, and a 12-byte control area record.

For information on how to access and print the layouts or lengths of the monitor records, see *z/VM: Performance*.

BUFFER-ADDRESS
is the 4-byte guest absolute address in the host-primary address space of the application data buffer. This address is treated as a 31-bit address if the guest PSW is in 31- or 64-bit addressing mode. It is treated as a 24-bit address in 24-bit addressing mode.

A trial fetch is made of the data buffer when DIAGNOSE code X'DC' is executed; this fetch is subject to key-controlled protection, but not to fetch-protection override or storage-protection override. Subsequent to the completion of DIAGNOSE code X'DC', the buffer contents are fetched periodically to be included in monitor records; no storage protection mechanisms apply to these fetches.

In z/Architecture mode the following 32–byte parameter list is preferred for 24–, 31– and 64–bit addressing mode callers:



DIAG
is a 2-byte field containing the DIAGNOSE code X'00DC'.

FCN
is a 1-byte field specifying the function code of the DIAGNOSE:

X'80'
START INTERVAL RECORDING

Declare the buffer for CP monitoring. Application data is collected from this buffer by CP at each sample interval when application data monitoring is enabled by the MONITOR SAMPLE ENABLE APPLDATA command. For more information on this command, see *z/VM: CP Commands and Utilities Reference*. The application may place data in the buffer at any time. The application may declare more than one buffer from the same virtual machine. Each START request for a given buffer address and length is treated independently.

X'81'
STOP INTERVAL OR CONFIGURATION RECORDINGS

Delete the buffer from CP monitoring. Application data is no longer collected from this buffer. Each STOP request for a given buffer address and length is treated independently. If this function is issued for an interval recording buffer, then if the APPLDATA domain is enabled for event monitoring, CP generates an event monitor record indicating that collection of application data has stopped.

X'82'
GENERATE EVENT RECORD

Collect application data from the buffer immediately as an event record if the APPLDATA domain is enabled for event recording by the MONITOR EVENT ENABLE APPLDATA command.

X'83'
START CONFIGURATION RECORDING

Collect application data from the buffer immediately as an event record if the APPLDATA domain is enabled for event recording by the MONITOR EVENT ENABLE APPLDATA command. In addition, collect application data from the buffer as a sample configuration record whenever monitor configuration data is generated.

LEN
is a 1-byte field specifying the byte length of the parameter list. The minimum length for 64-bit addressing mode access is X'20'. A length beyond X'20' is allowed, but only the first 32 bytes are used by CP.

-Not used 1-
is a 4-byte field not used under 64-bit addressing mode.

////...
is a 2-byte reserved field.

BUFFER_LEN
is a 2-byte field specifying the length of the application data buffer. The minimum value for BUFFER_LEN is 1; the maximum value is 4012.

Note: The maximum figure is derived to provide minimal space in a 4KB page for the static portion (APLSDT_APHDR) of the monitor record, MRAPLSDT, the end-of-frame monitor record, MRMTREOF, and a 12-byte control area record.

For information on how to access and print the layouts or lengths of the monitor records, see *z/VM: Performance*.

-Not used 2-
is a 4-byte field not used under 64-bit addressing mode.

PRODUCT_ID_ADDRESS
is an 8-byte guest real address in the same address space as the parameter list. The addressing mode of the guest PSW determines whether this address is a 24-bit, 31-bit, or 64-bit address. It points to a 16-byte field that identifies the product that is generating the data. The contents of this field are fetched at the time DIAGNOSE code X'DC' START is issued, and are saved for inclusion in subsequent monitor records.

A suggested format for the product ID is *ppppppfffnvvrrmm*, where:

ppppppp
is the product number or unique name

ff
is the function of the product

n
is the record number of the product

vv
is the version

rr
is the release

mm
is the modification level

CP does not check or interpret this data; it is included in all monitor records generated for this buffer. This data may be useful for identification purposes.

Note: This field is not specified with function code X'01'.

BUFFER_ADDRESS
is the 8-byte guest absolute address in the host-primary address space of the application data buffer. This address is always treated as a 64-bit address regardless of the guest PSW addressing mode.

A trial fetch is made of the data buffer when DIAGNOSE code X'DC' is executed; this fetch is subject to key-controlled protection, but not to fetch-protection override or storage-protection override. Subsequent to the completion of DIAGNOSE code X'DC', the buffer contents are fetched periodically to be included in monitor records; no storage protection mechanisms apply to these fetches.

Exit Values: Ry contains the return code and the condition code is set in the guest's PSW. See the Responses section for descriptions.

Usage Notes

1. Recording of application data takes effect only when the application has issued DIAGNOSE code X'DC' to declare the buffer, its user has been enabled (through the CP MONITOR command) for monitoring in the APPLDATA domain, the CP monitor has been started, and at least one user is connected to the *MONITOR IUCV system service.
2. If a buffer, or any part of it, resides within a saved segment and the segment is then purged, CP performs a DIAGNOSE code X'DC' STOP operation for this buffer, thereby stopping application data collection from this buffer.
3. If an application neglects to use the STOP function for its declared buffers then monitor data collection continues until the virtual machine logs off or performs a system reset. This may cause unexpected results.

Responses

Return Codes: Upon completion of DIAGNOSE code X'DC', the following return codes are placed in Ry:

Condition Code	Return Code in Ry	Meaning
0	0 (X'00')	The request was successfully completed.
1	1 (X'01')	The DIAGNOSE code is not X'00DC'.
1	2 (X'02')	The function code is invalid (not X'00', X'01', X'02', or X'03').
1	3 (X'03')	The length of the parameter list is invalid.
1	4 (X'04')	The parameter list could not be accessed because of a CP paging or storage error trying to read one of the following: • DIAGNOSE code X'DC' parameter list • Application buffer • Product ID field.
1	5 (X'05')	The virtual machine is not authorized to issue DIAGNOSE code X'DC'.
1	6 (X'06')	The BUFFER-LEN is not within the valid range.
1	7 (X'07')	The address contained in the PRODUCT-ID-ADDRESS field is not within the address space specified in Ax (applicable for function code X'00' only).

Condition Code	Return Code in Ry	Meaning
1	8 (X'08')	The address contained in the BUFFER-ADDRESS field is not within the addressable storage of the virtual machine's primary address space (applicable for function code X'00' only).
1	9 (X'09')	A DIAGNOSE code X'DC' STOP (function code X'01') was issued for a buffer that had not been declared to CP for monitoring. That is, no DIAGNOSE code X'DC' START had been issued for the buffer whose address and length correspond to the BUFFER-ADDRESS and BUFFER-LEN of the DIAGNOSE code X'DC' STOP request.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'DC' is given incorrect data:

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See "Access Exceptions" on page 8.)	An error occurred trying to • Fetch the parameter list or product ID (except that an addressing exception condition on the product ID is reported through return code 7). • Perform a trial fetch from the application data buffer (except that an addressing exception condition on the buffer is reported through return code 8). Fetch-protection override and storage-protection override do not apply to the trial fetch.

DIAGNOSE Code X'E0' – System Trace File Interface

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'E0' provides a virtual machine access to system trace files. It allows the virtual machine to pass a buffer containing trace records to CP to be recorded in a system trace file. The trace files created by DIAGNOSE code X'E0' may be read using the CP TRACERED utility. For more information on TRACERED, see *z/VM: CP Commands and Utilities Reference*.

Entry Values:

Rx
Is not used for the WRITE function. Rx may not be specified as register 15.

Rx+1
Contains the guest real address of the block of data to be written. The contents of this block are described in "Content and Format of Trace Blocks" on page 172. The block, including its header, may be up to 32KB in length.

Ax+1
Is used only by XC virtual machines in access-register mode. Ax+1 contains the ALET for the address space containing the block of data.

Ry
Is subcode X'00000004' for the Write function. Ry may not be specified as register 15.

The other subcodes of DIAGNOSE code X'E0' are not programming interfaces, therefore, they are described in Appendix F, "Reserved DIAGNOSE Codes," on page 1015.

Exit Values: On return from the DIAGNOSE processor, a return code is set in the Ry+1 register. The return codes are right-justified in the register and padded with zeros. Only the right-most byte value is given in Responses below.

Usage Notes

1. Write requests using this DIAGNOSE code are ignored if TRSOURCE...BLOCK had not been enabled.
2. When a program writes a block of trace data, the first two bytes of each record must be the length of that record so that a read routine can determine the length of each record.
3. An addressing-capability exception condition (RC=X'44') can occur after writing has begun.

Responses

Return Codes: The following return codes are returned for subcode X'00000004', the Write function:

Return Code in Ry +1	Meaning
0 (X'00')	Successful write of trace data
4 (X'04')	Trace not enabled for this user or user ID not in BLOCK mode
12 (X'0C')	I/O error
28 (X'1C')	Invalid buffer address
32 (X'20')	Invalid buffer length
36 (X'24')	Protection exception condition
40 (X'28')	Invalid header fields
48 (X'30')	Invalid subcode
52 (X'34')	Rx+1, Ry, and Ry+1 overlap or Rx is register 15
56 (X'38')	Severe error
60 (X'3C')	ALET-specification exception condition: For an XC virtual machine in access-register mode, Ax+1 contains an ALET that has an unexpected bit setting. See the “Access Exceptions” on page 8.
64 (X'40')	ALEN-translation exception condition: For an XC virtual machine in access-register mode, Ax+1 contains an ALET that cannot be translated. See the “Access Exceptions” on page 8.
68 (X'44')	Addressing-capability exception condition: For an XC virtual machine in access-register mode, Ax+1 contains an ALET that designates an address space for which your virtual machine's access has been revoked. (See usage note “3” on page 172.) See the “Access Exceptions” on page 8

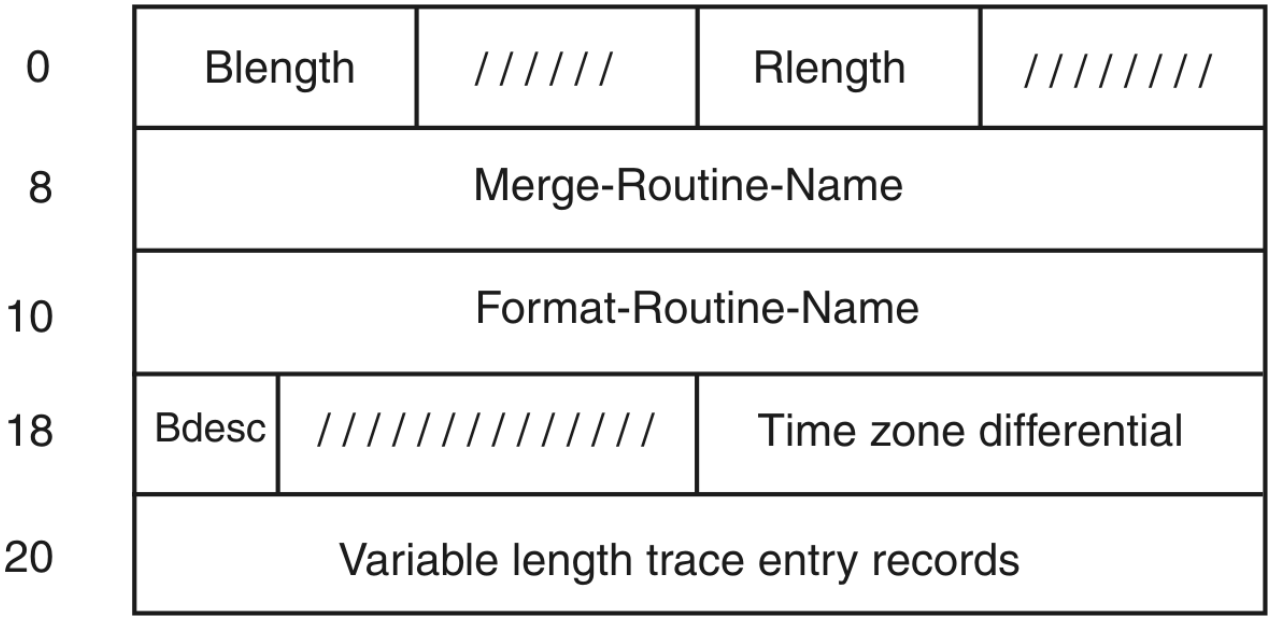
Program Exceptions: These program exceptions can occur if DIAGNOSE code X'04' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Ry is register 15.

Content and Format of Trace Blocks

Trace Blocks Containing Virtual Machine Trace Entries: The following diagram describes the format of a block of trace data collected by CP as a result of a TRSOURCE command and saved in a system data file or the block of trace data passed to CP through DIAGNOSE code X'E0' write interface. (When a

TRSOURCE...BLOCK request is made, CP inserts its own identifier into the block descriptor record and saves the time zone differential.)



Blength
Is the 2-byte field containing the number of bytes in the block. On a write request it must be less than or equal to 32KB and not less than 32 bytes (X'0020').

//////
Is a 2-byte reserved field. This field must contain zeros. It is reserved for system use.

Rlength
Is a 2-byte length of the first record, a block descriptor record used by TRACERED. This length is 28 (X'001C').

//////
Is a 2-byte reserved field. This field must contain zeros. It is reserved for system use.

Merge-routine-name
is the 8-character name of the user exit routine that can be called for each trace entry record in this block to determine its TOD clock value. This routine is used for merging entries. If no name is provided (that is, the field is blank), the records in this block cannot be merged with other trace output. On a write, this field is not checked by CP.

Note: The method used to create the merge routine user exit can be found in the *z/VM: CP Commands and Utilities Reference* under the TRACERED utility.

Format-routine-name
is the 8-character name of the user exit routine that can be called for each trace entry record in this block to provide formatted output. If no name is provided (that is, the field is blank), the records in this block cannot be formatted. On a write, this field is not checked by CP.

Note: The method used to create the format routine user exit can be found in the *z/VM: CP Commands and Utilities Reference* under the TRACERED utility.

Bdesc
This block descriptor code is reserved for system use. It is set by CP to indicate what kind of trace data the block contains.

- C'D' — Data type trace
- C'E' — Guest type trace, event mode
- C'B' — Guest type trace, block mode
- C'I' — I/O type trace

- C'L' — LAN type trace

///////
Is a 3-byte reserved field.

Time Zone Differential
is a 4-byte field reserved for system use. When the record is written, CP places the time zone differential here.

Trace Entry Records
contains trace records from the guest virtual machine. In order for these records to be processed by the CP TRACERED utility, each record must begin with a record descriptor word consisting of a 2-byte record length followed by the record.

DIAGNOSE Code X'E4' – Return Minidisk Information/Define Full-Pack Overlay

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'E4' to get information about a user's minidisk or to create a full-pack overlay.

The functions are:

Code
Function

X'00'
To get information about the device currently linked by a logged-on user.

X'01'
To get information about the device as defined in the directory when the device is attached to the system.

X'02'
To create a full-pack overlay minidisk of the volume on which a given user's specified minidisk resides.

X'03'
To create for the authorized user a full-pack overlay minidisk of the volume when given the real device and cylinder/block number.

To issue function X'00' or X'01' with a user ID other than your own, your virtual machine's directory entry must include OPTION DEVINFO or DEVMAINT. To issue function X'02' or X'03', your virtual machine's directory entry must include OPTION DEVMAINT. If External Security Manager (ESM) protection is enabled for DIAGNOSE code X'E4', then the ESM's criteria are used rather than the directory options.

Entry Values:

Rx
Contains the address of a parameter list. This must be on a doubleword boundary. The format of this parameter list is function dependent. References to the parameter list are not subject to key-controlled protection and low-address protection.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

Ry
Is not examined as input. Rx and Ry can be the same register.

Exit Values:

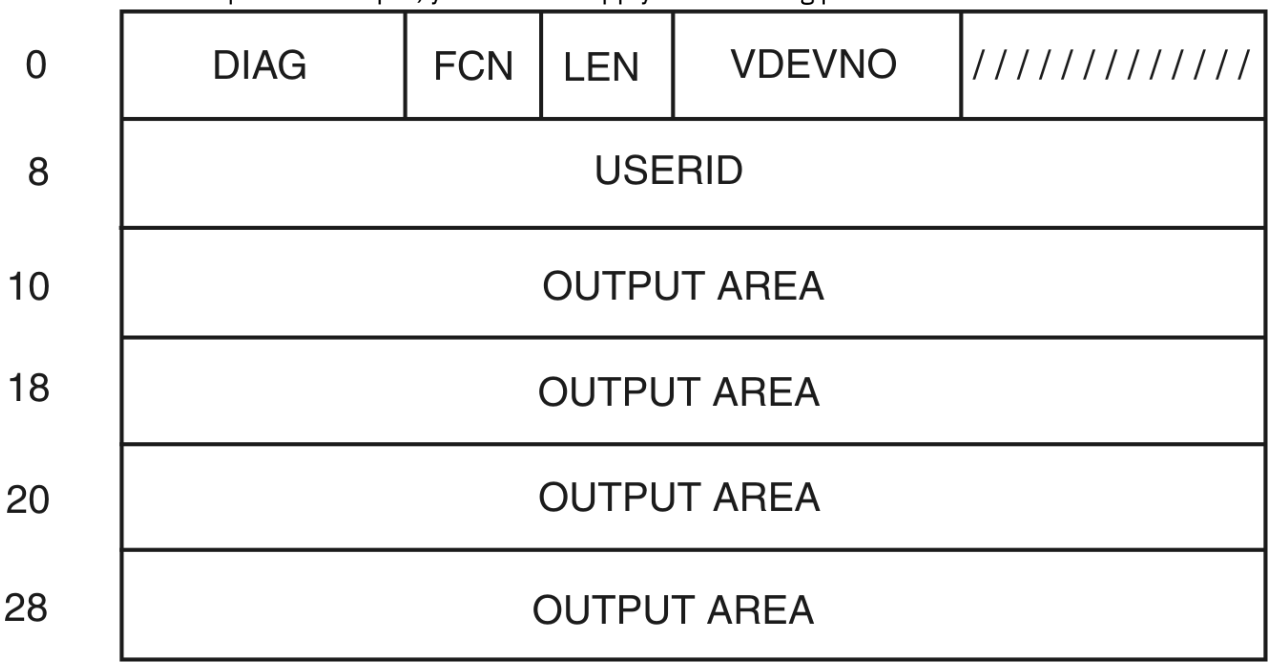
Rx
Contains the address of the parameter list supplied as input. If sufficient length LEN is specified, the parameter list output area contains the output values of the DIAGNOSE function. The format of this parameter list is function dependent. If Rx and Ry are the same register, the address of the parameter list will be overlaid by the return code.

Ry
Contains a return code. Rx and Ry can be the same register.

Function X'00' and Function X'01'

These functions of this DIAGNOSE let a user get relocation and real device information about the minidisk of another user. If ESM (External Security Manager) protection for DIAGNOSE code X'E4' is not enabled, then the DEVINFO or DEVMAINT options of the OPTION directory control statement are required for authorization. For more information on the OPTION directory control statement, see *z/VM: CP Planning and Administration*.

This DIAGNOSE also allows a user to get relocation information about the user's own minidisk. No special authorization is required. For input, you have to supply the following parameter list:



DIAG
is the halfword DIAGNOSE code. It must contain the halfword X'00E4'.

FCN
is the byte function code.

Code
Function

X'00'
To get information about the device currently linked by a logged-on user.

X'01'
To get information about the device as defined in the directory.

LEN
is the byte length, in bytes, of the parameter list. The minimum value is X'10'. The recommended value is X'30' to accommodate the output area.

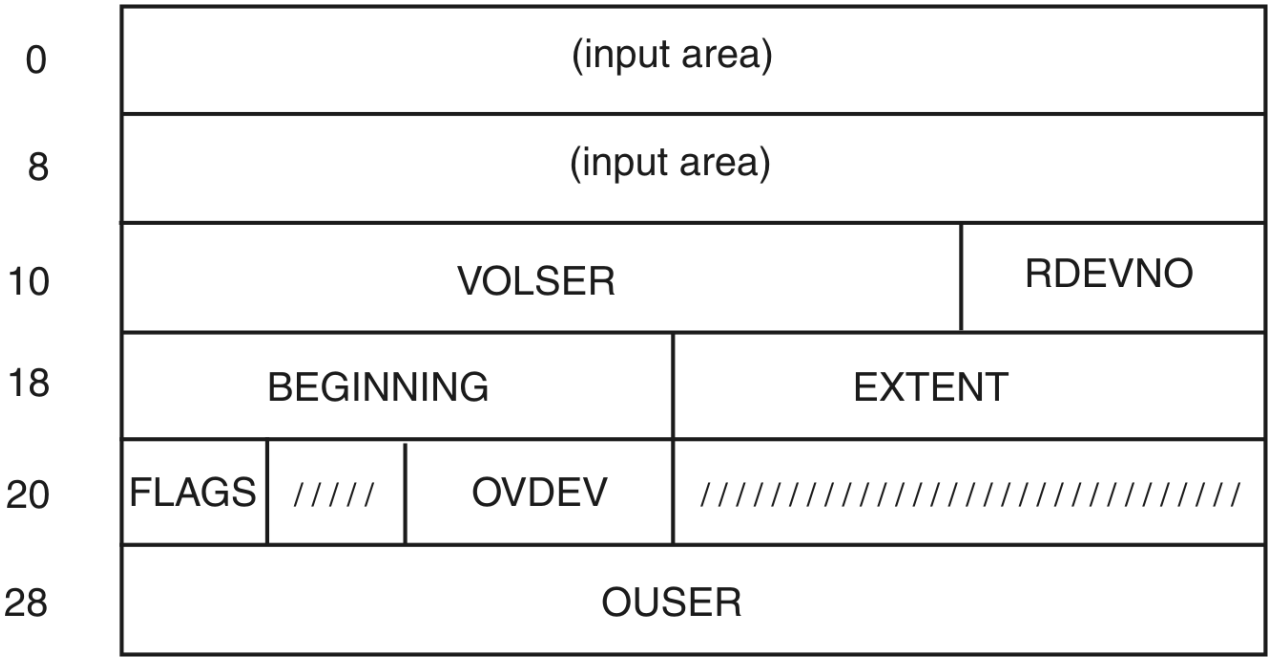
VDEVNO
is the halfword virtual device number of the minidisk for which information is to be returned.

////...
is a halfword reserved area.

USERID
is the user ID of the virtual machine that owns, or has a link to, the minidisk specified in VDEVNO. This is a doubleword of 8 characters. An asterisk (*) implies the issuer's user ID. The asterisk or user ID must be left-aligned and padded with blanks.

OUTPUT AREA
is the four doubleword parameter list output area. It is not checked or used as input.

For output, the following information is returned in the parameter list:



(input area)
is the two doublewords of the input area. This area is not used for output and is not changed from the user's input specified values.

VOLSER
is the 6-character volume serial number of the real volume that contains the minidisk. If the minidisk is a virtual disk in storage, it is not mapped to a real device, and the value (VDSK) is returned in this field.

RDEVNO
is the halfword real device number of the device containing the minidisk. If the minidisk is a virtual disk in storage, zeros are returned in this field.

BEGINNING
is the fullword relocation factor of the minidisk. This factor is specified in cylinders for CKD and ECKD devices or in blocks for FBA devices. The relocation factor is the number of the cylinder or block where the minidisk starts on the real volume.

EXTENT
is the fullword number of cylinders/blocks allocated to the minidisk. To get the number of the last cylinder or block, add the beginning cylinder/block to the total number of cylinders/blocks (extent) minus one.

FLAGS
is the device information:

Code
Meaning

X'80'
The real device associated with the specified virtual device is dedicated.

X'40'
The specified virtual device is a full-pack minidisk.

Note: The relocation factor is zero, and the extent is the number of cylinders or blocks apparently available to CP. The number of cylinders or blocks might be less than the full number of cylinders or blocks on the real volume if CP is running as a guest on another VM system in which less than a full-pack is defined.

X'20'
The specified virtual device is a non-full-pack minidisk.

X'10'
The device is a minidisk defined using the DEVNO operand of the MDISK directory control statement.

X'08'
The device is the primary device of a duplex pair.

X'04'
The device is the secondary device of a duplex pair.

X'02'
The scope of device is local.

X'01'
The scope of local is from DEFINE MDISK.

////...
is a 1-byte reserved area.

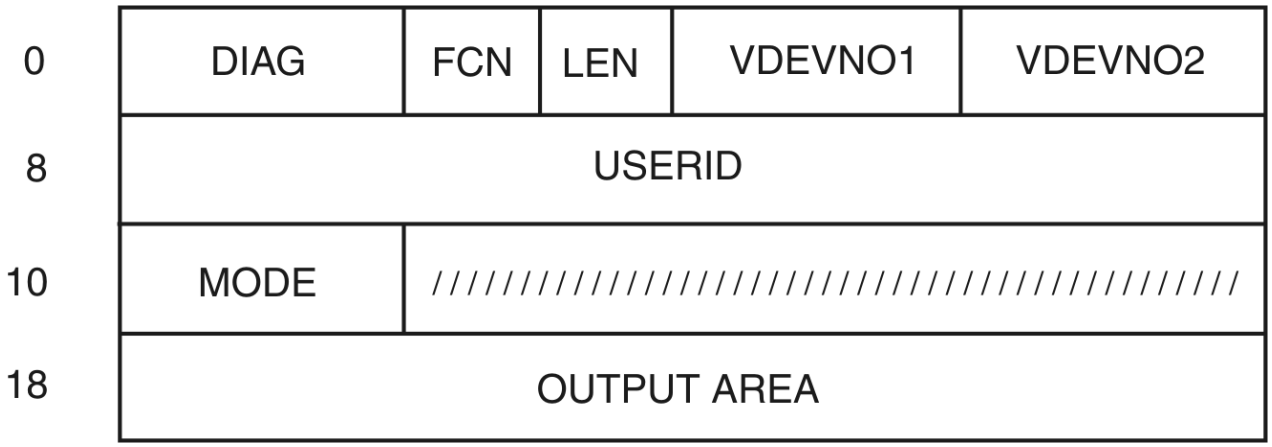
OVDEV
is the halfword virtual device number as defined for the virtual device owner.

////...
is a fullword reserved area.

OUSER
is the doubleword user ID of the virtual device owner. If the virtual device is a minidisk that was defined in terms of the source directory, OVDEV is the virtual device number that was found on the MDISK directory control statement and OUSER is the user ID of the user in whose directory entry it was found; otherwise, the OUSER-OVDEV pair is the same as was given as input.

Function X'02'

This function of DIAGNOSE code X'E4' creates for the invoker a full-pack overlay minidisk of the volume on which a given user's specified minidisk resides. If ESM (External Security Manager) protection for DIAGNOSE code X'E4' is not enabled, then the DEVMAINT option of the OPTION directory control statement is required for authorization. For more information on the OPTION directory control statement, see *z/VM: CP Planning and Administration*. For input, you have to supply the following parameter list:



DIAG
is the halfword DIAGNOSE code. It must contain the halfword X'00E4'.

FCN
is the byte function code. It must contain the byte X'02'.

LEN
is the byte length, in bytes, of the parameter list. The minimum value is X'12'. The recommended value is X'20' to accommodate the output area.

VDEVNO1
Is the halfword virtual device number of the minidisk in the other user's virtual machine configuration.

VDEVNO2
Is the halfword virtual device number in the invoker's virtual machine configuration which is to be assigned to the full-pack overlay minidisk created by this function.

USERID
is the user ID of the virtual machine that owns, or has a link to, the minidisk you specified in VDEVNO1. This is a doubleword of 8 characters. An asterisk (*) implies the issuer's user ID. The asterisk and user ID must be left-aligned and padded with blanks.

MODE
is the halfword access mode for LINK.

The valid 2-character modes are:

1. *R* specifies Read-Only access. The full-pack overlay request will not be granted if any other user has write or exclusive (read or write) access to the minidisk identified as VDEVNO1. *R* must be left-aligned and padded with a blank.
2. *W* specifies Write access. The full-pack overlay request will not be granted if any other user has read or write access to the minidisk identified as VDEVNO1. *W* must be left-aligned and padded with a blank.
3. *M* specifies Multiple access. The full-pack overlay request will be granted as a write link unless another user has an existing write, stable (read or write) or exclusive (read or write) access to the minidisk identified as VDEVNO1. *M* must be left-aligned and padded with a blank.
4. *RR* specifies Read Only access. The full-pack overlay request will always be granted as a read link unless another user has an existing exclusive (read or write) access.
5. *WR* specifies Write access desired, Read access acceptable. The full-pack overlay request will be granted as a write access unless another user holds an existing read or write access to the minidisk identified as VDEVNO1, in which case the request will be granted as a read access, unless the existing access is an exclusive (read or write) access.
6. *MR* specifies Write access desired, Read access acceptable. The full-pack overlay request will be granted as a write access unless another user has an existing write, stable or exclusive access to the minidisk identified as VDEVNO1, in which case the request will be granted as a read link, unless the existing access is an exclusive access.
7. *MW* specifies Write access is desired. The full-pack overlay request will always be granted as a write link, unless another user has an existing stable or exclusive access to the minidisk identified as VDEVNO1.
8. *SR* specifies stable Read-only access is desired. The full-pack overlay request will be granted unless another user has an existing write or exclusive access to the minidisk identified as VDEVNO1. No other write access requests for this minidisk will be granted while this access is held.
9. *SW* specifies stable Write access. The full-pack overlay request will be granted as a write access unless another user has an existing access to the minidisk identified as VDEVNO1. No other write access requests for this minidisk will be granted while this access is held.
10. *SM* specifies stable Multiple access. The full-pack overlay request will be granted as a write access unless another user has an existing write, stable or exclusive access to the minidisk identified as VDEVNO1. No other write access requests for this minidisk will be granted while this access is held.

11. *ER* specifies Exclusive Read-only access. The full-pack overlay request will be granted as read-only unless another user has an existing read or write access to the minidisk identified as VDEVNO1. No other access requests for this minidisk will be granted while this access is held.
12. *EW* specifies Exclusive Write access. The full-pack overlay request will be granted as a write access unless another user has an existing read or write access to the minidisk identified as VDEVNO1. No other access requests for this minidisk will be granted while this access is held.

////...
is a 6-byte reserved area.

OUTPUT AREA
is the doubleword parameter list output area. It is not checked or used as input.

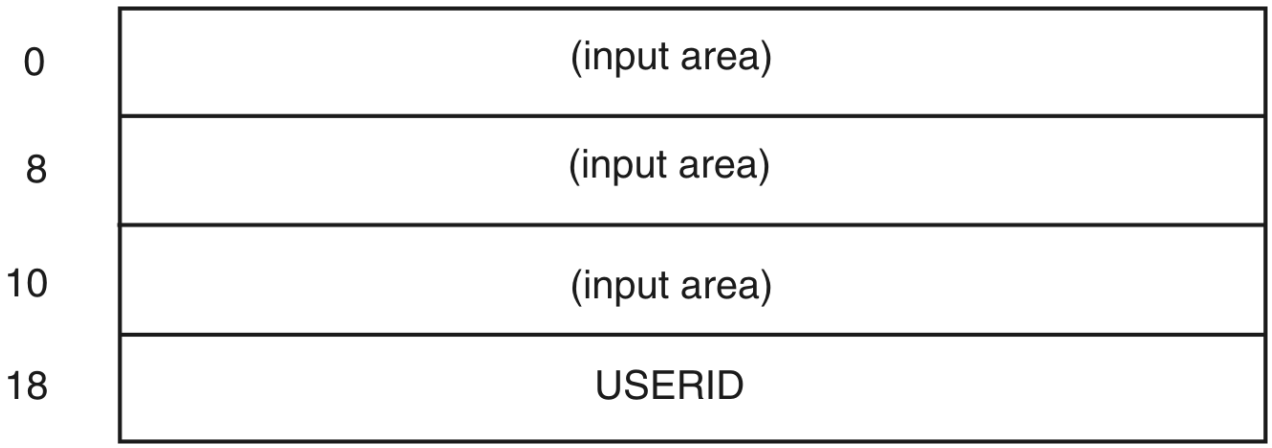
Notes:

1. If VDEVNO1 represents a LINK in the target user's directory, a maximum of 50 indirect directory iterations are attempted to find the MDISK definition.
2. If VDEVNO1 is a virtual disk in storage, this function fails with a return code of 0302. A virtual disk in storage is allocated from host storage rather than mapped to a real DASD, and therefore cannot be used to generate a full-pack overlay minidisk.
3. To use the stable and exclusive LINK access modes it is necessary for the user to have the appropriate option(s), LNKStabl or LNKExclu, specified on the OPTION directory control statement in the user's directory definition.
4. Function X'02' checks for conflicting links against all active minidisks that share any cylinders with the specified minidisk. For example, a user with a full-pack minidisk on the same volume might cause function X'02' to give return code 307.
5. Based on the state of the DASD and the target directory entry, function X'02' behaves as described by the following table:

DASD state	Directory MDISK statement	Result
Free	Defined by DEVNO	The device is attached to the system and a DEVNO full-pack overlay is defined.
Free	Defined by VOLSER	Error condition.
Attached to the system as a DEVNO-defined minidisk.	Defined by DEVNO	Define a DEVNO-defined full-pack overlay.
Attached to the system as a DEVNO-defined minidisk.	Defined by VOLSER	Error condition.
Attached to the system as a VOLSER-defined minidisk.	Defined by DEVNO	Error condition.
Attached to the system as a VOLSER-defined minidisk.	Defined by VOLSER	Define a VOLSER-defined full-pack overlay.

6. This full-pack overlay appears to its owner as an ordinary full-pack minidisk, but is intended for access to only the specified minidisk.

For output, the following information is returned in the parameter list:



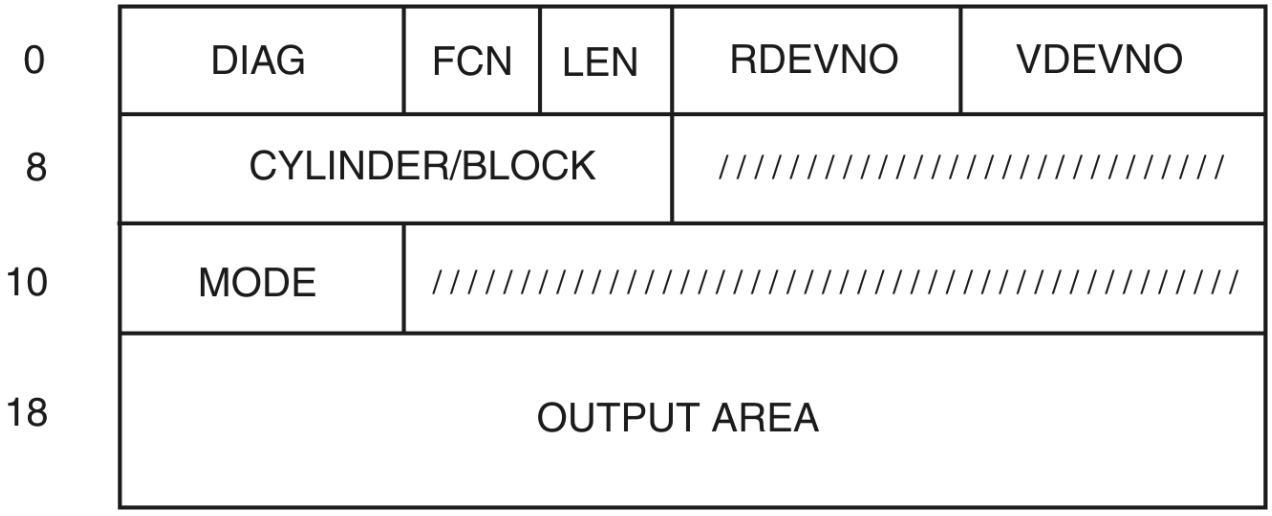
(input area)
contains the three doublewords of the input area. This area is not used for output and is not changed from the user's input specified values.

USERID
is the user ID of a virtual machine that has a link to the minidisk and is preventing the full-pack overlay from being defined. This field is filled in only if the return code is 307. This is a doubleword of 8 characters. The user ID is left-aligned and padded to the right with blanks.

Function X'03'

This function of DIAGNOSE code X'E4' creates a full-pack overlay read/write minidisk for the invoker, giving the volume real device number and the CYLINDER/BLOCK number. If ESM (External Security Manager) protection for DIAGNOSE code X'E4' is not enabled, then the DEVMAINT option of the OPTION directory control statement is required for authorization. For more information on the OPTION directory control statement, see *z/VM: CP Planning and Administration*.

For input, you have to supply the following parameter list:



DIAG
is the halfword DIAGNOSE code. It must contain the halfword X'00E4'.

FCN
is the byte function code. It must contain the byte X'03'.

LEN
is the byte parameter list length, in bytes. The minimum value is X'12'. The recommended value is X'20' to accommodate the output area.

RDEVNO
Is the halfword real device number of the volume.

VDEVNO
Is the halfword virtual device number in the invoker's virtual machine configuration which is to be assigned to the full-pack overlay minidisk created by this function.

CYLINDER/BLOCK
is one word and contains the real cylinder/block number.

////...
is a 4-byte reserved area

MODE
is the halfword access mode for LINK.

The valid 2-character modes are:

1. *R* specifies Read-Only access. The full-pack overlay request will not be granted if there is an existing write or exclusive access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. *R* must be left-aligned and padded with a blank.
2. *W* specifies Write access. The full-pack overlay request will not be granted if there is any existing read or write access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. *W* must be left-aligned and padded with a blank.
3. *M* specifies Multiple access. The full-pack overlay request will be granted as a write link unless another user has an existing write, stable or exclusive access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. *M* must be left-aligned and padded with a blank.
4. *RR* specifies Read-Only access. The full-pack overlay request will always be granted as a read link unless another user has an existing exclusive access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO.
5. *WR* specifies Write access desired, Read access acceptable. The full-pack overlay request will be granted as a write link unless another user has an existing read or write access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO, in which case the request will be granted as a read access unless the existing access is an exclusive access.
6. *MR* specifies Write access desired, Read access acceptable. The full-pack overlay request will be granted as a write link unless another user has an existing write access to the minidisk identified as CYLINDER/BLOCK on volume RDEVNO, in which case the request will be granted as a read access unless the existing access is an exclusive access.
7. *MW* specifies Write access. The full-pack overlay request will always be granted as a write link, unless there is an existing stable or exclusive access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO.
8. *SR* specifies stable Read-only access. The full-pack overlay request will not be granted if any other user has write or exclusive access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. No other write access requests that include this cylinder/block will be granted while this access is held.
9. *SW* specifies stable Write access. The full-pack overlay request will be granted as a write access unless another user has an existing access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. No other write access requests that include this cylinder/block will be granted while this access is held.
10. *SM* specifies stable Multiple access. The full-pack overlay request will be granted as a write access unless another user has an existing write, stable or exclusive access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. No other write access requests that include this cylinder/block will be granted while this access is held.
11. *ER* specifies Exclusive Read-only access. The full-pack overlay request will not be granted if any other user has an existing read or write access to the minidisk that includes CYLINDER/BLOCK on volume RDEVNO. No other access requests that include this cylinder/block will be granted while this access is held.
12. *EW* specifies Exclusive Write access. The full-pack overlay request will not be granted if any other user has an existing read or write access to the minidisk that includes CYLINDER/BLOCK on

volume RDEVNO. No other access requests that include this cylinder/block will be granted while this access is held.

////...
is a 6-byte reserved area.

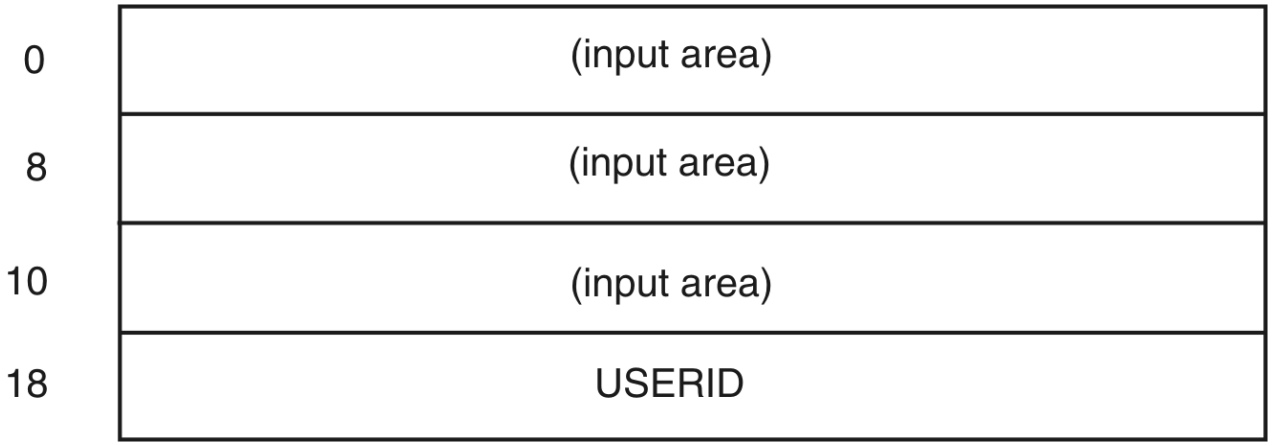
OUTPUT AREA
is a 14-byte parameter list output area. It is not checked or used as input.

Note:

1. To use the stable and exclusive LINK access modes it is necessary for the user to have the appropriate option(s), LNKStabl and/or LNKExclu, specified on the OPTION directory control statement in the user's directory definition.
2. Function X'03' checks for conflicting links against all active minidisks that include the specified cylinder. For example, a user with a full-pack minidisk on the specified volume might cause function X'03' to give return code 407.
3. This full-pack overlay appears to its owner as an ordinary full-pack minidisk, but is intended for access to only the specified cylinder or block.
4. Based on the state of the DASD, function X'03' behaves as described by the following table:

DASD state	Result
Free	The device is attached to the system and a DEVNO full-pack overlay is defined.
Attached to the system as a DEVNO-defined minidisk.	Define a DEVNO-defined full-pack overlay.
Attached to the system as a VOLSER-defined minidisk.	Define a VOLSER-defined full-pack overlay.

For output, the following information is returned in the parameter list:



(input area)
contains the three doublewords of the input area. This area is not used for output and is not changed from the user's input specified values.

USERID
is the user ID of a virtual machine that has a link to the minidisk and is preventing the full-pack overlay from being defined. This field is filled in only if the return code is 407. This is a doubleword of 8 characters. The user ID is left-aligned and padded to the right with blanks.

Usage Notes

1. For customers running RACF®/VM, please see *z/VM: RACF Security Server Security Administrator's Guide*, for procedures on how to protect the invocation of DIAGNOSE code X'E4'.

2. For functions X'00' and X'01', when the secondary device of a duplex pair is specified, the minidisk information for the corresponding primary will be returned except for the *rdev* field. This must occur because two volumes in a duplex, duplex pending, or suspended state allow only the primary volume to be fully functional. I/O operations to the secondary are limited.
3. The maximum allowed link indirections is 50.
4. You may not be authorized to issue this DIAGNOSE code if an external security manager is installed on your system. For additional information, contact your security administrator.

Responses

Condition Codes and Return Codes: Upon successful completion of DIAGNOSE code X'E4', the condition code is set to 0 and Ry is set to 0.

Upon failure of DIAGNOSE code X'E4', the condition code is set to 1 and Ry is assigned one of the following return codes:

Return Code	Meaning
0001 (X'01')	The parameter list contains the wrong DIAGNOSE code. The DIAGNOSE code must be X'00E4'.
0002 (X'02')	The parameter list contains an invalid function code. The function code must be one of X'00', X'01',X'02', or X'03'.
0003 (X'03')	The parameter list specifies a length that is too short to contain the input parameters.
0004 (X'04')	The parameter list could not be accessed because of a system error.
0005 (X'05')	The user does not have directory authorization or ESM authorization. You must have the appropriate options (such as DEVINFO, DEVMAINT, LNKStabl, LNKExclu, and so forth) specified on your OPTION directory control statement and if RACF/VM is installed, the appropriate RACF PERMITS executed on your behalf.
0100 (X'64')	For function code 00, user ID USERID is not logged on.
0101 (X'65')	For function code 00, virtual device VDEVNO is not in the specified user's current virtual I/O configuration.
0102 (X'66')	For function code 00, virtual device VDEVNO is not a minidisk or a temporary disk.
0200 (X'C8')	For function code 01, user ID USERID is not defined to the system.
0201 (X'C9')	For function code 01, virtual device VDEVNO is not in the specified user's current virtual I/O configuration.
0202 (X'CA')	For function code 01, virtual device VDEVNO is not a minidisk.
0203 (X'CB')	For function code 01, virtual device VDEVNO is on an unmounted real volume.
0204 (X'CC')	For function code 01, the directory blocks could not be accessed because of system error.
0211 (X'D3')	For function code 01, there is an excessive number of indirect links.
0300 (X'12C')	For function code 02, user ID USERID is not defined to the system.
0301 (X'12D')	For function code 02, virtual device VDEVNO1 is not in the specified user's current virtual I/O configuration.
0302 (X'12E')	For function code 02, virtual device VDEVNO1 must be a regular minidisk, not a T-disk or a virtual disk in storage.
0303 (X'12F')	For function code 02, virtual device VDEVNO1 is on an unmounted real volume.

Return Code	Meaning
0304 (X'130')	For function code 02, the directory blocks could not be accessed because of system error.
0305 (X'131')	For function code 02, a full-pack overlay already exists.
0306 (X'132')	For function code 02, virtual device VDEVNO1 is not accepted because it is a minidisk that contains CP system space (for example, paging, spooling, directory, or T-disk).
0307 (X'133')	For function code 02, minidisk has existing links to it that prevent linking with the access mode requested. The USERID returned identifies the holder of the "strongest" conflicting link (the first in the series EW, ER, SW, SR, R/W, R/O) to the minidisk identified by VDEVNO.
0308 (X'134')	For function code 02, LINK unsuccessful.
0309 (X'135')	For function code 02, access mode invalid.
0310 (X'136')	For function code 02, reserved for IBM use.
0311 (X'137')	For function code 02, there is an excessive number of indirect links.
0400 (X'190')	For function code 03, reserved for IBM use.
0401 (X'191')	For function code 03, reserved for IBM use.
0402 (X'192')	For function code 03, real device RDEVNO is not a DASD.
0403 (X'193')	For function code 03, real device RDEVNO is dedicated, off-line, or unmounted.
0404 (X'194')	Reserved.
0405 (X'195')	For function code 03, a full-pack overlay already exists.
0406 (X'196')	For function code 03, real device RDEVNO is not accepted because it lies within CP system space (for example, paging, spooling, directory, or T-disk).
0407 (X'197')	For function code 03, minidisk has existing links to it that prevent linking with the access mode requested. The USERID returned identifies the holder of the strongest conflicting link (the first in the series EW, ER, SW, SR, R/W, R/O) to the minidisk defined on the real volume RDEVNO at CYLINDER/BLOCK.
0408 (X'198')	For function code 03, LINK unsuccessful.
0409 (X'199')	For function code 03, access mode invalid.
0410 (X'19A')	For function code 03, CYLINDER/BLOCK number invalid.
0411 (X'19B')	For function code 03, real device RDEVNO is a PAV or HyperPAV alias device.

Program Exceptions: These program exceptions can occur if the DIAGNOSE X'E4' is given incorrect input data:

Problem Encountered	Cause
Specification exception	The address of the parameter list specified in Rx is not on a doubleword boundary.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch from or store into the parameter list.

DIAGNOSE Code X'EC' – Query GUEST Trace Status

Privilege Class: Any

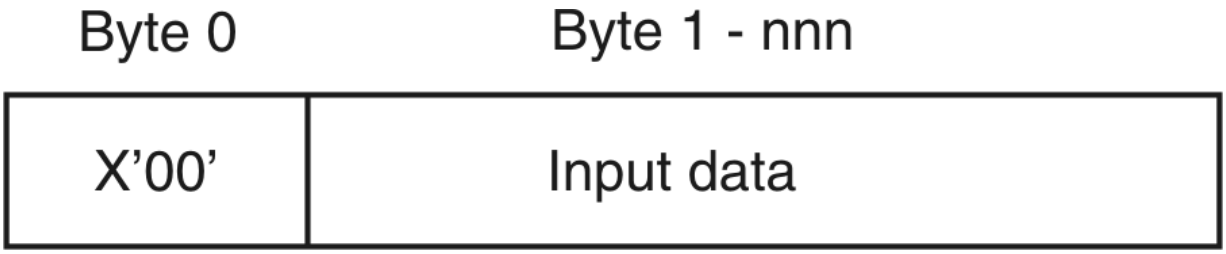
Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'EC' to determine whether data transmitted by a virtual machine through a Query GUEST Trace command is placed in a system trace file.

Entry Values:

Rx
Contains the address of the input buffer.

The input buffer should be in the following format:



Byte 0
is the subcode and must be X'00'

Byte 1
can be

- X'00' — Non-group trace entries
- X'01' — Group trace entries

Note: This value is not used by z/VM. It exists only for compatibility with VM/SP, VM/SP HPO, and VM/ESA (370 Feature).

Note: The minimum input length is 2 bytes.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the input buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the input buffer is in the host-primary address space.

Rx+1
Contains the length in bytes of the input buffer. The length must be at least 1 and no more than 255 bytes.

Ry
Contains the address of the output buffer. An output buffer must be provided.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the output buffer. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the output buffer is in the host-primary address space.

Ry+1
Contains the length in bytes of the output buffer. The length must be at least 1 and no more than 255 bytes.

Exit Values:

Ry+1
Contains the return code when an error occurred.

Information returned from DIAGNOSE code X'EC' is as follows:

Byte 0
X'00'

Byte 0 is in the following format:

x... ....
Reserved (always on)

.x.. ....
Indicates TRSOURCE has been issued for this user

..x. ....
Indicates TRSOURCE ENABLE has been issued in EVENT mode for this user

...x ....
Indicates TRSOURCE ENABLE has been issued in BLOCK mode for this user

.... xxxx
Reserved

Note: The minimum output length is 1 byte.

Responses

Condition Codes and Return Codes: Upon completion, DIAGNOSE code X'EC' sets the following condition and return codes:

Condition Code	Return Code in Ry +1	Meaning
0		Query completed without error. Rx, Rx+1, Ry, Ry+1 not changed.
1	4 (X'04')	Reserved. Not used by z/VM.
1	8 (X'08')	An error occurred. The input buffer size is invalid; it is either too small or too large.
1	12 (X'0C')	An error occurred. The output buffer size is invalid; it is either too small or too large.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'EC' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • Rx or Ry is register 15. • Rx is the same register as Ry or Ry+1 • Rx+1 is the same register as Ry • The subcode (input byte 0) is invalid.
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch input data or store output data.

DIAGNOSE Code X'F8' – Spool File Origin Information

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'F8' to have your virtual machine associate the originating node and the user ID information with a virtual output device, or retrieve the stored information for one of your virtual machine's spool files.

Subcodes:

- X'00' for the function to associate originating node and user ID with a virtual output device. This can be done only if you are authorized with the SETORIG operand on the OPTION control statement in your virtual machine's directory entry.
- X'01' for the general user function to retrieve the stored originating node and user ID information for a spool file.

Entry Values:

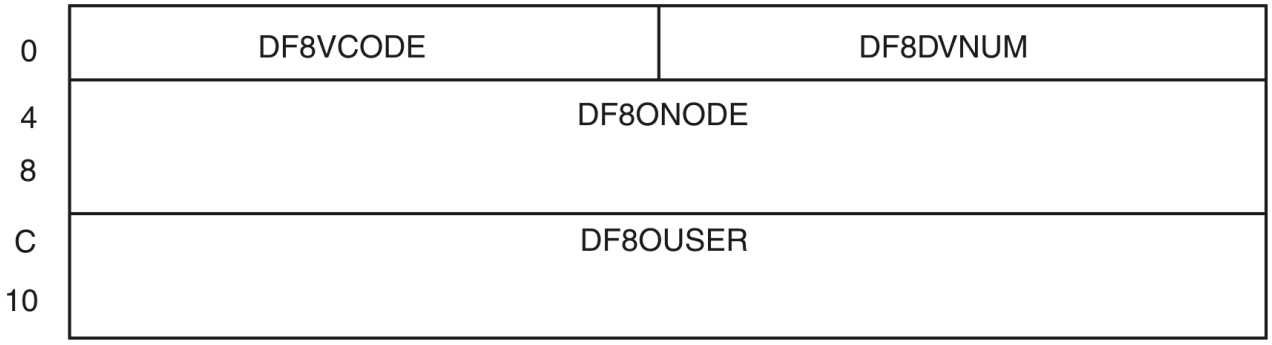
Rx
Contains the subcode.

Ry
Contains the guest real address of the doubleword-aligned parameter list, DF8PARM. DF8PARM COPY may be found in the HCPGPI macro library. The parameter list is used differently for each subcode.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the DF8PARM. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the DF8PARM is in the host-primary address space.

The DF8PARM parameter list for subcode X'00' is as follows:

DF8PARM DSECT



DF8VCODE
is reserved for IBM use. It must be binary zeros.

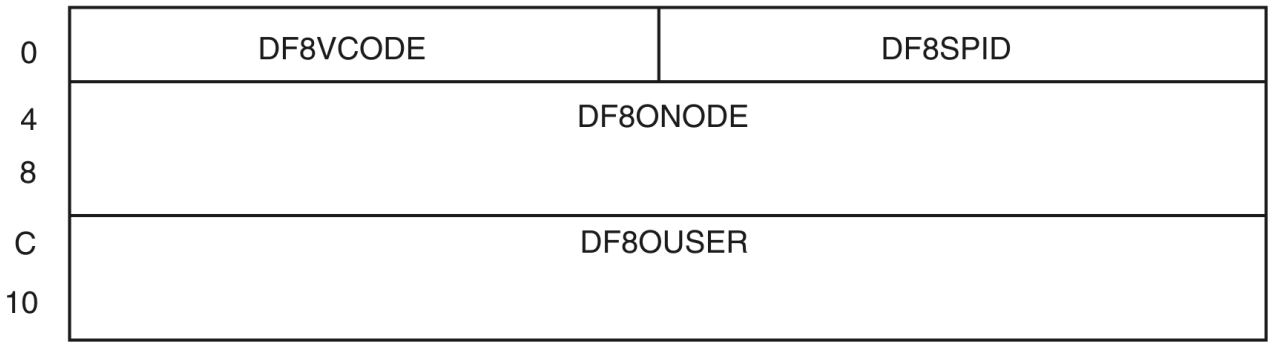
DF8DVNUM
is the output unit record device number (in hexadecimal).

DF8ONODE
is the originating node for the spool file creation (in EBCDIC). This is left-aligned and padded on the right with blanks.

DF8OUSER
is the originating user ID for the spool file created (in EBCDIC). This is left-aligned and padded on the right with blanks.

The DF8PARM parameter list for subcode X'01' is as follows:

DF8PARM DSECT



DF8VCODE
is reserved for IBM use. It must be binary zeros.

DF8SPID
is the spool file ID for which the originating node and the user ID information is retrieved (in hexadecimal).

DF8ONODE
is the originating node for the spool file (to be filled in by CP). This is left-aligned and padded on the right with blanks.

DF8OUSER
is the originating user ID for the spool file (to be filled in by CP). This is left-aligned and padded on the right with blanks.

Exit Values:

Rx
On return, Rx contains the return code for the subcode.

Usage Notes

1. If subcode X'00' is issued for a virtual output device, all files generated on that output device contain the originating node and user ID information. To change the originating node and user ID information for a virtual output device, subcode X'00' must be issued again. To clear the originating node and user ID information, subcode X'00' must be issued again for the virtual output device, specifying blanks for the originating information.
2. When subcode X'00' is issued for a virtual output device, the originating node and user ID information is stored with the next file to be opened on that device. If there is an open file on the device when subcode X'00' is issued, that file does not contain the originating node and user ID information specified on the DIAGNOSE issued after the file is opened. You must issue subcode X'00', then open the file on the specified device.
3. Subcode X'01' may not be used for an open spool file. A return code will be set indicating that the spool file was not found (RC=8).

Responses

Return Codes: Return codes for subcode X'00' are as follows:

Return Code in Rx	Meaning
0 (X'00')	Successful completion; the originating node and user ID information has been associated with the virtual output device.
8 (X'08')	Device not found
12 X'0C')	The device is not a unit record output device.

Return codes for subcode X'01':

Return Code in Rx	Meaning
0 (X'00')	Successful completion; the originating node and user ID information has been stored in the parameter list.
4 (X'04')	The originating node and user ID information were not stored for this file; DF8OUSER contains the user ID of the originator of the spool file on this node.
8 (X'08')	Spool file not found
16 (X'10')	Fatal paging I/O error

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'F8' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • The subcode is not defined. • The parameter list is not doubleword-aligned. • The DF8VCODE is invalid. • The issuer of the privileged subcode X'00' is not authorized in the CP directory.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See "Access Exceptions" on page 8.)	An error occurred trying to fetch from or store into the parameter list.

DIAGNOSE Code X'210' – Retrieve Device Information

Privilege Class: G

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'210' to request identifying information and status information about a particular virtual device. DIAGNOSE code X'210' replaces DIAGNOSE code X'24' for new applications. Your virtual machine must specify the address of the virtual device for which information is requested. DIAGNOSE code X'24' should still be used in order to determine the address of the virtual console.

Entry Values:

Rx
contains the guest real address of the Virtual/Real Device Characteristics Block (VRDCBLOK), which includes the input parameter list and an area for the output data.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Virtual/Real Device Characteristics Block (VRDCBLOK)
is the buffer containing the address of the virtual device (bytes 0 and 1) and the length of the buffer (bytes 2 and 3). The block must be on a fullword boundary and if a length of 16 bytes or more is specified, then bytes 14-15 must be zeros; otherwise, a specification exception occurs.

Exit Values:

Virtual/Real Device Characteristics Block
contains virtual device, real device, and control unit information.

Virtual/Real Device Characteristics Block

The Virtual/Real Device Characteristics Block contains virtual device, real device, and control unit information, and data from READ DEVICE CHARACTERISTICS. The symbolic names in Table 21 on page 192 describe the corresponding fields. See Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969 for a description and for bit definitions of the fields in the diagram.

You can use the VRDCBLOK DSECT, which is in the VRDCBLOK COPY file (found in the HCPGPI macro library), to map the buffer.

The VRDCBLOK DSECT fields are shown in Figure 15 on page 191 and are described in the table following the figure.

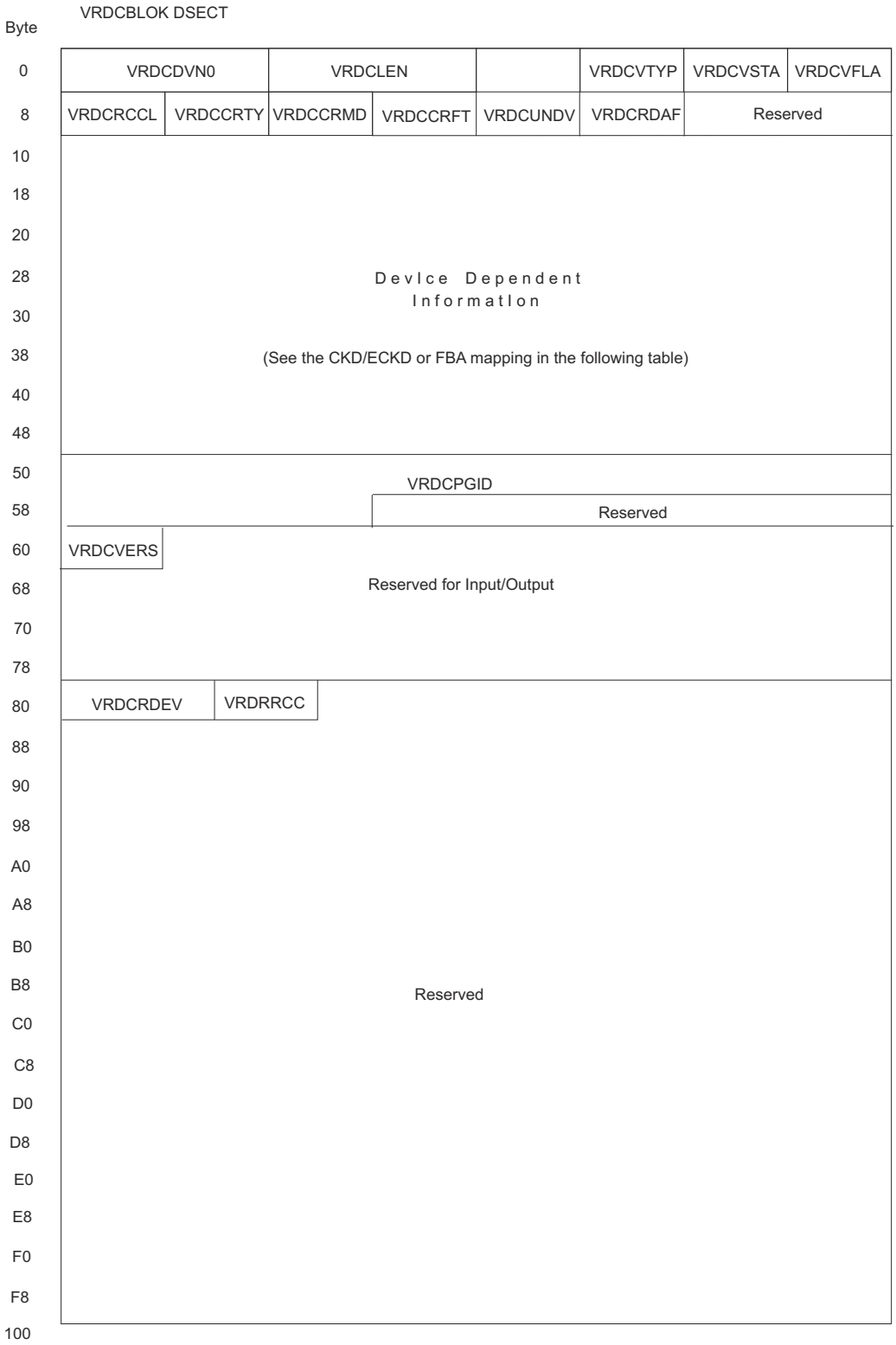


Figure 15. Fields in the VRDCBLOK DSECT

[2] For the meanings of these bytes, refer to Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969.

Table 21. Fields in the VRDCBLOK DSECT

Displace-ment	Name	Length	Description
000	VRDCDVNO	XL2	Virtual device number (input field)
002	VRDCLEN	XL2	VRDCBLOK length (input)
004	VRDCVDAT	Virtual Device Data (Ry information from DIAGNOSE code X'24')	
004	VRDCVCLA	XL1	Virtual device class.[2] Indicates the device class of the device (DASD, Tape, Unit Record...)
005	VRDCVTYP	XL1	Virtual device type.[2] Specifies the device type of the device.
006	VRDCVSTA	XL1	Virtual device status.[2]
007	VRDCVFLA	XL1	Virtual device flag.[2]
008	VRDCRCDT	Real Device Data (Ry+1 information from DIAGNOSE code X'24')	
008	VRDCRCCL	XL1	Real device type class.[2] Indicates the device class of the device (DASD, Tape, Unit Record....)
009	VRDCCRTY	XL1	Real device type.[2] Specifies the device type of the device.
00A	VRDCCRMD	XL1	Real device model. See Usage Note “3” on page 197.
00B	VRDCCRFT	XL1	Real device feature. Real device feature code[2] for nondisplay (graphic) devices or real device line length for display/graphic devices.

Table 21. Fields in the VRDCBLOK DSECT (continued)

Displace-ment	Name	Length	Description
00C	VRDCUNDV	XL1	Underlying device code for devices that emulate other device types. Valid only for CLASTAPE devices. Zero for all other device classes. Possible values for CLASTAPE devices are: **VRDCTNAT X'00'** Native non-emulation drive **VRDCT120 X'01'** 3590 128-TRK drive (B1A or B11) attached to 3590 A00/A50/A60 control unit, emulating 3490E **VRDCTVTS X'02'** 3490E virtual drive in a 3494 VTS tape library **VRDCT121 X'03'** 3590 128-TRK drive (B1A or B11) attached to 3591 control unit, emulating 3490E **VRDCT255 X'09'** 3590 256-TRK drive (E1A or E11) attached to 3590 A00/A50/A60 control unit, emulating 3590 B1A or B11 **VRDCT254 X'0A'** 3590 256-TRK drive (E1A or E11) attached to 3590 A00/A50/A60 control unit, emulating 3490E **VRDCT384 X'0B'** 3590 384-TRK drive (H11 or H1A) attached to 3590 A60 control unit, emulating 3590/B1x **VRDCT383 X'0C'** 3590 384-TRK drive (H11 or H1A) attached to 3590 A60 control unit, emulating 3490E **VRDCT512 X'10'** 3592 512–track drive (J1A or J11) attached to 3590 A60 or 3592 J70 control unit, emulating 3590 B1A or B11 **VRDCT511 X'11'** 3592 512–track drive (J1A or J11) attached to 3590 A60 control unit, emulating 3490E **VRDCT896 X'12'** 3592 896–track drive (Model E05) attached to 3590 J70 control unit, emulating 3590/B1x **VRDCTENC X'13'** 3592 896-track drive (Model E05) emulating 3590 B1x, and enabled for encryption. **VRDCTEN2 X'14'** 3592 1152-track drive (Model E06) emulating 3590 B1x, and enabled for encryption.

Table 21. Fields in the VRDCBLOK DSECT (continued)

Displace-ment	Name	Length	Description
			VRDCTEN3 X'15' 3592 2176-track drive (Model E07) emulating 3590 B1x, and enabled for encryption. **VRDCTUNK X'FF'** Unknown emulation drive
00D	VRDCRDAF	XL1	Real device additional features that are present, such as FlashCopy® capabilities. **VRDCDA4F X'01'** Diagnose A4 SCSI format supported. **VRDCEMRD X'02'** No underlying real device. Real device features are emulated. **VRDCXRCT X'04'** Time stamping is active for specified device. **VRDCFCFV X'20'** Full volume FlashCopy enabled. **VRDCFCDS X'80'** Data set-level FlashCopy enabled.
00E	VRDCRSVD	XL2	Reserved. This value must be zero on z/VM 5.3.0 and earlier.
060	VRDCVERS	XL1	Specifies version of output buffer. Should be 0 on input. On output, if this value is X'00' only the first 96B of output data are valid.
061	VRDCRSIO	XL31	Reserved for future Input/Output fields. Should be set to zero for input.
080	VRDCRDEV	XL2	Contains the device number of the underlying real device. This value is only valid when the diagnose returns with a CC of 0 and the VRDCEMRD bit is OFF. If there is a nonzero CC or the VRDCEMRD bit is ON, then VRDCRDEV will be X'0000' but invalid.
082	VRDRRCC	XL1	Contains condition codes explaining reasons why a real device may not be able to be reserved. If this field is zero, real Reserve CCWs will be issued to the underlying real device. **VRDSUPRR X'80'** Underlying device does not support reserve/release. **VRDMDWOV X'40'** Minidisk without the 'v' suffix. **VRDNFPDD X'20'** Minidisk that is not a full-pack. **VRDFPNSB X'10'** Full-pack/dedicated device and the underlying device does not have the SHARED bit on.
083	VRDCRSVE	XL125	Reserved.

Table 21. Fields in the VRDCBLOK DSECT (continued)

Displace-ment	Name	Length	Description
CKD/ECKD mapping follows:			
010	VRDCSTRT	Start of the RDC Data Bytes	
010	VRDCCUTY	XL2	Control unit type as returned by SENSE ID (bytes 1,2) or READ DEVICE CHARACTERISTICS (bytes 0,1) channel commands.
012	VRDCCUMD	1X	Control unit model as returned by SENSE ID (byte 3) or READ DEVICE CHARACTERISTICS (byte 2) channel commands
013	VRDCDVTY	XL2	Device type number as returned by SENSE ID (bytes 4,5) or READ DEVICE CHARACTERISTICS (bytes 3,4) channel commands.
015	VRDCDVMD	1X	Device model number as returned by SENSE ID (byte 6) or READ DEVICE CHARACTERISTICS (byte 5) channel commands.
016	VRDCDVFE	XL3	Device features as returned by READ DEVICE CHARACTERISTICS (bytes 6-8) channel command.
019	VRDCSDFE	1X	Storage director features as returned by READ DEVICE CHARACTERISTICS (byte 9) channel command.
01A	VRDCDVCL	1X	Device class code as returned by READ DEVICE CHARACTERISTICS (byte 10) channel command.
01B	VRDCDVCO	1X	Device type code as returned by READ DEVICE CHARACTERISTICS (byte 11) channel command.
01C	VRDDEVF1	XL28	Device Specific Field
01C	VRDCPRIM	1H	Number of primary cylinders. If greater than 65520 cylinders, this will be set to X'FFFE' and the correct value will be found only in VRDCCYLS (see displacement 04C below).
01E	VRDCTRKC	1H	Tracks per cylinder
020	VRDCSECT	1X	Number of sectors
021	VRDCTOTR	XL3	Total usable track length
024	VRDCHA	XL2	Length for HA and R0
026	VRDCMODE	1X	Track capacity calculation mode
027	VRDCMDFR	1X	Track capacity calculation modification
028	VRDCNKOV	1H	Nonkeyed record overhead
02A	VRDCKOVH	1H	Keyed area overhead
02C	VRDCALTC	1H	Address of first alternate cylinder
02E	VRDCALTR	1H	Number of alternate tracks
030	VRDCDIG	1H	Address of diagnostic cylinder
032	VRDCDIGN	1H	Number of diagnostic tracks
034	VRDCDVCY	1H	Address of first device cylinder

Table 21. Fields in the VRDCBLOK DSECT (continued)

Displace-ment	Name	Length	Description
036	VRDCDVTR	1H	Number of device support tracks
038	VRDCMDR	1X	MDR record ID (CKD, ECKD, FBA)
039	VRDCOBR	1X	OBR record ID (CKD, ECKD, FBA)
03A	VRDDEVF2	XL22	Device Specific Field
03A	VRDCCUID	1X	Control unit ID
04C	VRDCCYLS	F	Number of primary cylinders. See also VRDCPRIM (displacement 01C above).
050	VRDCPGID	XL11	VM Real Path Group ID
FBA mapping follows:			
010	VRDCFBA	XL32	FBA DASD Data
010	VRDCOPER	1X	Device operation modes
011	VRDCFBAF	1X	FBA device features
012	VRDCFBAC	1X	FBA device class
013	VRDCFBAT	1X	FBA device type
014	VRDFBAD1	1X	Device Specific Field
014	VRDCRCSZ	XL2	Physical record size
016	VRDCBKCG	XL4	Blocks per cyclical access group (track)
01A	VRDCBKAP	XL4	Blocks per access position
01E	VRDCBKMA	XL4	Blocks under movable access
022	VRDCBKFA	XL4	Blocks under fixed access
026	VRDCBKAA	XL2	Blocks in alternate area
028	VRDCBKCE	XL2	Blocks in CE area
02A	VRDCBFLG	XL2	Number of buffered log bytes
02C	VRDCATMI	XL2	Minimum access time
02E	VRDCATMA	XL2	Maximum access time
03A	VRDFBAD2	XL22	Device Specific Field

Usage Notes

1. The field VRDCVERS should be cleared before issuing DIAGNOSE X'210'. This field will be updated as output from the DIAGNOSE, such that:
   - The field will remain unchanged if the value of VRDCLEN is not greater than X'60'.
   - The field will remain unchanged if z/VM is version 5.3.0 or earlier.
   - The field will be set to X'01' if z/VM is version 5.4.0 or greater and the value of VRDCLEN is greater than X'60'. This implies that the data returned beginning at offset X'60' is valid up until VRDCLEN.
2. It is strongly recommended that the field VRDCRSIO be cleared prior to issuing the DIAGNOSE. IBM presently does not check these fields for input, but may do so in future releases.

3. The device information in bytes X'004' through X'009', and X'00B', is returned in a format consistent with DIAGNOSE code X'24'. However, the real device model field in DIAGNOSE code X'24' and the corresponding byte (X'00A') in DIAGNOSE code X'210' return different information. For a 3380 DASD, DIAGNOSE code X'24' returns the high-order 4 bits of the control unit model number in its high-order 4 bits, and the low-order 4 bits of the device model number in its low-order 4 bits in the real device model field. For a 3390 and 9345, DASD, the model number does not apply for DIAGNOSE code X'24'. Instead, byte X'00A' in DIAGNOSE code X'210' contains only device model information for all devices.
4. Refer to the hardware manuals for content descriptions of the sense ID or the READ DEVICE CHARACTERISTICS channel command results.
5. If DIAGNOSE code X'210' is issued for a device that does not support RDC, the RDC data bytes starting with Byte X'015' show the static device information and do not necessarily conform to the format given.
6. When DIAGNOSE code X'210' is issued for a minidisk, the number of primary cylinders or blocks is set to the size of the minidisk. The alternate, diagnostic, and device support cylinder or block addresses, and number of tracks, are all set to zero.
7. DIAGNOSE code X'210' returns as much information to the virtual machine as possible in the VRDCBLOK. For example, if the length is only 8 bytes, then only the virtual device information is returned.
8. If the virtual device is a virtual disk in storage, it is not mapped to a real device. However, condition code 0 is returned for successful completion, and VRDCBLOK contains information about the simulated real device.
9. When issued for a virtual console and the user is logged on to the system console, CP will return real device information indicating that the real device is an undefined line mode terminal.

Responses

Condition Codes: Upon completion of DIAGNOSE code X'210', the condition codes are:

Condition Code	Meaning
0	Normal completion.
1	CP paging error, no data returned.
2	The virtual device exists, but is not associated with a real device.
3	Invalid device address, or the virtual device does not exist.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'210' is given incorrect data:

Problem Encountered	Cause
Specification exception	• The VRDCBLOK address is not on a fullword boundary. • The VRDCBLOK length is not 8 bytes or greater. • For z/VM 5.3.0 and earlier: The VRDCRSVD field does not contain zeros.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch from or store into the VRDCBLOK.

DIAGNOSE Code X'218' – Retrieve Real CPU Identification

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'218' to request the real CPU identification. The information is stored in a buffer address supplied in the Ry register.

Entry Values:

Rx
Contains a function code:

- Function code 0

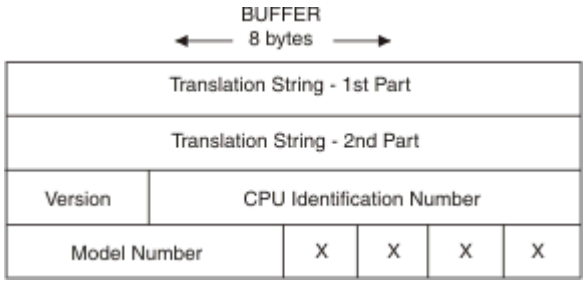
  The CPU identification is returned to an 8-byte buffer address in Ry. The data returned is the same as supplied by the STIDP instruction on the real machine.

- Function code 1

  The CPU identification is returned in 16-byte character format to the second half of a 32-byte buffer addressed by Ry. The first half of the 32-byte buffer contains a 16-byte translation string supplied by the calling program. Each byte of the translation string corresponds in sequence to the characters 0 (zero) through F.

Ry
Contains a guest real address of a buffer in the following format:

- Function code 0 in Rx: CPU identification; for format, see the description of STORE CPU ID in z/Architecture Principles of Operation, SA22-7832.
- Function code 1 in Rx:



  A 32-byte buffer; the first 16 bytes contains a translation string supplied by the calling program. The translated CPU identification is stored in character format in the second half of the buffer.

  The 8-byte binary CPU ID (STIDP instruction format) is converted to a 16-byte character string by translating each 4-bit nibble to one byte, picking the byte at the corresponding offset (0-15) in the given 16-byte translation string.

  For example, specifying the translation string '0123456789ABCDEF' results in the CPU identification being returned in character format in the second half of the 32-byte buffer.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Exit Values:

Rx
Contains a return code

Ry
Does not change.

Results are stored in the CPU identification buffer.

Usage Notes

1. In a multiprocessor configuration, this DIAGNOSE may execute on different processors in subsequent calls, resulting in different a CPU identification being returned.
2. As is the case for all programs that run in virtual machines, the execution of a program that uses this DIAGNOSE code, and the results of the DIAGNOSE code that the program observes, can be altered through the use of CP debugging commands (such as TRACE).

Responses

Return Codes: Upon completion of DIAGNOSE code X'218', the following return codes are placed in Rx:

Return Code in Rx	Meaning
0 (X'00')	Real CPU identification is successfully returned to the buffer address supplied in Ry.
4 (X'04')	DIAGNOSE is not supported on a lower level Control Program. Virtual CPU identification is returned to the buffer address supplied in Ry.

Examples

The following code shows how DIAGNOSE code X'218' can be called to get the binary CPU ID from the real processor:

```
DIAG     CSECT
*************************************************************************
* Register Equates                                                      *
*************************************************************************
R0       EQU       0
R1       EQU       1
R2       EQU       2
R3       EQU       3
R4       EQU       4
R5       EQU       5
R6       EQU       6
R7       EQU       7
R8       EQU       8
R9       EQU       9
R10      EQU       10
R11      EQU       11
R12      EQU       12
R13      EQU       13
R14      EQU       14
R15      EQU       15
*************************************************************************
*   Save registers and set addressability                               *
*************************************************************************
         STM       R14,R12,12(R13)     Save the registers
         LR        R12,R15             Point to start of program
         USING     DIAG,R12            Establish addressability
         LR        R6,R13              Save callers save area address
         LA        R13,SAVEAREA        Get my save area address
         ST        R13,8(R6)           Store my save area address
         ST        R6,4(R13)           Store caller save area address
*************************************************************************
*   Set Rx, Ry and call diagnose                                        *
*************************************************************************
         LA        R7,0                Set up Rx Register
         LA        R8,BUFFER           Ry points to buffer
         DIAG      R7,R8,X'218'        Call DIAG 218
         C         R7,=F'4'            Virtual CPU ID?
         BE        VIRTUAL
         C         R7,=F'0'            Real CPU ID?
         BE        REAL
         LINEWRT   DATA=BADRC          Bad return code from diagnose
         B         QUIT                Exit program
VIRTUAL  LINEWRT   DATA=VIRTCPU        Virtual CPU ID message
         B         UNPACK
REAL     LINEWRT   DATA=REALCPU        Real CPU ID message
```

```
*************************************************************************
*   Convert CPU ID to character format and display                      *
*************************************************************************
UNPACK    UNPK      CPUID(16),BUFFER(8) Unpack buffer
          OI        CPUID+15,X'F0'      Fix last byte
          TR        CPUID(16),TABLE     Translate to character
          LINEWRT   DATA=CPUID+1        Display CPU ID
QUIT      L         R13,SAVEAREA+4      Get old savearea address
          LM        R0,R12,20(R13)      Restore the registers
          L         R14,12(R13)         Restore return address
          BR        R14                 Exit program
SAVEAREA  DC        18F'0'              Program save area
VIRTCPU   DC        C'Virtual CPU ID returned'
REALCPU   DC        C'Real CPU ID returned'
BADRC     DC        C'Bad Return code from diagnose'
BUFFER    DC        8X'0'               8 byte buffer for diagnose
CPUID     DC        CL16' '             Character format CPU ID
          DC        C'0'                Extra character for UNPACK
          DS        0D
TABLE     DC        240X'00'            Translate table
          DC        X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'
          LTORG
          END
```

This sample code shows how DIAGNOSE code X'218' can be called to retrieve the character format CPU ID:

```
DIAG     CSECT
*************************************************************************
* Register Equates                                                      *
*************************************************************************
R0        EQU       0
R1        EQU       1
R2        EQU       2
R3        EQU       3
R4        EQU       4
R5        EQU       5
R6        EQU       6
R7        EQU       7
R8        EQU       8
R9        EQU       9
R10       EQU       10
R11       EQU       11
R12       EQU       12
R13       EQU       13
R14       EQU       14
R15       EQU       15
*************************************************************************
*   Save registers and set addressability                               *
*************************************************************************
          STM       R14,R12,12(R13)     Save the registers
          LR        R12,R15             Point to start of program
          USING     DIAG,R12            Establish addressability
          LR        R6,R13              Save callers save area address
          LA        R13,SAVEAREA        Get my save area address
          ST        R13,8(R6)           Store my save area address
          ST        R6,4(R13)           Store caller save area address
*************************************************************************
*   Set Rx, Ry and call diagnose                                        *
*************************************************************************
          LA        R7,1                Set up Rx Register
          LA        R8,BUFFER           Ry points to buffer
          DIAG      R7,R8,X'218'        Call DIAG 218
          C         R7,=F'4'            Virtual CPU ID?
          BE        VIRTUAL
          C         R7,=F'0'            Real CPU ID?
          BE        REAL
          LINEWRT   DATA=BADRC          Bad return code from diagnose
          B         QUIT                Exit program
VIRTUAL   LINEWRT   DATA=VIRTCPU        Virtual CPU ID message
          B         DISPLAY
REAL      LINEWRT   DATA=REALCPU        Real CPU ID message
DISPLAY   LINEWRT   DATA=CHARCPU        Display CPU ID
QUIT      L         R13,SAVEAREA+4      Get old savearea address
          LM        R0,R12,20(R13)      Restore the registers
          L         R14,12(R13)         Restore return address
          BR        R14                 Exit program
SAVEAREA  DC        18F'0'              Program save area
VIRTCPU   DC        C'Virtual CPU ID returned'
```

```
REALCPU   DC        C'Real CPU ID returned'
BADRC     DC        C'Bad Return code from diagnose'
```

```
BUFFER    DS        0D                  32 byte buffer for diagnose
TRSTRING  DC        C'0123456789ABCDEF' Translation string
CHARCPU   DC        CL16' '             Character CPU ID stored here
          LTORG
          END
```

DIAGNOSE Code X'238' – Time-Based Unique Identifiers

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'238' to obtain a time stamp value and clock sequence number to form unique time-based serial numbers.

The time stamp value is defined as a 60 bit number. It represents the number of 100-nanosecond intervals of time that have passed since the standard time origin. The standard time origin, the time which corresponds to a TOD clock value of zero, is January 1, 1900 0 a.m. Coordinated Universal Time (UTC).

The clock sequence number is defined as a 14 bit value. It is given a random non-zero value on the first IPL of the system or when the system is IPLed with no valid checkpoint data. It is incremented (modulo 16384) every time the system is IPLed or every time diagnose X'238' is issued and the system clock appears to have been set backwards. The range of values for the clock sequence number is from 1 to 16383. When the clock sequence number reaches 16383, the value on the subsequent IPL would be 1.

Entry Values:

Rx
Contents must be 0.

Ry
Contents must be 0.

Exit Values:

Rx
The low order 28 bits (4-31) are set to the high order 28 bits of the time stamp value. The high order 4 bits (0-3) are set to zero.

Rx+1
Set to the low order 32 bits of the time stamp value.

Ry
The clock sequence number is placed in the low order 14 bits (18-31). The high order 18 bits (0-17) are set to zero.

Usage Notes

1. No two invocations of this diagnose instruction will return the same 60 bit time stamp value unless the time on the system clock is set backwards. This is true even when the diagnose instruction is issued simultaneously by multiple virtual CPUs.
2. The clock sequence number is maintained over a system shutdown. Thus, along with the 60 bit time stamp value, the clock sequence number can be used to ensure that a unique time-based serial number is returned to the issuer of the diagnose.
3. Although the time stamp value returned by this diagnose is derived from the TOD clock, this diagnose is not intended to be used as a timing mechanism for application programs.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'238' is given incorrect data:

Problem Encountered	Cause
Specification exception	Any of the following: • Input value for either Rx or Ry was not 0. • Rx and Ry are specified as the same register. • Rx was specified as R15. • Rx+1 and Ry were specified as the same register.
Privileged-operation exception	The virtual machine is in the problem state.

DIAGNOSE Code X'248' – Copy-To-Primary Service

Privilege Class: Any (XA, ESA, and Z virtual machines only)

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'248' to copy data from an address space designated by a host access-list entry (ALE) on your virtual machine's host access list into your virtual machine's host-primary address space. An XA, ESA, or Z virtual machine must use this DIAGNOSE to access address spaces on its host access list since the host access-register translation process of the ESA/XC architecture is not available to it.

This DIAGNOSE code copies data from an ALET-specified source address space into the host-primary address space. The source location is specified by an address and an ALET contained in an even-odd general-register pair. The target location is specified by an address contained in the even register of an even-odd general-register pair, and the number of bytes to be copied is specified in the odd register of that even-odd register pair.

The ALET identifying the source address space is translated through host access-register translation as if the virtual machine was an XC virtual machine operating in the access-register mode. An ALEN-translation, ALET-specification, or addressing-capability exception may possibly be recognized on this DIAGNOSE code as a result of host access-register translation, even though these exceptions cannot normally occur for an XA or DAT-off ESA or Z virtual machine. If an ALEN-translation or addressing-capability exception occurs, zeros will be stored at location 160 and the ALET being translated will be stored at locations 168-171.

Movement of data from the source area to the target area starts at the left end (lowest-numbered address) of both areas and proceeds to the right. The storage-operand-consistency specification for the storage accesses and the handling of destructive overlap conditions is the same as is defined for the MOVE CHARACTER (MVC) instruction. For destructive-overlap purposes, the target is considered to be specified by ALET X'00000000'; thus, if a non-zero source ALET is used to access the host-primary space, destructive overlap may not be recognized and may yield unpredictable results.

Entry Values:

Rx

Contains the real address of the first byte of the target storage area. This target-area address is treated as a 24-bit or 31-bit real address as appropriate.

The target area always resides within the host-primary address space.

Address wraparound for the target addresses occurs according to the ESA rules for 24-bit address wraparound when the target addresses are treated as 24-bit addresses, and according to the ESA rules for 31-bit address wraparound when the target addresses are treated as 31-bit addresses.

Rx must designate the even register of an even-odd pair of general registers; otherwise a specification exception is recognized.

Rx+1

Bits 8-31 of the Rx+1 register contain the number of bytes to be copied from the source address space into the host-primary address space, starting at the addresses contained in the Rx and Ry registers.

Bits 0-7 of the Rx+1 register are ignored.

Ry
Contains the real address of the first byte of the source storage area. This source-area address is always treated as a 31-bit real address.

The source area resides within the host-address space designated by the ALET in register Ry+1.

Address wraparound for the source addresses occurs according to the ESA rules for 31-bit address wraparound.

Ry must designate the even register of an even-odd pair of general registers; otherwise a specification exception is recognized.

Ry+1
Contains an ALET designating the source address space. This ALET must designate a valid ALE in your virtual machine's host access list, or be an ALET of X'00000000' designating your virtual machine's host-primary address space. If the ALET does not designate a valid ALE, then host-ART-related exceptions will be recognized as in ESA/XC (ALEN-translation, ALET-specification, or addressing-capability) depending on the particular exception condition.

The Rx and Ry fields must designate different registers; otherwise a specification exception is recognized.

Exit Values: None.

Usage Note

The target addresses are treated as 24-bit or 31-bit real addresses based on the setting of the addressing-mode bit, bit 32 of the PSW.

See "How Addresses Are Processed" on page 5.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'248' is given incorrect data:

Problem Encountered	Cause
Specification exception	• The hardware support required for VM Data Spaces is not available. • Your virtual machine is not an XA, ESA, or Z virtual machine. • The Rx or Ry field does not designate an even-numbered register. • The Rx and Ry fields designate the same register.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception, including ESA/XC host-ART exceptions (See "Access Exceptions" on page 8.)	An error occurred trying to fetch the source data or store into the target area.

DIAGNOSE Code X'250' – Block I/O (Standard Blocksize)

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'250' to perform input/output operations to a direct-access storage device with consistent block sizes and supported by z/VM. DIAGNOSE code X'250' provides a virtual machine with device independent access to its virtual DASD devices either synchronously or asynchronously.

DIAGNOSE code X'250' also supports input/output operations to and from data spaces. Status of the DIAGNOSE is contained in the condition code, the Rx+1 register and the block I/O entry list.

Entry Values:

Rx
is the general register that contains the guest real address of the block I/O parameter list (HCPBIOPL). The BIOPL must be on a doubleword boundary.

The entire BIOPL will be replaced in guest storage when this function completes.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the block I/O parameter list (BIOPL). If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

Ry
In z/Architecture mode the high order word of register Ry is ignored. The low order register word contains a function code in the low order byte. The possible function codes are:

- X'00' - Initialize the environment to perform block I/O to a virtual DASD
- X'01' - Read/Write to a virtual DASD
- X'02' - Remove the block I/O environment for a virtual DASD

The remaining three bytes must contain binary zeros.

Exit Values:

Rx+1
Contains the return code indicating the result of this request. Refer to “Responses” on page 212 for a description of the possible values of this field. In z/Architecture mode the high order word of register Rx+1 is unaltered.

Initialize Block I/O to a Device

This function establishes the necessary environment to invoke subsequent read/write functions of DIAGNOSE code X'250' on a virtual device.

The function code for an initialization request is X'00', and the BIOPL is defined by one of the following formats determined by bit 0 of byte X'02'.

If bit 0 of byte X'02' is zero:

00	Device Number	FlagA | 00000000	00000000000000000000000000000000
08	00000000000000000000000000000000		00000000000000000000000000000000
10	00000000000000000000000000000000		00000000000000000000000000000000
18	Block Size		Offset
20	Start Block		End Block
28	00000000000000000000000000000000		00000000000000000000000000000000
30	00000000000000000000000000000000		00000000000000000000000000000000
38	00000000000000000000000000000000		00000000000000000000000000000000

If bit 0 of byte X'02' is one:

00	Device Number	FlagA | 00000000	00000000000000000000000000000000
08	00000000000000000000000000000000		00000000000000000000000000000000
10	00000000000000000000000000000000		00000000000000000000000000000000
18	Block Size		00000000000000000000000000000000
20	Offset		
28	Start Block		
30	End Block		
38	00000000000000000000000000000000		00000000000000000000000000000000

Device Number
is a halfword field containing the virtual device number of the DASD to which subsequent DIAGNOSE code X'250' read/write requests will be targeted. The DASD must be fully supported by z/VM. The device number must be in the range X'0000'-X'FFFF'.

Flag A
Bit 0 of byte X'02' indicates the format of the parameter list.

Bits 1-7 of byte X'02' must contain binary zeros.

Block Size
is a fullword field containing an unsigned binary number that specifies the size of the storage blocks for this request. The block size must be one of the following:

- 512
- 1024
- 2048
- 4096

Offset
is a fullword or doubleword field containing the number of sequential blocks used at the beginning of the disk by the file system to implement its structure.

This DIAGNOSE does not check the validity of this number; therefore the application can change the number if desired, but you could overlay blocks used by the file system.

Start Block
is a fullword or doubleword field set by this function to contain a signed integer representing 1 minus the offset specified on input. Start block and end block specify the range of block numbers allowable on subsequent invocations of the read/write function of DIAGNOSE code X'250' for this virtual device.

End Block
is a fullword or doubleword field set by this function to contain the number of blocks on the specified device minus the offset specified on input. Start block and end block specify the range of block numbers allowable on subsequent invocations of the read/write function of DIAGNOSE code X'250' for this virtual device.

Reserved
The rest of the fields in this block are reserved for IBM use and must contain binary zeros.

Usage Notes:

1. If the disk is in CMS format, the CMS RESERVE command may be used to allocate all available blocks of the minidisk to a unique CMS file, although this is not required.
2. If the minidisk is in CMS format, and has been reserved, an application can use the CMS function DISKID to obtain the device number, block size, and offset information. DISKID is described in *z/VM: CMS Macros and Functions Reference*.
3. The use of bit 0 of byte X'02' in the BIOPL is restricted to guests in z/Architecture mode.

Read/Write to DASD

Once block I/O initialization for a virtual DASD has completed, you can use the read/write function of DIAGNOSE code X'250' to perform I/O to that virtual device.

The function code for a read/write request is X'01', and the BIOPL is defined by one of the following formats determined by bit 0 of byte X'02'.

If bit 0 of byteX'02' is zero:

00	Device Number	FlagA | 00000000	00000000000000000000000000000000	
08	00000000000000000000000000000000		00000000000000000000000000000000	
10	00000000000000000000000000000000		00000000000000000000000000000000	
18	Key	Flags	0000000000000000	Block Count
20	ALET of the block-entry list			31 bit address of the block-entry list
28	32 bit Interruption parameter			00000000000000000000000000000000
30	00000000000000000000000000000000			00000000000000000000000000000000
38	00000000000000000000000000000000			00000000000000000000000000000000

If bit 0 of byte X'02' is one:

00	Device Number	FlagA | 00000000	00000000000000000000000000000000	
08	00000000000000000000000000000000		00000000000000000000000000000000	
10	00000000000000000000000000000000		00000000000000000000000000000000	
18	Key	Flags	0000000000000000	Block Count
20	ALET of the block-entry list			00000000000000000000000000000000
28	64 bit Interruption parameter			
30	64 bit address of the block-entry list			
38	00000000000000000000000000000000			00000000000000000000000000000000

Device Number
is a halfword field containing the virtual device number of the DASD to which the read/write operation is targeted.

Flag A
Bit 0 of byte X'02' indicates the format of the parameter list, and the block I/O entries, see “Format of a Block I/O Entry” on page 209.

Bits 1-7 of byte X'02' must contain binary zeros.

Key
Bits 0-3 of byte X'18' contain the subchannel key for fetching of output data and for storing of input data associated with this read/write request. This key is matched with a storage key during these storage references.

Bits 4-7 of this field must contain binary zeros.

Flags
Bits 0-5 of byte X'19' must contain binary zeros.

Bit 6 of byte X'19' indicates whether this is a synchronous or an asynchronous request. If the bit is zero, then the read/write operation is to be performed synchronously. The condition code and return code returned by this function will indicate the results of the operation.

If the bit is one, then the I/O may be performed asynchronously. Status of the I/O request will be presented by a block I/O external interruption. However, if the asynchronous request can be serviced entirely from minidisk cache, and bit 7 of byte X'19' is zero to indicate that minidisk cache should be interrogated, then a condition code and return code are returned by this function and no block I/O external interruption is generated.

Table 22. Summary of the Effects of Byte X'19' Bits 6 and 7 on Read/Write Processing

Bit 6	Bit 7	Effect
0	0	The request will be performed synchronously; minidisk cache may be interrogated. The condition code and return code will indicate the results of the operation.
0	1	The request will be performed synchronously; minidisk cache will not be interrogated. The condition code and return code will indicate the results of the operation. This option is ignored for write requests.
1	0	The request may be performed asynchronously; minidisk cache may be interrogated. The condition code and return code must be examined to determine if the requested was performed asynchronously; if is was, the results of the I/O request will be presented via a block I/O external interruption; otherwise, the condition code and return code will indicate the results of the operation.
1	1	The request will be performed asynchronously; minidisk cache will not be interrogated. The results of the I/O operation will be presented by a block I/O external interruption. This option is ignored for write requests.

Bit 7 of byte X'19' indicates whether minidisk cache should be bypassed for the read request. If the bit is zero, then the request will be satisfied from minidisk cache if possible. If the bit is one, then minidisk cache will not be interrogated. I/O will be performed even if the request could be satisfied from cache.

Use the minidisk cache bypass option when reading data that is not referenced frequently. This prevents infrequently-used data from filling the cache and flushing out frequently referenced data.

Note: This option is ignored for write requests.

Block Count
is a fullword field containing an unsigned binary number specifying the count of entries in the block I/O entry list. The minimum number of entries is 1, and the maximum is 256.

ALET of the block I/O entry list
If your virtual machine is an ESA/XC virtual machine executing in host-access-register mode, then this fullword field contains the access-list-entry token (ALET) designating the address space containing the block I/O entry list. If the virtual CPU of your ESA/XC virtual machine is executing in host-primary-

space mode or your virtual machine is not an ESA/XC virtual machine, then this field is ignored, and the block I/O entry list is contained in the host-primary address space.

The system performs host access-register translation for the ALETs contained within the BIOPL during the execution of DIAGNOSE code X'250'. When asynchronous I/O has been requested, the resulting ASITs are used to reference the data buffers during the ensuing asynchronous process.

31 bit address of block I/O entry list
If bit 0 of byte X'02' is zero, then bytes X'24' - X'27' is a fullword field containing the guest real address of a contiguous list of block I/O entries. Each entry in the list identifies a read or write request for the virtual device. The data within an individual entry is described in "Format of a Block I/O Entry" on page 209.

The address of the block I/O entry list must be an address on a doubleword boundary.

If bit 0 of byte X'02' is one, bytes X'24' - X'27' must contain zeros.

32 bit Interruption Parameter
If bit 0 of byte X'02' is zero, then this fullword field contains user data to be stored at guest real storage locations 128-131 in the host-primary address space upon presentation of the block I/O external interruption at the completion of an asynchronous read/write request.

64 bit Interruption Parameter
If bit 0 of byte X'02' is one, then this doubleword field contains user data to be stored at guest real storage locations 4536-4543 in the host-primary address space upon presentation of the block I/O external interruption at the completion of an asynchronous read/write request.

64 bit address of block I/O entry list
If bit 0 of byte X'02' is one, then bytes X'30' - X'37' are a doubleword field containing the guest real address of a contiguous list of block I/O entries. Each entry in the list identifies a read or write request for the virtual device. The data within an individual entry is described in "Format of a Block I/O Entry" on page 209.

The address of the block I/O entry list must be an address on a doubleword boundary.

If bit 0 of byte X'02' is zero, bytes X'30' - X'37' must contain zeros.

Reserved
The rest of the fields in this block are reserved for IBM use and must contain binary zeros.

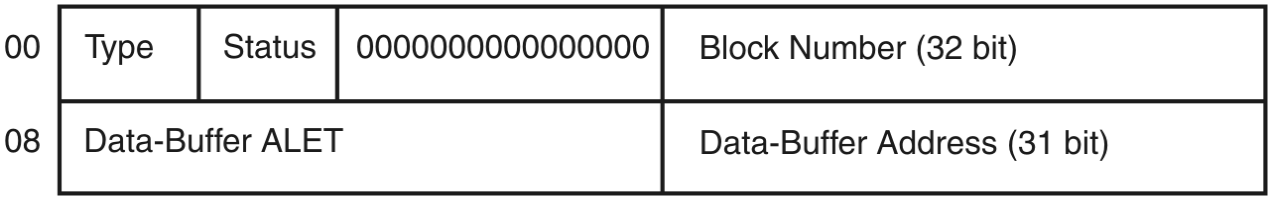
Format of a Block I/O Entry

The block I/O entry list is a contiguous list of entries, each defining a read or write request. The starting address of the list is specified in the block I/O parameter list (BIOPL). If your virtual machine is an ESA/XC virtual machine executing in host-access-register mode, then the ALET at offset X'20' of the BIOPL identifies the address space containing the block I/O entry list; otherwise, the block I/O entry list resides in the host-primary address space. For performance reasons, try not to let the block I/O entry list cross a page boundary.

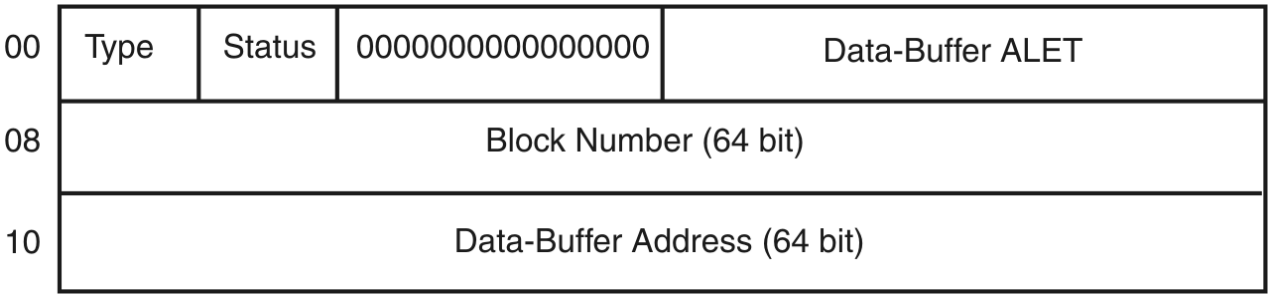
The status code field in each entry is updated in guest storage to reflect the status of the corresponding operation; it is unpredictable whether the remainder of the block I/O entry list is stored back into guest storage.

Each entry within the block I/O entry list is mapped by HCPBELBK to one the following formats.

If bit 0 of byte X'02' in the BIOPL is zero:



If bit of byte X'02' in the BIOPL is one:



Type
is the one-byte request type for this I/O operation:

X'01' Write request
X'02' Read request

Status
is a byte set by this function to contain a status code for this read or write request. See "Status Codes" on page 210 for a description of the values assigned to this field.

Block Number
is a fullword or doubleword field containing the DASD block number for this request. Block numbers are assigned sequentially to DASD records.

Data-Buffer ALET
If your virtual machine is an ESA/XC virtual machine executing in host-access-register mode, then this fullword field contains the access-list-entry token (ALET) designating the address space containing the data buffer. If the virtual CPU of your ESA/XC virtual machine is executing in host-primary-space mode or your virtual machine is not an ESA/XC virtual machine, then this field is ignored, and the data buffer is contained in the host-primary address space.

Data-Buffer Address
is a fullword or doubleword field containing the guest absolute address of the data for this request. For a *write* request this is the location in storage from which the data is written to DASD. For a *read* request this is the location in storage where the data read from DASD is placed.

Reserved
The rest of this block is reserved for IBM use and must contain binary zeros.

Status Codes

One of the following status codes is set within each block I/O entry. Since an error on a single block I/O entry does not prevent processing of other block I/O entries, you should check both the return code in Rx +1 and the status codes in each entry for the status of your request.

Table 23. Status codes for block I/O entries

Status Code in Block I/O Entry	Description
X'00'	The read or write request was successful.
X'01'	An invalid block number is specified in this block I/O entry.
X'02'	An addressing exception condition occurred because the data buffer addressed by this block I/O entry extends into storage locations not available in the designated address space.
X'03'	This block I/O entry indicates a write request; however, the target virtual DASD is read-only.
X'04'[1]	The block I/O was initiated at the device; however, the block size specified was not consistent with the block size on the virtual DASD. (This status code is valid only for CKD/ECKD devices.)

Table 23. Status codes for block I/O entries (continued)

Status Code in Block I/O Entry	Description
X'05'[1]	An irrecoverable I/O error was encountered on the virtual DASD.
X'06'	The request type specified in this block I/O entry is invalid.
X'07'[1]	A protection exception condition occurred.
X'08'[1]	An addressing-capability exception condition occurred because the address space containing the data buffer is in the revoked state (no longer accessible to your virtual machine).
X'09'[1]	An ALEN-translation exception condition occurred; the ALET in the block I/O entry for the data buffer designates an ALET that is in neither the valid nor the revoked state.
X'0A'[1]	An ALET-specification exception condition occurred; the ALET in the block I/O entry for the data buffer is invalid.
X'0B'[1]	Reserved fields in the block I/O entry are not zeros.
X'0C'[1]	This block I/O entry was not processed. Rx+1 contains the return code that indicates the condition that occurred to cause partial processing of the block I/O entry list. **Note:** This status code will only occur if a condition for which CC1 is returned on the DIAGNOSE code invocation occurs.

Note:

1

Some I/O for this request may have been performed to the device.

Usage Notes:

1. You can use the interruption parameter in the BIOPL to associate a block I/O external interruption with a particular invocation of the read/write function.
2. You can attempt to terminate a synchronous read/write operation by entering an exigent command from the virtual machine console as the first or only command on a CP command line (that is, while TERMINAL MODE is CP, or CP READ is displayed, or the command begins with the #CP prefix).
3. No I/O interruptions are returned by CP to the virtual machine for DIAGNOSE code X'250' I/O.
4. The use of bit 0 of byte X'02' in the BIOPL is restricted to guests in z/Architecture mode. This is because a 64 bit interruption parameter may be stored in the second page of the prefix area.
5. The format of the fields in the BIOPL and block I/O entries is determined by bit 0 of byte X'02' in the BIOPL and is independent of the PSW addressing mode.
6. The length of the individual block I/O entries is determined by bit 0 of byte X'02' in the BIOPL. If bit 0 of byte X'02' in the BIOPL is zero, the block I/O entries are 2 doublewords in length. If bit 0 of byte X'02' in the BIOPL is one, the block I/O entries are 3 doublewords in length.
7. The I/O for the DASD blocks specified in the block I/O entries of the BELBK may not occur in the same order that they are listed. If the application requires that the DASD blocks or I/O data buffers be updated in a particular order, then that I/O request should be implemented with separate DIAGNOSE X'250' invocations.
8. This DIAGNOSE code does not support HyperPAV alias devices.
9. The *BLOCKIO IUCV system service and Diagnose X'250' do not honor DASD reserves managed by VM's virtual reserve/release function. Therefore, do not use these I/O interfaces if a minidisk is shared by multiple guests on the same VM image where another guest is expecting to use reserve/release I/O to serialize its data access. Also, *BLOCKIO and Diagnose X'250' do not use reserve/release.

Therefore, do not use these interfaces for any DASD (CP-attached or full-pack minidisk) that is shared with other LPARs where another LPAR expects to use reserve/release to serialize its data.

Remove the Block I/O Environment

The block I/O environment established by the initialization function of DIAGNOSE code X'250' remains in effect until you explicitly clean up the environment using the remove function of DIAGNOSE code X'250' or until an I/O reset is performed for the device, for example by the SYSTEM RESET, DETACH, or RESET commands.

The remove function will clear any pending DIAGNOSE code X'250' I/O to the virtual device. An I/O reset will clear all pending I/O to that virtual device, for instance any *BLOCKIO requests in addition to DIAGNOSE code X'250' I/O.

Note: Any I/O cancelled because of a REMOVE or reset operation will be marked as having encountered a fatal I/O error.

The function code for the remove function is X'02', and the BIOPL is defined as follows:

00	Device Number	0000000000000000	00000000000000000000000000000000
08	00000000000000000000000000000000		00000000000000000000000000000000
10	00000000000000000000000000000000		00000000000000000000000000000000
18	00000000000000000000000000000000		00000000000000000000000000000000
20	00000000000000000000000000000000		00000000000000000000000000000000
28	00000000000000000000000000000000		00000000000000000000000000000000
30	00000000000000000000000000000000		00000000000000000000000000000000
38	00000000000000000000000000000000		00000000000000000000000000000000

Device Number
is the halfword field containing the virtual device number of the DASD to which block I/O is to be discontinued.

Reserved
The rest of the fields in this block are reserved for IBM use and must contain zeros.

Responses

Condition Codes and Return Codes: Upon completion of DIAGNOSE code X'250', control is returned to the invoker with a condition code set to indicate the status of both input parameter list processing and the function requested. A return code in Rx+1 further defines that status.

Table 24 on page 213 contains a general description of each of the condition codes.

Table 24. DIAGNOSE X'250' condition codes

Condition Code	Meaning
0	Function completed successfully.
1	Function partially completed. Some of the I/O completed successfully. A return code in Rx+1 indicates the condition that caused the partial completion.
2	Function failed. The environment has not been set up or removed, or no I/O has completed successfully. The return code in Rx+1 indicates the reason for the failure.

Return codes and their corresponding condition codes for the **INITIALIZE** function are listed in Table 25 on page 213.

Table 25. Condition codes and return codes for the Initialize function

Condition Code	Return Code in Rx +1	Meaning
0	0 (X'00')	Initialization for DIAGNOSE code X'250' to the specified virtual DASD is complete. The virtual device is not read-only. The starting block number and the ending block number have been stored in the BIOPL.
0	4 (X'04')	Initialization for DIAGNOSE code X'250' to the specified virtual device is complete. The virtual device is read-only. The starting block number and the ending block number have been stored in the BIOPL.
2	16 (X'10')	The virtual device is not defined.
2	20 (X'14')	The virtual device is not a supported DASD.
2	24 (X'18')	The block size is not supported.
2	28 (X'1C')	A DIAGNOSE code X'250' environment already exists for this virtual device.
2	255 (X'FF')	An irrecoverable error occurred while processing the DIAGNOSE and a soft abend may have been taken. The environment was not initialized.

Return codes and their corresponding condition codes for the **READ/WRITE** function are listed in Table 26 on page 213.

Table 26. Condition codes and return codes for the Read/Write function

Condition Code	Return Code in Rx +1	Meaning
0	0 (X'00')	A synchronous request has completed successfully, or an asynchronous request was successfully serviced from minidisk cache.
0	8 (X'08')	The asynchronous request has been initiated. The BIOPL used for the initiated request is now available for re-use.

Table 26. Condition codes and return codes for the Read/Write function (continued)

Condition Code	Return Code in Rx +1	Meaning
1	12 (X'0C')	A synchronous request was partially successful; you must check each individual block I/O entry for the status code. This return code also applies to an asynchronous request that could be serviced from minidisk cache, however, an error occurred preventing the entire request from completing successfully.
2	16 (X'10')	The virtual device is not defined.
2	28 (X'1C')	The DIAGNOSE code X'250' environment does not exist for this virtual device.
2	32 (X'20')	A CP paging error occurred while accessing the block I/O entry list.
2	36 (X'24')	The number of buffer list entries was not a positive number within the range of 1 to 256.
2	40 (X'28')	Every block I/O entry list entry is in error. You must check the status code in each block I/O entry list entry.
1	44 (X'2C')	The remove function has terminated a synchronous read/write request. Some I/O may have been performed.
1	48 (X'30')	A synchronous read/write was terminated at the user's request (for example, by an exigent command). Some I/O may have been performed.
1, 2	255 (X'FF')	An irrecoverable error occurred while processing the DIAGNOSE and a soft abend may have been taken. If this return code is returned with condition code 1, some I/O may have been performed.

Return codes and their corresponding condition codes for the **REMOVE** function are listed in Table 27 on page 214.

Table 27. Condition codes and return codes for the Remove function

Condition Code	Return Code in Rx +1	Meaning
0	0 (X'00')	Any pending DIAGNOSE code X'250' block I/O to this virtual device has been cleared. The block I/O environment has been deleted.
2	16 (X'10')	The virtual device is not defined.
2	28 (X'1C')	Block I/O to the virtual device was not previously established using the initialization function of DIAGNOSE code X'250'.
0	255 (X'FF')	An irrecoverable error occurred while processing the DIAGNOSE and a soft abend may have been taken. The environment was removed.

Program Exceptions: DIAGNOSE code X'250' may result in one of the following program exceptions:

Table 28. Program exceptions

Problem Encountered	Cause
Access exceptions (See "Access Exceptions" on page 8.)	An error occurred trying to • Fetch or store the block I/O parameter list (BIOPL) while processing an initialize function • Fetch the BIOPL while processing a read/write or remove function (it is unpredictable whether store-protection violations are reported for the read/write and remove functions) • Fetch or store the block I/O entry list while processing a read/write function.
Specification exception	• The BIOPL is not on a doubleword boundary. • A reserved field in the BIOPL does not contain binary zeros. • Bit 0 of byte X'02' in the BIOPL is one for a guest that is not in z/Architecture mode. • The block I/O entry list is not on a doubleword boundary. • An invalid function code was specified. • The high-order three bytes of the low half of the Ry register do not contain binary zeros.

Block I/O External Interruption

A block I/O external interruption is generated when an asynchronous DIAGNOSE code X'250' read/write request has completed normally, or when I/O is cancelled because of either a device reset on a device that has a DIAGNOSE code X'250' environment active, or a REMOVE request for the device. The interruption is a floating interruption condition and is presented to the first virtual CPU in the virtual configuration that is enabled for the interruption. The condition is cleared once the interruption is presented and also by a virtual subsystem reset (for example, a SYSTEM RESET or IPL command).

The subclass mask to enable for the interruption is bit 22 of control register 0.

The block I/O condition is indicated by an external-interruption code of X'2603' stored at guest real location 134-135, and a sub-interruption code of X'03' or X'07' stored at guest real location 132. If the interruption is the result of a read/write request, then an interruption parameter (as specified in the BIOPL on a DIAGNOSE code X'250' read/write request) is stored at guest real locations 128-131 for sub-interruption code X'03' or at guest real locations 4536-4543 for sub-interruption code X'07'; otherwise, binary zeros are stored at guest real locations 128-131 or 4536-4543. In addition, one of the following status codes will be stored at guest real location 133:

X'00'
All requested I/O completed successfully.

X'01'
One or more errors occurred which prevented all of the I/O requests from completing successfully. You must check each entry in the block I/O entry list to determine which requests were successfully processed.

X'02'
The updated block I/O entry list could not be stored into guest storage. The results of the I/O operations are indeterminate.

X'03'
The virtual device was reset or the environment removed. Any pending I/O to the device has been cleared. The results of the I/O operations can be determined by examining the entries in the block I/O entry list.

Notes:

1. All locations updated as a result of the external interruption are in the host-primary address space.
2. If the guest turned on bit 0 of byte X'02' in the BIOPL and is no longer in z/Architecture mode at the completion of an asynchronous request, the external interruption will not be presented.

DIAGNOSE Code X'258' – Page-Reference Services

Page-Reference Services

Privilege Class: Any

DIAGNOSE X'258' is invoked from the guest PFAULT macro and REFPAGE macro to perform page reference service functions.

Note: The preferred method of invocation for the page-reference services is the PFAULT and REFPAGE macros. For more information, refer to "PFAULT Macro -- Page-Fault Handshaking Services" on page 852 and "REFPAGE — Page Reference Services" on page 863.

The following page-reference-service functions can be invoked using this DIAGNOSE:

- Page-fault-token
- Page-fault-cancel
- Page-reference-inform, list form
- Page-reference-inform, block form.

Entry Values:

Rx
The address of a function parameter list, the format of which is determined by the function code in the parameter list. The parameter list is built by the PFAULT macro or the REFPAGE macro in the macro's work area.

Ax
Is used only in access-register mode. Ax contains the ALET for the address space containing the parameter list.

When Rx is general register 0, Ax is not examined. The ALET is assumed to be X'00000000', which indicates the host-primary address space.

Exit Values:

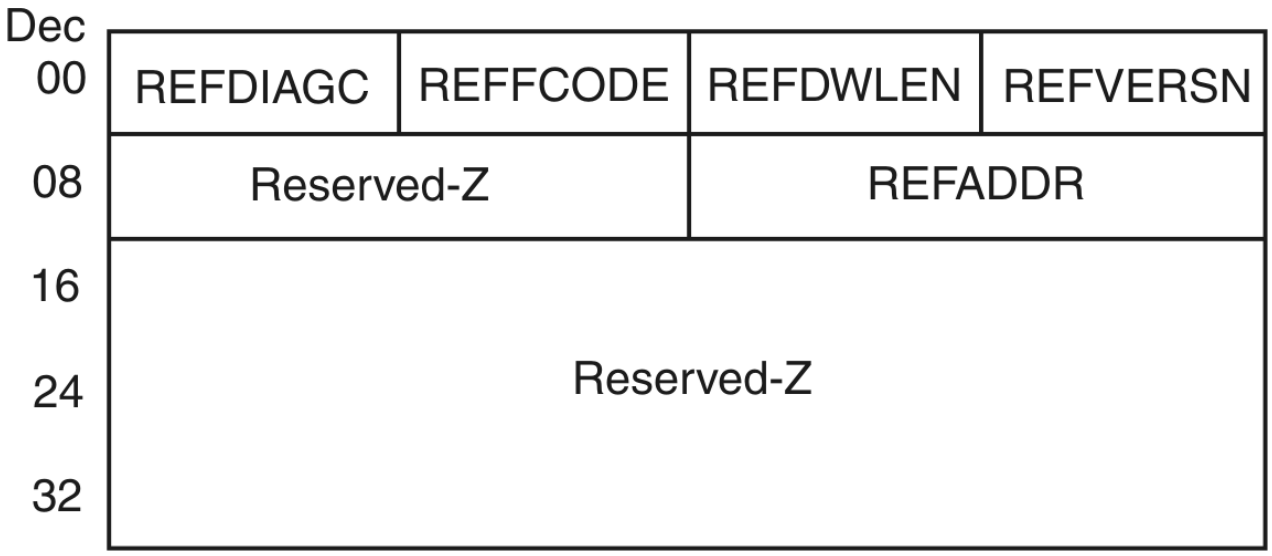
Ry
On return, register Ry contains a return code, as defined for the PFAULT or REFPAGE macros; refer to "PFAULT Macro -- Page-Fault Handshaking Services" on page 852 and "REFPAGE — Page Reference Services" on page 863.

Page-Fault-Token Function

Addressing Mode: 24-bit, 31-bit, or 64–bit

The TOKEN function of the PFAULT macro establishes the guest real storage location of the page-fault handshaking token. If the VERSION=2 parameter is specified, then it also provides the masks used to determine if a page-fault is eligible for page-fault handshaking.

Register Rx contains the address of a doubleword-aligned parameter list in the following format when VERSION=2 is **not** specified :



REFDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal code, X'0258'.

REFFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0000' indicates the TOKEN function of the PFAULT macro.

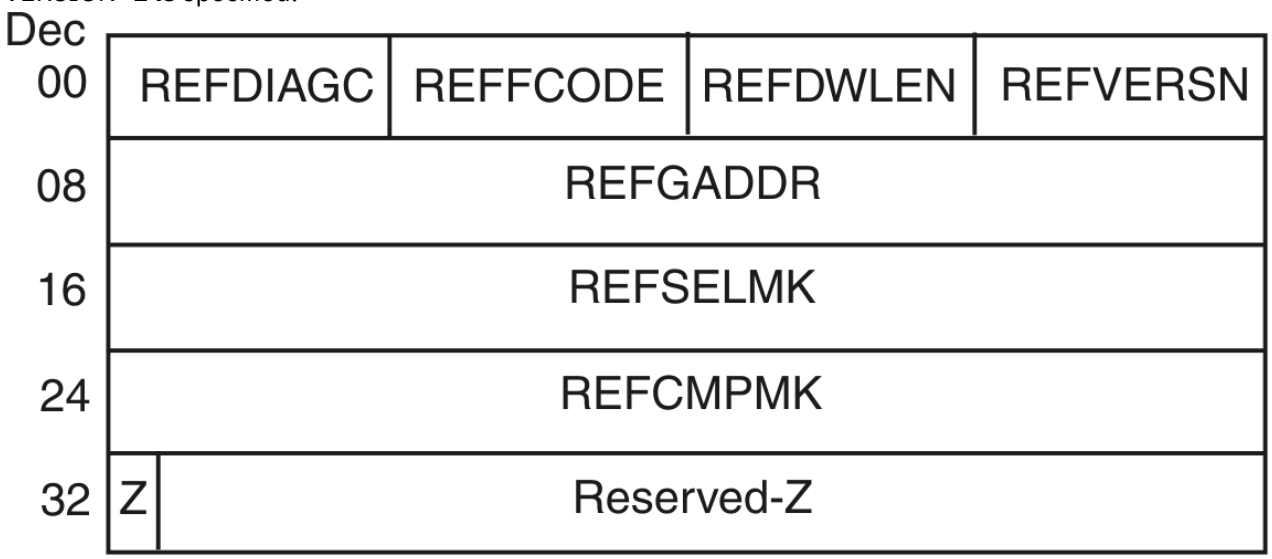
REFDWLEN
Bytes 4 and 5 of the parameter list contain the length of this parameter list in doublewords. The value must be at least X'0005.'

REFVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

Reserved-Z
Bytes 8-11 and 16-39 of the parameter list are reserved and contain binary zeros.

REFADDR
Bytes 12 through 15 of the parameter list contain the guest real address which will contain the token to be associated with AR-specified page fault handshaking. This address is always in the host-primary address space.

Register Rx contains the address of a doubleword-aligned parameter list in the following format when VERSION=2 **is** specified:



REFDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal code X'0258.'

REFFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal diagnose function code. A function code of X'0000' indicates the TOKEN function of the PFAULT macro.

REFDWLEN
Bytes 4 and 5 of the parameter list contain the length of this parameter list in doublewords. The value must be at least X'0005.'

REFVERSN
Bytes 6 and 7 of the parameter list contains a version code of X'0002.'

REFGADDR
Bytes 8 through 15 of the parameter list contain the guest real address of the token area to be associated with page-fault handshaking. REFGADDR is always in the host-primary address space. If the virtual machine is in z/Architecture mode, this is an 8 byte doubleword aligned area. If the virtual machine is not in z/Architecture mode, this is a 4 byte fullword aligned area. This 64-bit field is used as an address in the addressing mode of the virtual CPU when the diagnose is executed and the storage it points to must be accessible. At page fault time, this address is used to fetch the token to present on the initial interruption. This same token is presented on the subsequent completion interruption.

REFSELMK
Bytes 16 through 23 of the parameter list contain the 64-bit **selection mask**. At the time of a page fault, bits from the current PSW corresponding to bits set to 1 in this mask are selected for comparison against the compare mask. In z/Architecture mode, all 64 bits are used. When not in z/Architecture mode, bits 0 through 32 of this mask are used; bits 33 through 63 are ignored and assumed to be zero.

REFCMPMK
Bytes 24 through 31 of the parameter list contain the 64-bit **compare mask**. When there is a bit that is 1 in this compare mask, it must also be 1 in the selection mask, or a specification exception is presented to the virtual CPU. If the selected bits in the PSW at the time of a page fault exactly match the compare mask, then the page fault is eligible for page-fault handshaking and an initial interruption may be presented. Note that the compare mask is not used when presenting the completion interruption.

Z
This is bit 0 of this doubleword. When this bit is 0, the interface is ESA, which means the token area address points to a 4-byte field and the interruption parameters are in the format from PFAULT TOKEN being specified without including the VERSION=2 parameter. When this bit is 1, the interface is z/Architecture and the token area address points to an 8-byte field and the interruption parameters are in the format from the VERSION=2 and ARCHITECTURE=z parameters being specified with the PFAULT TOKEN macro. The setting of this bit must match the architecture of the virtual CPU executing the diagnose. This bit corresponds to the PFAULT macro ARCHITECTURE parameter.

Reserved-Z
Bytes 32 (bits 1–7 only) through 39 of the parameter list are reserved and contain binary zeros.

Comparison and Selection Mask Example

The program has determined that the PSW must be in the problem state for an initial interruption to be presented. One way of accomplishing this is for the selection mask to have a 1 in bit position 15 and to have the compare mask be the exact same mask.

Usage Notes

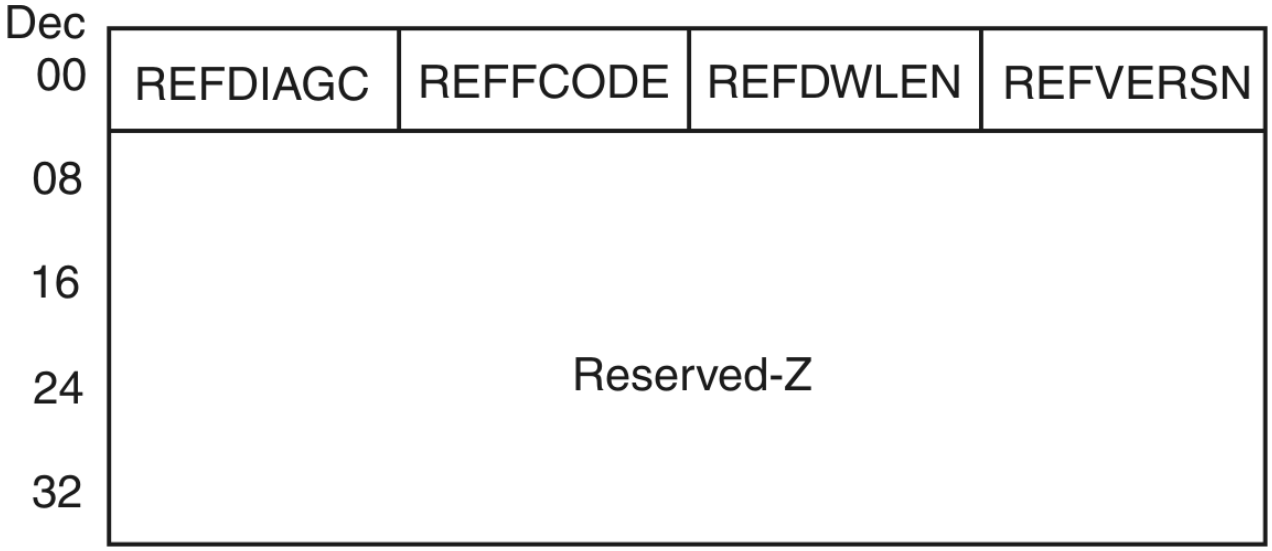
1. The address that is provided in the REFGADDR field is a real address. This address points to a token area in the host-primary address space. This address is saved and associated with the invoking virtual CPU. The length of the token area depends on whether or not the virtual CPU is in z/Architecture mode. If this address does not meet alignment restrictions, a specification exception is presented to the virtual CPU.
2. The masks in the parameter list are saved and associated with the invoking virtual CPU.

3. When VERSION=2 is in effect, this diagnose may be issued in any architecture and any addressing mode supported by CP.
4. If SIGP Set-Architecture is executed by the virtual configuration, page-fault handshaking is reset.
5. Using the PFAULT TOKEN macro is the preferred interface from assembly language to this diagnose. Refer to “PFAULT TOKEN” on page 859 for additional information regarding parameters.
6. Refer to “PFAULT Macro -- Page-Fault Handshaking Services” on page 852 for information about page-fault handshaking and the use of the token area, interruptions, and eligibility.
7. If the token area or parameter list are not accessible, then a program check is presented to the virtual CPU.
8. If any fields do not contain the value specified for them, then a specification exception is presented to the virtual CPU.
9. If the specification of the "Z" bit does not match the architecture of the invoking virtual CPU, then a specification exception is presented to the virtual CPU.

Page-Fault-Cancel Function

Addressing Mode: 24-bit, 31-bit, or 64–bit

The CANCEL function of the PFAULT macro cancels the location of the page-fault handshaking token. Register Rx contains the address of a doubleword-aligned parameter list in the following format:



REFDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal code, X'0258'.

REFFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0001' indicates the CANCEL function of the PFAULT macro.

REFDWLEN
Bytes 4 and 5 of the parameter list contain the length of this parameter list in doublewords. The value must be at least X'0005.'

REFVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001' or X'0002.' In either case, page-fault handshaking (invoked by PFAULT TOKEN) is cancelled for the invoking virtual CPU.

Reserved-Z
Bytes 8 through 39 of the parameter list are reserved and contain binary zeros.

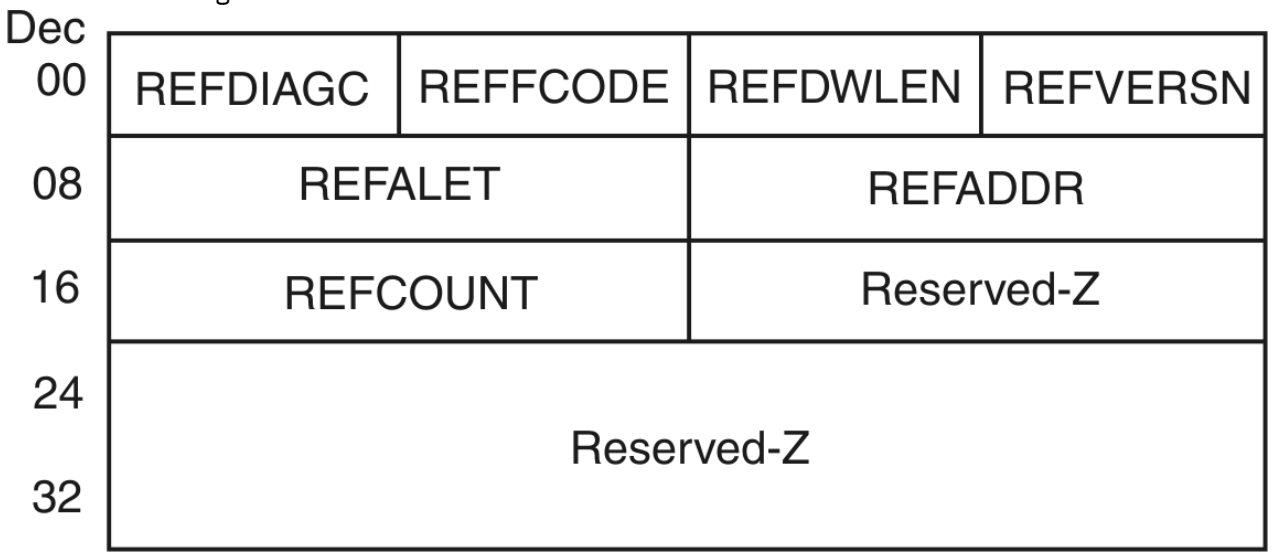
Page-Reference-Inform Function

Addressing Mode: 24-bit or 31-bit

The INFORM function of the REFPAGE macro identifies a range of pages that are about to be referenced in a specific order.

List Form

For the list-form INFORM function, register Rx contains the address of a doubleword-aligned parameter list in the following format:



REFDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal code, X'0258'.

REFFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code. A function code of X'0002' indicates the list form of the INFORM function.

REFDWLEN
Bytes 4 and 5 of the parameter list contain the length of this parameter list in doublewords. The value must be at least X'0005.'.

REFVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

REFALET
Bytes 8 through 11 of the parameter list contain the ALET designating the address space that contains the page list addressed by REFADDR. This field is ignored in primary space mode.

REFADDR
Bytes 12 through 15 of the parameter list contain the guest real storage address of the page list. The page list is the set of ALET/Page numbers.

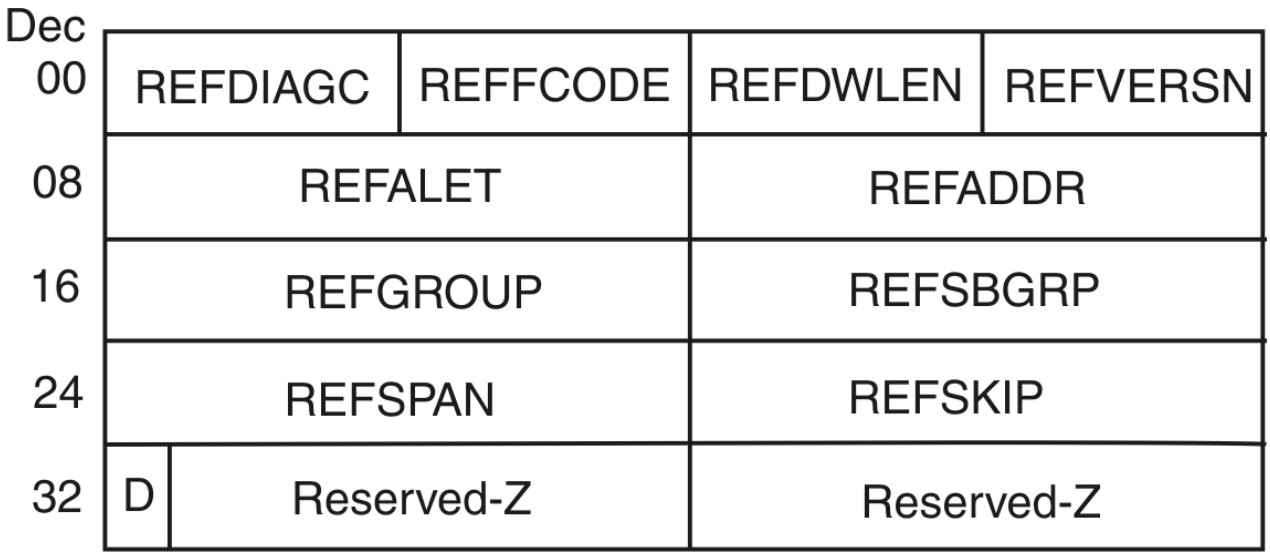
REFCOUNT
Bytes 16 through 19 of the parameter list contain the count of ALET/Page-number pairs in the list pointed to by REFADDR.

Note: The ALETs in this list are meaningful even in primary-space mode.

Reserved-Z
Bytes 20 through 39 of the parameter list are reserved and contain binary zeros.

Block Form

For the block-form INFORM function, register Rx contains the address of a doubleword-aligned parameter list in the following format:



REFDIAGC
Bytes 0 through 1 of the parameter list contain the hexadecimal code, X'0258'.

REFFCODE
Bytes 2 through 3 of the parameter list contain the hexadecimal function code. A function code of X'0003' indicates the block form of the INFORM function.

REFDWLEN
Bytes 4 through 5 of the parameter list contain the length of this parameter list in doublewords. The value must be at least X'0005.'

REFVERSN
Bytes 6 through 7 of the parameter list contain a version code of X'0001'.

REFALET
Bytes 8 through 11 of the parameter list contain the ALET of the address space that contains the page addressed by REFADDR.

REFADDR
Bytes 12 through 15 of the parameter list contain the starting page address in the page-reference-pattern. The starting address is the lowest address when the direction is ascending, or it is the highest address when the direction is descending.

REFGROUP
Bytes 16 through 19 of the parameter list contain the number of subgroups contained in the page-reference-pattern.

REFSBGRP
Bytes 20 through 23 of the parameter list contain the number of spans within a single subgroup.

REFSPAN
Bytes 24 through 27 of the parameter list contain the number of consecutive pages that should be considered as a single span of pages within the range of pages.

REFSKIP
Bytes 28 through 31 of the parameter list contain the number of pages to be skipped from the last page of a given span to the starting page of the next span.

D
Byte 32 of the parameter list contains the direction indicator. This byte should contain a value of 1 for ascending or -1 for descending, but the actual value is not validated. Bit 0 is checked to determine if it is ascending (0) or descending (1).

Reserved-Z
Bytes 33 through 39 of the parameter list are reserved and contain binary zeros.

Program Exceptions:

The program exceptions for DIAGNOSE X'258' are the same as those documented for the PFAULT and REFPAGE macros; refer to “PFAULT Macro -- Page-Fault Handshaking Services” on page 852 and “REFPAGE — Page Reference Services” on page 863.

DIAGNOSE Code X'260' – Access Certain Virtual Machine Information

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'260' to access certain virtual machine information.

Entry Values:

Rx
Depends on the subcode specified in Ry (see description below).

Rx+1
Depends on the subcode specified in Ry.

Ry
In z/Architecture mode, the high-order word of register Ry is ignored. The 4 bytes of the low-order register word contain the function subcode.

The function subcodes are as follows:

Subcode X'00000000'

Return the highest addressable byte of virtual storage in the host-primary space, including named saved systems and saved segments. This subcode is valid only for 24-bit or 31-bit addressing mode.

Entry Values:

Ry
Must be X'00000000'.

Exit Values:

Rx
Equal to the highest addressable byte of virtual storage in the host-primary space, including named saved systems and saved segments. For example, for a 2047 MB address space, Rx would be X'7FEFFFFF'. For an 8 MB address space with the highest addressable saved segment loaded in MB X'19', Rx would be X'019FFFFF'.

Usage Note: XC virtual machines in access register mode can execute DIAGNOSE code X'260' Subcode X'00000000', but only the host-primary address space is implied.

Subcode X'00000004'

Return the BYUSER ID for the user issuing this diagnose. This subcode is valid only for 24-bit or 31-bit addressing mode.

Entry Values:

Rx
Contains the address of a doubleword that may contain the BYUSER ID (left-justified, followed by spaces). The output buffer is only used if there is a BYUSER (return code 0).

Ax
Used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the output doubleword. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the output doubleword is in the host-primary address space.

Ry
Must be X'00000004'. Ry cannot be the same as Rx.

Exit Values:

Ry+1
Contains a return code.

Return Code in Ry +1	Meaning
0 (X'00')	BYUSER ID was successfully returned in the specified output buffer.
4 (X'04')	BYUSER ID was not returned. There is no BYUSER ID for this user ID.

Usage Notes:

1. Rx and Ry cannot be the same register.
2. Ry must be an even-numbered register.
3. A BYUSER ID is only defined for a virtual machine that is logged on with the BY option of the LOGON command.
4. When a virtual machine is disconnected, its BYUSER ID remains unchanged.
5. A virtual machine's BYUSER ID is updated on a successful reconnect.

Subcode X'00000008'

Return the lines per page (LPP) value for the virtual printer or virtual console. This subcode is valid only for 24-bit or 31-bit addressing mode.

Entry Values:

Rx
Contains the virtual printer or virtual console device address.

Ry
Must be X'00000008'.

Exit Values:

Rx
Contains zero if the lines-per-page (LPP) value for the virtual printer or virtual console has been set to OFF either by use of the SPOOL LPP option or through the global system setting. Rx contains the binary value of the virtual LPP if it has been specified for the device. The binary value has a range of (decimal) 30 to 255.

Ry
Contains a return code.

Return Code in Ry	Meaning
0 (X'00')	A valid LPP value was returned (including zero if LPP is OFF).
4 (X'04')	Rx is not a valid virtual printer address

Subcode X'0000000C'

Return the highest addressable byte of virtual storage in the host-primary address space, including named saved systems and saved segments. This subcode is valid only for a z/Architecture virtual machine.

Entry Values:

Ry
Must be X'0000000C' (the high-order word is ignored).

Exit Values:

Rx
Equal to the highest addressable byte of virtual storage in the first defined storage extent of the host-primary space, including named saved systems and saved segments. For example, for an 8.5G address space, Rx would be X'000000021FFFFFFF'.

Ry
Equal to the highest addressable byte of virtual storage in the host-primary address space, including named saved systems and saved segments. For example, for a guest with DEFINE STORAGE CONFIG 0.1G 8G.1G in effect, Rx would be X'000000003FFFFFFF' and Ry would be X'000000023FFFFFFF'.

Usage Notes:

1. The CONFIGURATION option on the DEFINE STORAGE command allows the definition of multiple noncontiguous storage extents. If the CONFIGURATION, RESERVED, and STANDBY options were not used on the DEFINE STORAGE command, the defined storage is one extent, and Rx and Ry will be identical.
2. If the RESERVED or STANDBY option was used on the DEFINE STORAGE command to configure reserved or standby storage for a guest, the values returned in Rx and Ry will be the current values, but these values can change dynamically depending on the options specified and any dynamic storage reconfiguration (DSR) changes initiated by the guest.

Subcode X'00000010'

Return information about a guest's storage configuration.

Entry Values:

Rx
Guest logical address of storage configuration output area.

Rx+1
Length of storage configuration output area.

Ry
Must be X'00000010'.

Exit Values:

Rx
Unchanged.

Rx+1
Unchanged.

Ry
Number of storage configuration extents.

Usage Note: If the CONFIGURATION option was not used on the DEFINE STORAGE command, the defined storage is one extent, and the value of Ry will be 1.

Condition Code	Meaning
0	All extents returned in output area
1	More extents exist than will fit in the output area

Output Area: The output area consists of one or more pairs of 64-bit values giving the start (first byte) and end (last byte) addresses of a guest storage extent. The area must be aligned on a quadword boundary.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'260' is given incorrect data:

Problem Encountered	Cause
Specification exception	Ry does not contain a valid function subcode or does not contain a function subcode that is allowed with the virtual machine's current addressing mode. For subcode X'04': • Rx and Ry are the same register. • Ry is not an even-numbered register. • The output buffer is not on a doubleword boundary. For subcode X'10': • Rx is not an even-numbered register. • The address contained in Rx is not on a quadword boundary. • The length contained in Rx+1 is not a positive multiple of 16.
Access exception	For subcode X'04' only, an error occurred trying to store BYUSER information into the output buffer. For subcode X'10', an error occurred trying to store the extent information into the guest's output area.
Privileged-operation exception	The virtual machine is in the problem state.

DIAGNOSE Code X'268' – 370 Accommodation Services

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'268' to access services in support of the 370 Accommodation facility. For more information on the 370 Accommodation facility, see Part 6, “Architectural Extensions and Accommodations for Virtual Machines,” on page 881.

Entry Values:

Rx
Contains a function subcode. This subcode selects the service to be performed. The rest of the entry values and the exit values depend upon the specific subcode. A specification exception is recognized if an unassigned subcode is given. The following subcode is recognized:

0
Convert a BC-mode or mapped PSW to EC-mode

Subcode 0 — Convert a BC-mode or mapped PSW to EC mode

This function converts a PSW which is in BC mode or which has been "mapped" by the 370 Accommodation facility to an EC-mode PSW.

Entry Values:

Rx
Contains function subcode 0.

Ry
Contains bytes 0-3 of the PSW to be converted. R15 must not be specified as the Ry register, or a specification exception will result.

Ry+1
Contains bytes 4-7 of the PSW to be converted.

Exit Values:

Ry
If the condition code is zero, Ry contains bytes 0-3 of the converted PSW. If the condition code is non-zero, Ry is unchanged.

Ry+1
If the condition code is zero, Ry+1 contains bytes 4-7 of the converted PSW. If the condition code is non-zero, Ry+1 is unchanged.

Condition Codes: The following condition codes are set by subcode 0:

Condition Code	Status
0	The conversion was successful. The contents of the Ry and Ry+1 general registers are replaced with the converted PSW.
1	The conversion was unsuccessful because the input PSW is not a BC-mode or mapped PSW. The contents of the Ry and Ry+1 general registers are unchanged.

Program Exceptions: These program exceptions may be recognized for subcode 0 of DIAGNOSE code X'268':

Problem Encountered	Cause
Specification exception	R15 is specified for Ry.

Responses

Program Exceptions: These program exceptions may be recognized for any subcode of DIAGNOSE code X'268':

Problem Encountered	Cause
Specification exception	An unknown subcode is specified in Rx.
Privileged-operation exception	The virtual machine is in the problem state.

DIAGNOSE Code X'26C' – Access Certain System Information

Privilege Class: B, E; Subcode X'00000024' and Subcode X'00000030' - G

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'26C' to access certain system information.

Entry Values:

Rx
Contents depend on the value of subcode. Rx must be an even-numbered register and cannot be the same as Ry.

Rx+1
Contains the address of an output buffer on a doubleword boundary.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the doubleword passed as input. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the input doubleword is in the host-primary address space.

Ry
The second data register contains the function subcode. Ry must be an even-numbered register and cannot be the same as Rx.

- Subcode X'00000004' Return the BYUSER ID for a given user ID.
- Subcode X'00000008' Return virtual LAN system information.
- Subcode X'0000000C' Return controller list.
- Subcode X'00000010' Return controller information.
- Subcode X'00000014' Return guest LAN list.
- Subcode X'00000018' Return guest LAN information.
- Subcode X'0000001C' Return virtual switch list.
- Subcode X'00000020' Return virtual switch information.
- Subcode X'00000024' Return virtual port, virtual NIC or HiperSockets logical port information.
- Subcode X'00000030' MAC Services.

Ax+1
Is used only by XC virtual machines in access-register mode. Ax+1 contains the ALET for the address space containing the output doubleword. If an XC-mode virtual machine is not in access register mode, or if the virtual machine is not in XC mode, Ax+1 is ignored.

Exit Values:

Ry+1
Contains a return code value. Meaning is determined by subcode.

Common Usage Notes:

1. Rx and Ry cannot be the same register.
2. Rx and Ry must be even-numbered registers.
3. Many subcodes use *version* as input parameter. *Version* indicates the desired format of the information in the **output** buffer. Note that the version defines the length of the output data, not the content of reserved fields. For example, reserved fields in version N may actually contain some content defined for level N+1.

   Use CSI66219 to provide information in the format described below.

   Valid versions are defined by HCPCSIBK COPY and include:

   Table 29. Version definitions. This describes the versions that may be specified on subcodes that accept a version indicator on input.

Value	HCPCSIBK equate	Description
0000 0001	CSICPVER	z/VM 5.3.0, no service
0000 0001	CSIVERS1	z/VM 5.3.0, no service
0001 0001	CSI64281	z/VM 5.3.0, with APAR VM64281
0002 0001	CSI64277	z/VM 5.3.0, with APAR VM64277
0000 0002	CSIVERS2	z/VM 5.4.0, no service
0000 0003	CSIVERS3	z/VM 6.1.0, no service
0001 0003	CSI64780	z/VM 6.1.0, with APAR VM64780
0000 0004	CSIVERS4	z/VM 6.2.0, no service
0001 0004	CSI65042	z/VM 6.2.0, with APAR VM65042
0000 0005	CSIVERS5	z/VM 6.3.0, no service

Table 29. Version definitions. This describes the versions that may be specified on subcodes that accept a version indicator on input. *(continued)*

Value	HCPCSIBK equate	Description
0001 0005	CSI65583	z/VM 6.3.0, with APAR VM65583
0000 0006	CSIVERS6	z/VM 6.4.0 no service
0001 0006	CSI65925	z/VM 6.4.0 with APAR VM65925
0002 0006	CSI65918	z/VM 6.4.0 with APAR VM65918
0000 0007	CSIVERS7	z/VM 7.1.0, no service
0001 0007	CSI66219	z/VM 7.1.0 with APAR VM66219
0000 0008	CSIVERS8	z/VM 7.2.0, no service

Programs written to use HCPCSIBK COPY on one release of z/VM can be executed on a new release of z/VM without change. To exploit the function provided by a follow-on release, examine your program for necessary changes and use the new value for version.

An error is returned if the version supplied is not supported by the current level of CP.

Subcode X'00000004'—Return the BYUSER ID For a Given User ID

Entry Values:

Rx
Contains the address of an input buffer two fullwords in length. The input buffer contains the user ID (left-justified, followed by blanks) whose BYUSER value is to be returned.

Rx+1
Contains the address of a doubleword that may contain the BYUSER ID (left-justified, followed by blanks) of the specified user ID. The output buffer is only used if there is a BYUSER for the user ID (return code 0).

Table 30. Subcode X'00000004' Return Codes

Return Code in Ry+1	Meaning
0 (X'00')	BYUSER ID was successfully returned in the specified output buffer.
4 (X'04')	BYUSER ID was not returned. There is no BYUSER ID for this user ID.
8 (X'08')	BYUSER ID was not returned. The user ID is not logged on.

Usage Notes:

1. A BYUSER ID is only defined for a virtual machine that is logged on with the BY option of the LOGON command.
2. When a virtual machine is disconnected, its BYUSER ID remains unchanged.
3. A virtual machine's BYUSER ID is updated on a successful reconnect.

Subcode X'00000008'—Return Virtual LAN System Information

Entry Values:

Rx
Contains the address of an input buffer two fullwords in length. The first word of the buffer contains the length of the output buffer (Rx+1). The second word contains the version. Valid version numbers are defined in HCPCSIBK COPY.

This address must be aligned on a doubleword boundary.

Rx+1

Contains the address of an output buffer to contain the virtual LAN system information. This structure is mapped by HCPCSIBK COPY. The output buffer is modified only if Ry+1 contains return 0 or return code 16 (X'10').

This address must be aligned on a doubleword boundary.

Output Buffer Format

Table 31. Return Virtual LAN System Information (DSECT CSISRESP) . Length CSISOL51, X'68'. Length for versions prior to CSI65583 was CSISOL31, X'50'. Length for versions prior to CSI64780 was CSISOSIZ, X'30'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSISOVER - Output buffer format version
4	X'04'	CSIVERS1	CSISMNTL - Maintenance level ID
8	X'08'	CSIVERS1	CSISAPAR - Maintenance level. The most recent service update that affects virtual networking support.
3	X'10'	CSIVERS1	CSISMAPF - System MAC address prefix
3	X'13'	CSIVERS1	CSISMASS - System MAC ID range start
3	X'16'	CSIVERS1	CSISMASE - System MAC ID range end
3	X'19'	CSIVERS1	CSISMAUS - User MAC ID range start
3	X'1C'	CSIVERS1	CSISMAUE - User MAC ID range end
1	X'1F'	CSIVERS1	CSISAC - Flags X'80' - System Accounting on X'40' - User Accounting on X'20' - System MAC address protection on
4	X'20'	CSIVERS1	CSISPRCT - Current persistent value
4	X'24'	CSIVERS1	CSISTRCT - Current transient value
4	X'28'	CSIVERS1	CSISPRMX - Persistent limit x'FFFF' is returned for INFINITE
4	X'2C'	CSIVERS1	CSISTRMX - Transient limit x'FFFF' is returned for INFINITE
3	X'30'	CSI64780	CSISUMAP - User MAC address prefix
3	X'33'	CSI64780	Reserved.
1	X'36'	CSI64780	Reserved.
1	X'37'	CSI64780	Reserved.
16	X'38'	CSI64780	Reserved.
6	X'48'	CSI65583	CSISDMAC - IVL Multicast MAC address associated with the IVL domain to which this system belongs.

Table 31. Return Virtual LAN System Information (DSECT CSISRESP) . Length CSISOL51, X'68'. Length for versions prior to CSI65583 was CSISOL31, X'50'. Length for versions prior to CSI64780 was CSISOSIZ, X'30'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'4E'	CSI65583	CSISIVLD - IVL domain to which this system belongs.
1	X'4F'	CSI65583	Reserved.

8	X'50'	CSI65583	CSISDCAP - domain capabilities.
			As z/VM enhancements are made IVL domain members may be using different service levels or different releases of z/VM. The *capabilitystring* for each system indicates which enhancements are available on that IVL domain member.
			`8000000000000000 - Base capability`
			`C000000000000000 - Collaborative Load Balance Support`

2	X'58'	CSI65583	CSISVLAN - VLAN ID assigned to the domain.
2	X'5A'	CSI65583	CSISHBTO - IVL Heartbeat timeout for the domain.
4	X'5C'	CSI65583	CSISICTR - Count of CSISISTR structures.
4	X'60'	CSI65583	CSISIOFF - When CSISICTR is not zero, byte offset, relative from the start of this record, to the start of the first CSISISTR structure.
4	X'64'	CSI65583	CSISISSZ - Length of the CSISISTR structure.

Table 32. Return IVL Membership Information (DSECT CSISISTR) . Length CSISISLN, X'28'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSI65583	CSISMNAM - Member System Name.
6	X'08'	CSI65583	CSISMMAC - Unicast MAC address assigned to member.
1	X'0E'	CSI65583	CSISMSTA - Member State X'80' - Active X'40' - Inactive X'20' - Leaving X'10' - Error X'08' - Pending Join
1	X'0F'	CSI65583	Reserved.
8	X'10'	CSI65583	CSISMCAP - Member capabilities.
4	X'18'	CSI65583	CSISMHBC - IVLPORT Heartbeat count.
4	X'1C'	CSI65583	CSISMMNT - Maintenance Level ID.

Table 32. Return IVL Membership Information (DSECT CSISISTR) . Length CSISISLN, X'28'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'20'	CSI65583	CSISPSTA - Shared Port Group Status Contains a high level view of the synchronization status of all shared port groups in the system. X'80' - Not Synchronized X'40' - Error X'20' - Pending Synchronization X'10' - Synchronized X'00' - No Shared Port Groups
1	X'21'	CSI65583	CSISGSTA - Global VSwitch Status Contains a high level view of the synchronization status of all global virtual switches in the system. X'80' - Not Synchronized X'40' - Error X'20' - Pending Synchronization X'10' - Synchronized X'00' - No Global Virtual Switches
6	X'22'	CSI65583	Reserved.

Exit Values:

Table 33. Subcode X'00000008' Return Codes

Return Code in Ry+1	Meaning
0 (X'00')	Virtual LAN system information was successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
12 (X'0C')	The requested version is not supported by CP.
16 (X'10')	The buffer provided is not large enough to contain the virtual LAN system information. The fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.

Usage Notes: None.

Subcode X'0000000C'—Return Controller List

Entry values:

Rx

Contains the address of an input buffer one fullword in length. The input buffer contains the length of the output buffer (Rx+1).

This address must be aligned on a doubleword boundary.

Rx+1
Contains the address of an output buffer to contain the list of controller virtual machine names. Each name is padded to 8 bytes. The output buffer is modified only if Ry+1 contains return 0 or return code 16 (X'10').

This address must be aligned on a doubleword boundary.

Exit Values:

Table 34. Subcode X'0000000C' Return Codes

Return Code in Ry +1	Meaning
0 (X'00')	Controller names were successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	Controller names were not returned. No controllers exist.
16 (X'10')	The buffer provided is not large enough to contain the controller names. Some information may be returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.

Usage Notes: None.

Subcode X'00000010'—Return Controller Information

Entry Values:

Rx
Contains the address of an input buffer four fullwords in length. The first word of the buffer contains the length of the output buffer (Rx+1). The second fullword contains the version. Valid version numbers are defined in HCPCSIBK COPY. The third and fourth words contain a controller name (left-justified, padded with blanks) to obtain information about a specific controller. If the controller name provided is all blanks, information is returned for all controllers.

This address must be aligned on a doubleword boundary.

Rx+1
Contains the address of an output buffer to contain the controller information. This structure is mapped by HCPCSIBK COPY. The output buffer is modified only if Ry+1 contains return 0 or return code 16 (X'10').

This address must be aligned on a doubleword boundary.

Output Buffer Format

Table 35. Return Controller Information (DSECT CSICRESP). Length CSICOSIZ, X'08' This structure also includes CISCCCTR entries of type CSICCSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSICOVER - Output buffer format version
4	X'04'	CSIVERS1	CSICCCTR - Controller count

Table 35. Return Controller Information (DSECT CSICRESP). Length CSICOSIZ, X'08' This structure also includes CISCCCTR entries of type CSICCSTR. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
**	X'08'	CSIVERS1	CSICCBEG - CSICCCTR entries of type CSICCSTR

Table 36. Controller Information (DSECT CSICCSTR). Length CSICCLEN, X'34'. This structure also includes CSICVCTR entries of type CSICVSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSICNEXT - Offset of next controller entry
8	X'04'	CSIVERS1	CSICONAM - Controller name
1	X'0C'	CSIVERS1	CSICAV - Availability X'00' - Not available X'01' - Available
3	X'0D'	CSIVERS1	Reserved
2	X'10'	CSIVERS1	CSICRNGS - Virtual device range start
2	X'12'	CSIVERS1	CSICRNGE - Virtual device range end
20	X'14'	CSIVERS1	CSICTCPI - TCP/IP stack level string
4	X'28'	CSIVERS1	CSICFL Status flags Byte 1 .... ..1. Failover timeout function enabled ....1 Controller is stalled The remainder of byte 1 and all of bytes 2-4 is reserved for future use.

Table 36. Controller Information (DSECT CSICCSTR). Length CSICCLEN, X'34'. This structure also includes CSICVCTR entries of type CSICVSTR. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'2C'	CSIVERS1	CSICAP - Capability flags Byte 1 1... .... IP .1.. .... Ethernet ..1. .... Release Level Information ...1 .... VLAN_ARP .... 1... GVRP .... .1.. Management and Link Aggregation .... ..1. Isolation ....1 Reserved Byte 2 1... .... Reserved .1.. .... Bridge Capable ..1. .... VEPA ...1 .... Enhanced Grat ARP (VM65104+) .... 1... Shared Link Aggregation (VM65583+) .... .1.. Priority Queuing (VM66219+) The remainder of byte 2 and all of bytes 3 and 4 is reserved for future use.
4	X'30'	CSIVERS1	CSICVCTR - Count of controlled virtual switches
**	X'34'	CSIVERS1	CSICVBEG - CSICVCTR entries of type CSICVSTR

Table 37. Vswitch Information (DSECT CSICVSTR). Length CSICVLEN, X'0C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSIVERS1	CSICVNAM - Virtual switch name
1	X'08'	CSIVERS1	CSICVR - Role X'80' - UPLINK Port Active, Primary RDEV X'40' - UPLINK Port Backup RDEV X'20' - Bridge Port Active RDEV X'10' - Bridge Port Standby RDEV X'08' - Bridge Port Inactive RDEF X'04' - Bridge Port Undefined RDEV
1	X'09'	CSIVERS1	Reserved
2	X'0A'	CSIVERS1	CSICVDEV - Virtual device address

Exit Values:

Table 38. Subcode X'00000010' Return Codes

Return Code in Ry +1	Meaning
0 (X'00')	Controller information was successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	Controller information was not returned. The specified user ID is not a controller or no controllers exist.
12 (X'0C')	The requested version is not supported by CP.
16 (X'10')	The buffer provided is not large enough to contain the controller information. Some information may be returned. The value of *controller count* in the output buffer indicates how many complete sets of controller information are returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.
28 (X'1C')	Controller information was not returned. Internal error.

Usage Notes: None.

Subcode X'00000014'—Return Guest LAN List

Entry Values:

Rx

Contains the address of an input buffer one fullword in length. The input buffer contains the length of the output buffer (Rx+1).

This address must be aligned on a doubleword boundary.

Rx+1

Contains the address of an output buffer to contain the guest LAN names. For each guest LAN, the guest LAN owner is returned in the first 8 bytes padded with blanks, followed by the guest LAN name in the next 8 bytes padded with blanks. The output buffer is modified only if Ry+1 contains return code 0 or return code 16 (X'10').

The address must be aligned on a doubleword boundary.

Exit Values:

Table 39. Subcode X'00000014' Return Codes

Return Code in Ry +1	Meaning
0 (X'00')	Guest LAN names were successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	Guest LAN names were not returned. No guest LANs exist.
16 (X'10')	The buffer provided is not large enough to contain the guest LAN names. Some information may be returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.

Table 39. Subcode X'00000014' Return Codes (continued)

Return Code in Ry +1	Meaning
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.

Usage Note: The list of guest LAN names returned can include virtual switches, which are a special kind of guest LAN.

Subcode X'00000018'—Return Guest LAN Information

Entry Values:

Rx

Contains the address of an input buffer six fullwords in length. The first word of the buffer contains the length of the output buffer (Rx+1). The second word contains the version. Valid version numbers are defined in HCPCSIBK COPY. The third and fourth words contain the guest LAN owner (left-justified, padded with blanks). The remaining words contain the guest LAN name (left-justified, padded with blanks).

Providing blanks for guest LAN owner or guest LAN name can return all guest LANs with a given name, or all guest LANs with a given owner, or all guest LANs.

This address must be aligned on a doubleword boundary.

Rx+1

Contains the address of an output buffer to contain the guest LAN information. This structure is mapped by HCPCSIBK COPY. The output buffer is modified only if Ry+1 contains return code 0 or return code 16 (X'10').

The address must be aligned on a doubleword boundary.

Output Buffer Format

Table 40. Return Guest LAN Information (DSECT CSIGRESP). Length CSIGOSIZ, X'08'. This structure also includes CSIGGCTR Entries of Type CSIGGSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIGOVER - Output buffer format version
4	X'04'	CSIVERS1	CSIGGCTR - Guest LAN count
**	X'08'	CSIVERS1	CSIGGBEG - CSIGGCTR entries of type CSIGGSTR

Table 41. Guest LAN Information (DSECT CSIGGSTR). Length CSIGGLEN, X'2C'. This structure also includes CISGACTR entries of type CSIGASTR and CSIGUCTR entries of type CSIGUSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIGNEXT - Offset to next Guest LAN entry
8	X'04'	CSIVERS1	CSIGOOWN - Guest LAN owner
8	X'0C'	CSIVERS1	CSIGONAM - Guest LAN name

Table 41. Guest LAN Information (DSECT CSIGGSTR). Length CSIGGLEN, X'2C'. This structure also includes CISGACTR entries of type CSIGASTR and CSIGUCTR entries of type CSIGUSTR. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'14'	CSIVERS1	CSIGMAX Maximum connections (0 for infinite)
2	X'18'	CSIVERS1	CSIGAT - Attributes Byte 1 1... .... Persistent .1.. .... Restricted ..1. .... Prirouter requested ...1 .... Prirouter active .... 1... Accounting on The remainder of byte 1 and all of byte 2 is reserved for future use.
1	X'1A'	CSIVERS1	CSIGMPRO - MAC address protection setting 0 - Unspecified 1 - On 2 - Off
1	X'1B'	CSIVERS1	Reserved
4	X'1C'	CSIVERS1	CSIGMFS - MFS. (0 for QDIO guest LANs)
1	X'20'	CSIVERS1	CSIGTY - LAN Type 1 - HiperSockets 2 - QDIO 4 - Reserved 5 - Reserved
1	X'21'	CSIVERS1	Reserved
2	X'22'	CSIVERS1	CSIGIPTI - IPTimeout
4	X'24'	CSIVERS1	CSIGACTR - Connected adapter count
**	X'28'	CSIVERS1	CSIGABEG - CSIGACTR entries of type CSIGASTR
4	**	CSIVERS1	CSIGUCTR - CP Authorized user ID count
**	**	CSIVERS1	CSIGUBEG - CSIGUCTR entries of type CSIGUSTR

Table 42. Connected Adapter Information (DSECT CSIGASTR). Length CSIGALEN, X'0C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSIVERS1	CSIGAOWN - Adapter owner
2	X'08'	CSIVERS1	Reserved
2	X'0A'	CSIVERS1	CSIGANIC - NIC address

Table 43. Authorized User Information (DSECT CSIGUSTR). Length CSIGULEN, X'0C'

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSIVERS1	CSIGUSER - Authorized user ID
4	X'08'	CSIVERS1	CSIGUST - Authorization status 1... .... User ID is authorized for promiscuous mode The remainder of byte 1 and all of bytes 2-4 is reserved for future use.

Exit Values:

Table 44. Subcode X'00000018' Return Codes

Return Code in Ry +1	Meaning
0 (X'00')	Guest LAN information was successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	Guest LAN information was not returned. No guest LANs exist to match the specified guest LAN owner and guest LAN name, or no guest LANs exist.
12 (X'0C')	The requested version is not supported by CP.
16 (X'10')	The buffer provided is not large enough to contain the guest LAN information. Some information may be returned. The value of *guest LAN count* in the output buffer indicates how many complete sets of guest LAN information are returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.

Usage Note: The guest LAN information returned can be for a virtual switch, which is a special kind of guest LAN. Use subcode X'00000020' to obtain information specific to virtual switches.

Subcode X'0000001C'—Return Virtual Switch List

Entry Values:

Rx
Contains the address of an input buffer one fullword in length. The input buffer contains the length of the output buffer (Rx+1).

This address must be aligned on a doubleword boundary.

Rx+1
Contains the address of an output buffer to contain the list of virtual switch names. Each name is padded to 8 bytes. The output buffer is modified only if Ry+1 contains return code 0 or return code 16 (X'10').

The address must be aligned on a doubleword boundary.

Exit Values:

Table 45. Subcode X'0000001C' Return Codes

Return Code in Ry+1	Meaning
0 (X'00')	Virtual switch names were successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	Virtual switch names were not returned. No virtual switches exist.
16 (X'10')	The buffer provided is not large enough to contain the virtual switch names. Some information may be returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.

Usage Note: None.

Subcode X'00000020'—Return Virtual Switch Information

Entry Values:

Rx
Contains the address of an input buffer four fullwords in length. The first word of the buffer contains the length of the output buffer (Rx+1). The second word contains the version. Valid version numbers are defined in HCPCSIBK COPY.

The third and fourth words contain a VSWITCH name (left-justified, padded with blanks). If the VSWITCH name provided is all blanks, information is returned for all virtual switches.

OR

The third word contains a virtual switch management IP address in hexadecimal format and the fourth word contains hex zeroes. If the third and fourth words contain zeroes, information is returned for all virtual switches.

This address must be aligned on a doubleword boundary.

Rx+1
Contains the address of an output buffer to contain the virtual switch information. The structure is mapped by HCPCSIBK COPY. The output buffer is modified only if Ry+1 contains return code 0 or return code 16 (X'10').

The address must be aligned on a doubleword boundary.

Output Buffer Format
The following diagram shows the layout of structures returned in the output buffer CSIVRESP. The individual structures are shown in subsequent tables.

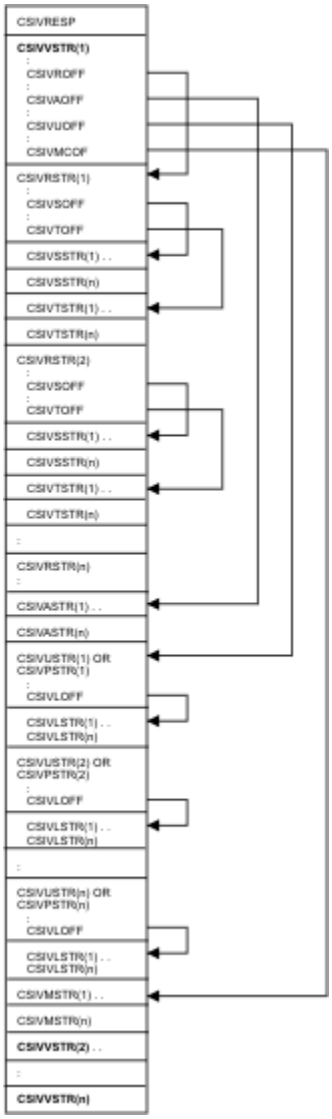


Table 46. Return Virtual Switch Information (DSECT CSIVRESP). Length CSIVOSIZ, X'08'. This structure also includes CSIVVCTR entries of type CSIVVSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIVOVER - Output buffer format version
4	X'04'	CSIVERS1	CSIVVCTR - Virtual switch count
**	X'08'	CSIVERS1	CSIVVBEG - CSIVVCTR entries of type CSIVVSTR

Table 47. Virtual Switch Information (DSECT CSIVVSTR). Length CSIVVL51, X'80'. Length for versions prior to CSI65583 was CSIVVL20, X'5C'. Length for versions prior to CSIVERS2 was CSIVVLEN, X'50'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIVNEXT - Offset to next virtual switch entry

Table 47. Virtual Switch Information (DSECT CSIVVSTR). Length CSIVVL51, X'80'. Length for versions prior to CSI65583 was CSIVVL20, X'5C'. Length for versions prior to CSIVERS2 was CSIVVLEN, X'50'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'04'	CSIVERS1	CSIVONAM - Virtual switch name
1	X'0C'	CSIVERS1	CSIVTY - Virtual switch transport type 1 - IP 2 - Ethernet
1	X'0D'	CSIVERS1	CSIVST - Virtual Switch UPLINK port status X'01' - Vswitch defined X'02' - Controller not available X'03' - Operator intervention required X'04' - Disconnected X'05' - VDEVs attached to controller (1) X'06' - OSA initialization in progress (1) X'07' - OSA device not ready X'08' - OSA device ready X'09' - OSA devices being detached (1) X'0A' - VSWITCH delete pending (1) X'0B' - VSWITCH failover recovering (1) X'0C' - Autorestart in progress (1) (1) If the virtual switch status contains this value, it is expected to be temporary. Remaining in this status is an indicator that an error has occurred.
1	X'0E'	CSIVERS1	CSIVPRT - Default Port type 1 - Access 2 - Trunk (0 for a VLAN unaware virtual switch)
1	X'0F'	CSIVERS1	CSIVQUEU - QueueStorage

Table 47. Virtual Switch Information (DSECT CSIVVSTR). Length CSIVVL51, X'80'. Length for versions prior to CSI65583 was CSIVVL20, X'5C'. Length for versions prior to CSIVERS2 was CSIVVLEN, X'50'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'10'	CSIVERS1	CSIVAT Attributes Byte 1 1... .... Persistent .1.. .... Restricted ..1. .... Prirouter requested ...1 .... Prirouter active .... 1... Accounting on .... .1.. VLAN_ARP .... ..1. GVRP requested ....1 GVRP enabled Byte 2 1... .... VLAN_counters ON .1.. .... Reserved ..1. .... Uplink port NIC ...1 .... NOUPLINK .... 1... Port Based .... .1.. User Based .... ..1. Isolation Status ON ....1 VEPA Status ON
1	X'12'	CSIVERS1	CSIVMPRO - MAC address protection setting 0 - Unspecified 1 - On 2 - Off
1	X'13'	CSIVERS1	CSIVSB - Virtual Switch Bridge Port Status X'01' - Bridge port defined X'02' - Controller not available X'03' - Operator intervention required X'04' - Disconnected X'05' - VDEVs attached to controller (2) X'06' - Initialization in progress (2) X'07' - Device not ready X'08' - Device is active bridge port X'09' - Devices being detached (2) X'0A' - VSWITCH delete pending (2) X'0B' - VSWITCH failover recovering (2) X'0C' - Autorestart in progress (2) (2) If the virtual switch status contains this value, it is expected to be temporary. Remaining in this status is an indicator that an error has occurred.

Table 47. Virtual Switch Information (DSECT CSIVVSTR). Length CSIVVL51, X'80'. Length for versions prior to CSI65583 was CSIVVL20, X'5C'. Length for versions prior to CSIVERS2 was CSIVVLEN, X'50'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'14'	CSIVERS1	CSIVDEFV - Default VLAN ID (0 for a VLAN unaware virtual switch or X'8000' if VLAN AWARE without a default VLAN ID)
2	X'16'	CSIVERS1	CSIVNATV - Native VLAN ID (0 for a VLAN unaware virtual switch) (1 if native VLAN ID not specified for a VLAN aware virtual switch or X'8000' if VLAN AWARE with native VLAN ID NONE)
2	X'18'	CSIVERS1	CSIVIPTI - IPTimeout
6	X'1A'	CSIVERS1	CSIVMAC - Virtual switch MAC address
8	X'20'	CSIVERS1	CSIVMGMT - Management stack user ID
8	X'28'	CSIVERS1	CSIVUNUS - UPLINK NIC user ID
2	X'30'	CSIVERS1	CSIVUNVD - UPLINK NIC vdev
1	X'32'	CSIVERS1	CSIVUNRN - UPLINK NIC Error status X'00' - No error X'01' - Userid not logged on X'02' - Not authorized X'03' - VDEV does not exist X'04' - VDEV is attached elsewhere X'05' - VDEV not compatible type X'06' - VLAN conflict X'07' - No MAC address X'08' - Not managed X'09' - Port Error X'0D' - Type mismatch X'FF' - Unknown error
1	X'33'	CSIVERS1	Reserved
4	X'34'	CSIVERS1	CSIVOIPA - Virtual switch IP address

Table 47. Virtual Switch Information (DSECT CSIVVSTR). Length CSIVVL51, X'80'. Length for versions prior to CSI65583 was CSIVVL20, X'5C'. Length for versions prior to CSIVERS2 was CSIVVLEN, X'50'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'38'	CSIVERS1	CSIVLIN - Link aggregation attributes 1... .... GROUP attribute specified .1.. .... LACP Active ..1. .... Reserved ...1 .... Reserved .... 1... Shared Port Group .... .1.. Exclusive Port Group The remainder of the byte is reserved for future use. X'00' is returned if this vswitch is not using a shared port group.
1	X'39'	CSIVERS1	CSIVTYP - Virtual Switch Type 2 - QDIO VSWITCH 4 - Reserved 5 - Reserved 6 - IVL VSWITCH
2	X'3A'	CSUVERS1	CSIVINT - Link aggregation group interval
8	X'3C'	CSIVERS1	CSIVGROU - Link aggregation group name (All blanks for NOGROUP)
8	X'44'	CSIVERS2	CSIVTIME - Timestamp in TOD clock format representing the time the virtual switch manager was assigned.
4	X'4C'	CSIVERS2	CSIVVLAC - Number of VLAN IDs activated. A VLAN ID is considered to be activated when at least one guest initialized a port on which the VLAN ID may flow. 0 is returned for a VLAN UNAWARE VSWITCH.
4	X'50'	CSIVERS1	CSIVRCTR - Device count
4	X'54'	CSI65583	CSIVROFF - When CSIVRCTR is not zero, byte offset, relative to the start of this CSIVVSTR structure, to the start of the first CSIVRSTR structure.
4	X'58'	CSI65583	CSIVACTR - Connected adapter count
4	X'5C'	CSI65583	CSIVAOFF - When CSIVACTR is not zero, byte offset, relative to the start of this CSIVVSTR structure, to the start of the first CSIVASTR structure.

Table 47. Virtual Switch Information (DSECT CSIVVSTR). Length CSIVVL51, X'80'. Length for versions prior to CSI65583 was CSIVVL20, X'5C'. Length for versions prior to CSIVERS2 was CSIVVLEN, X'50'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'60'	CSI65583	CSIVUCTR or CSIVPCTR - CP Authorized user ID count for user based virtual switch or Port count for port based virtual switch CSIVUCTR applies to a user-based virtual switch prior to version CSI65925 but is replaced by CSIVPCTR with version CSI65925 and above output for all virtual switches (whether it is user based or port based).
4	X'64'	CSI65583	CSIVUOFF - When CSIVUCTR/CSIVPCTR is not zero, byte offset, relative to the start of this CSIVVSTR structure, to the start of the first CSIVUSTR structure or the first CSIVPSTR structure.
1	X'68'	CSI65918	CSIVLBAL - Load Balance Type 0 - Independent 8 - Forced Independent 16 - Collaborative
1	X'69'	CSI66219	CSIVPRIQ - Priority Queuing state 0 - Off 4 - Forced off 8 - On
5	X'6A'	CSI65918	Reserved.
1	X'6F'	CSI65583	CSIVSPGS - Shared Port Group Scope X'00' - Not Shared Port Group X'80' - Not Synchronized X'40' - Error X'20' - Pending Synchronization X'10' - Synchronized
8	X'70'	CSI65583	CSIVSTOK - Shared Port Group Synchronization token. Zeroes if NOGROUP or port group is not a shared port group.
4	X'78'	CSI65583	CSIVMCTR - Count of the number of Global Virtual Switch members.
4	X'7C'	CSI65583	CSIVMCOF - When CSIVMCTR is not zero, byte offset, relative to the start of this CSIVVSTR structure, to the start of the first CSIVMSTR structure.

Table 48. RDEV Information (DSECT CSIVRSTR) . Length CSIVRL51, X'70'. Length for versions prior to CSI65583 was CSIVRL40, X'50'. Length for versions prior to CSIVERS4 was CSIVRL12, X'28'. Length for versions prior to CSI64277 was CSIVRLEN, X'24'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'00'	CSIVERS1	CSIVDEVA - Base device address
2	X'02'	CSIVERS1	CSIVVDEV - Virtual device address
8	X'04'	CSIVERS1	CSIVCONT - Controller user ID
2	X'0C'	CSIVERS1	CSIVREAD - Read control device address
2	X'0E'	CSIVERS1	CSIVWRIT - Write control device address
2	X'10'	CSIVERS1	CSIVDATA - Data device address
2	X'12'	CSIVERS1	CSIVUNIT - Data device unit
1	X'14'	CSIVERS1	CSIVDST - Device status X'00' - Device is not active X'01' - Device is active X'02' - Device is backup X'03' - Device is standby

Table 48. RDEV Information (DSECT CSIVRSTR) . Length CSIVRL51, X'70'. Length for versions prior to CSI65583 was CSIVRL40, X'50'. Length for versions prior to CSIVERS4 was CSIVRL12, X'28'. Length for versions prior to CSI64277 was CSIVRLEN, X'24'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'15'	CSIVERS1	CSIVER - Error status X'00' - No error X'01' - Portname conflict X'02' - No layer 2 support X'03' - RDEV does not exist X'04' - RDEV is attached elsewhere X'05' - RDEV not compatible type X'06' - Initialization error X'07' - Stalled OSA X'08' - Stalled controller X'09' - Controller connection severed X'0A' - Primary or secondary routing conflict X'0B' - Device is offline X'0C' - Device was detached X'0D' - Type Mismatch (IP<->Ethernet) X'0E' - Insufficient storage in controller virtual machine X'0F' - TCP/IP Configuration conflict X'10' - No link Aggregation support X'11' - OSA-E Attribute mismatch X'12' - Reserved for future use X'13' - OSA-E is not ready X'14' - Reserved for future use X'15' - Attempting restart for device X'16' - Exclusive use error X'17' - Device state is invalid X'18' - Port number is invalid for device X'19' - No OSA Connection Isolation X'1A' - EQID mismatch X'1B' - Incompatible controller X'1C' - No HiperSockets Bridge Support X'1D' - Error on initialization of HiperSockets Bridge device X'1E' - No Reflective Relay X'1F' - Reflective Relay Error X'20' - No VEPA Support X'21' - SWITCHOVER Command Issued X'22' - No Priority Queuing
1	X'16'	CSIVERS1	CSIVPTST - Port status (UPLINK RDEV) X'00' - Error State X'02' - Suspended State X'03' - Waiting State X'04' - Active State

Table 48. RDEV Information (DSECT CSIVRSTR) . Length CSIVRL51, X'70'. Length for versions prior to CSI65583 was CSIVRL40, X'50'. Length for versions prior to CSIVERS4 was CSIVRL12, X'28'. Length for versions prior to CSI64277 was CSIVRLEN, X'24'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'17'	CSIVERS1	CSIVPRSN - Port status reason (UPLINK RDEV) X'00' - No status to report X'01' - Pending QDIO action X'02' - Pending routing assignment X'03' - Port is inoperable X'04' - ABEND occurred while processing this port X'05' - Pending fail back from backup device to link aggregation group X'06' - Waiting for LACP port negotiation to add this port to the link aggregation group. Datagrams are accepted for the port X'07' - Port System ID and or Key do not match our link aggregation group X'08' - Port marked inoperable by the partner switch via LACP protocol X'09' - LACP currently NOT enabled on the partner switch port
8	X'18'	CSIVERS1	CSIVPORT or CSIVPLPR - For UPLINK RDEV: Portname (All blanks for unassigned) For active or standby HiperSockets Bridge RDEV: Active LPAR. Otherwise blanks.
1	X'20'	CSI64277	CSIVOSAP - OSA-Express hardware port number
3	X'21'	CSI64277	Reserved
8	X'24'	CSIVERS4	CSIVEQID - User specified device equivalency ID (EQID). Zero if a user EQID has not been set for the device.
1	X'2C'	CSI65042	CSIVRTYP - Virtual Switch RDEV Type X'00' - UPLINK RDEV X'01' - Primary HiperSockets Bridge RDEV X'02' - Secondary HiperSockets Bridge RDEV
1	X'2D'	CSI65042	CSIVMTUS - For UPLINK RDEV, Path MTU Discovery Setting X'80' - Off X'40' - Value X'20' - External
2	X'2E'	CSI65042	CSIVMTUV - For UPLINK RDEV, Path MTU Discovery Value

Table 48. RDEV Information (DSECT CSIVRSTR) . Length CSIVRL51, X'70'. Length for versions prior to CSI65583 was CSIVRL40, X'50'. Length for versions prior to CSIVERS4 was CSIVRL12, X'28'. Length for versions prior to CSI64277 was CSIVRLEN, X'24'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'30'	CSI65042	CSIVHMFS - For HiperSockets Bridge RDEV, MFS.
4	X'34'	CSI65042	CSIVHBFL - For HiperSockets Bridge RDEV, Buffer Limit.
4	X'38'	CSI65042	CSIVHBFU - For HiperSockets Bridge RDEV, Buffer in use.
4	X'3C'	CSI65042	CSIVHASY - For HiperSockets Bridge RDEV, count of asynchronous data transfers.
4	X'40'	CSI65042	CSIVHCC2 - For HiperSockets Bridge RDEV, SIGa Busy Count.
4	X'44'	CSIVERS5	CSIVPATR - Partner switch capability X'00000000' - Not available X'00000001' - Standard - VEB X'00000002' - Reflective Relay
4	X'48'	CSIVERS5	Reserved.
4	X'4C'	CSIVERS1	CSIVSCTR - Segment count.
4	X'50'	CSI65583	CSIVSOFF - When CSIVSCTR is not zero, byte offset, relative to the start of this CSIVRSTR structure, to the start of the first CSIVSSTR structure.
4	X'54'	CSI65583	CSIVSLN - Length of the CSIVSSTR structure.
4	X'58'	CSI65583	CSIVTCTR - Take-Over MAC address count
4	X'5C'	CSI65583	CSIVTOFF - When CSIVTCTR is not zero, byte offset, relative to the start of this CSIVRSTR structure, to the start of the first CSIVTSTR structure.
4	X'60'	CSI65583	CSIVTLN - Length of the CSIVTSTR structure.
8	X'64'	CSI65583	CSIVTDS - Adapter description, 8 bytes of data CECID - 6 bytes of data that uniquely identifies the processor. PCHID - 2 bytes of data that identifies the Physical Channel ID representing the OSA-Express adapter.
4	X'6C'	CSI65583	Reserved.

Table 49. Segment Information (DSECT CSIVSSTR). Length CSIVSL50, X'AC'. Length for versions prior to CSI66219 was CSIVSL40, X'4C'. Length for versions prior to CSIVERS4 was CSIVS20, X'3C'. Length for versions prior to CSIVER2 was CSIVSLEN, X'24'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'00'	CSIVERS1	CSIVVLAN - VLAN ID or 0. (0 is returned for a VLAN UNAWARE virtual switch, or a VLAN AWARE virtual switch with the VLAN_counters attribute set to OFF.)
2	X'02'	CSIVERS1	Reserved for future IBM use
8	X'04'	CSIVERS1	CSIVRXFM - Received Frames
8	X'0C'	CSIVERS1	CSIVRXDS - Received Frames Discarded
8	X'14'	CSIVERS1	CSIVTXFM - Transmitted Frames
8	X'1C'	CSIVERS1	CSIVTXDS - Transmitted Frames Discarded
8	X'24'	CSIVERS2	CSIVLTMC - Timestamp in TOD clock format representing the time the VLAN most recently became active. A VLAN ID is considered to be activated when at least one guest initialized a port on which the VLAN ID may flow. Null for a VLAN UNAWARE virtual switch, a VLAN AWARE virtual switch with VLAN_counters set to OFF, or for the Native VLAN when it is not in the list of authorized VLANs.
8	X'2C'	CSIVERS2	CSIVLTMU - Timestamp in TOD clock format representing the most recent change to the VLAN configuration. A VLAN configuration change occurs when a port is added or removed from the list of ports on which the VLAN ID may flow. Null for a VLAN UNAWARE virtual switch, a VLAN AWARE virtual switch with VLAN_counters set to OFF, or for the Native VLAN when it is not in the list of authorized VLANs.
4	X'34'	CSIVERS2	CSIVLACT - Number of interfaces on which the VLAN is active. Null for the Native VLAN when it is not in the list of authorized VLANs.
4	X'38'	CSIVERS2	CSIVLDEL - Number of times VLAN was deleted, when VLAN ID is non-zero. Null for the Native VLAN when it is not in the list of authorized VLANs.
8	X'3C'	CSI65042	CSIVRXBT - Total number of bytes received
8	X'44'	CSI65042	CSIVTXBT - Total number of bytes transmitted
8	X'4C'	CSI66219	CSIVTX0B - Transmitted bytes - Queue 0

Table 49. Segment Information (DSECT CSIVSSTR). Length CSIVSL50, X'AC'. Length for versions prior to CSI66219 was CSIVSL40, X'4C'. Length for versions prior to CSIVERS4 was CSIVS20, X'3C'. Length for versions prior to CSIVER2 was CSIVSLEN, X'24'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'54'	CSI66219	CSIVTX0F - Transmitted frames - Queue 0
8	X'5C'	CSI66219	CSIVTX0D - Transmitted frames discarded - Queue 0
8	X'64'	CSI66219	CSIVTX1B- Transmitted bytes - Queue 1
8	X'6C'	CSI66219	CSIVTX1F- Transmitted frames - Queue 1
8	X'74'	CSI66219	CSIVTX1D- Transmitted frames discarded - Queue 1
8	X'7C'	CSI66219	CSIVTX2B - Transmitted bytes - Queue 2
8	X'84'	CSI66219	CSIVTX2F - Transmitted frames - Queue 2
8	X'8C'	CSI66219	CSIVTX2D - Transmitted frames discarded - Queue 2
8	X'94'	CSI66219	CSIVTX3B - Transmitted bytes - Queue 3
8	X'9C'	CSI66219	CSIVTX3F - Transmitted frames - Queue 3
8	X'A4'	CSI66219	CSIVTX3D - Transmitted frames discarded - Queue 3

Table 50. Take-Over MAC Address Information (DSECT CSIVTSTR). Length CSIVTLEN. X'18'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
6	X'00'	CSI65583	CSIVTKM - Take-Over MAC Address
6	X'06'	CSI65583	CSIVTHM - MAC Address of Failing Host system
8	X'0C'	CSI65583	CSIVTVS - Failing VSwitch name
4	X'14'	CSI65583	Reserved

Table 51. Connected Adapter Information (DSECT CSIVASTR). Length CSIVALEN. X'0C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSIVERS1	CSIVAOWN - Adapter owner
1	X'08'	CSIVERS1	CSIVAPTY - Port Type X'00' - Simulated Guest Port X'01' - HiperSockets Logical Port
1	X'09'	CSIVERS1	Reserved
2	X'0A'	CSIVERS1	CSIVANIC - NIC address

Table 52. Authorized User Information (DSECT CSIVUSTR). Length CSIVUL51, X'14'. Length for versions prior to VM65583 was CSIVULEN, X'10'. CSIVUSTR applies to a user-based virtual switch prior to version CSI65925 but is replaced by CSIVPSTR with version CSI65925 and above output.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSIVERS1	CSIVUSER - User ID
4	X'08'	CSIVERS1	CSIVUST - Authorization status 1... .... User ID is authorized for promiscuous mode The remainder of byte 1 and all of bytes 2-4 is reserved for future use.
4	X'0C'	CSIVERS1	CSIVLCTR - Authorized VLAN list count
4	X'10'	CSI65583	CSIVLOFF - When CSIVLCTR is not zero, byte offset, relative to the start of this CSIVUSTR structure, to the start of the first CSIVLSTR structure.

Table 53. Authorized Port Information (DSECT CSIVPSTR). Length CSIVPL51, X'28'. Length for versions prior to VM65583 was CSIVPLEN, X'24'. For versions CSI65925 and above the authorized user/port list is included as a series of CSIVPSTR structures for every virtual switch (whether it is user based or port based).

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS4	CSIVPNUM - Port number
8	X'04'	CSIVERS4	CSIVPSER - User ID
4	X'0C'	CSIVERS4	CSIVPST - Authorization status 1... .... Port is authorized for promiscuous mode The remainder of byte 1 and all of bytes 2-4 is reserved for future use.
1	X'10'	CSI66219	CSIVUPTX - Guest transmission priority level PQUPLINKTX value 0 - z/VM 1 - High 2 - Normal 3 - Low
15	X'11'	CSIVERS4	CSIVPRSV - Reserved for additional port attributes
4	X'20'	CSIVERS4	CSIVPVTR - Authorized VLAN list count
4	X'24'	CSI65583	CSIVPOFF - When CSIVPVTR is not zero, byte offset, relative to the start of this CSIVPSTR structure, to the start of the first CSIVLSTR.

Table 54. Authorized User VLAN Information (DSECT CSIVLSTR). Length CSIVLLEN, X'04'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'00'	CSIVERS1	CSIVUVID - Authorized VLAN ID
2	X'02'	CSIVERS1	Reserved

Table 55. Global Virtual Switch Member Information (DSECT CSIVMSTR). Length CSIVMLEN, X'10'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
8	X'00'	CSI65583	CSIVMNAM - IVL member name.
1	X'08'	CSI65583	CSIVMSTA - State Member State X'80' - Not Synchronized X'40' - Error X'20' - Pending Synchronization X'10' - Synchronized
3	X'09'	CSI65583	Reserved
4	X'0C'	CSI65583	CSIVMERC - Synchronization error code.

Exit Values:

Table 56. Subcode X'00000020' Return Codes

Return Code in Ry +1	Meaning
0 (X'00')	Virtual switch information was successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 X'04')	Virtual switch information was not returned. The specified VSWITCH does not exist, or no virtual switch has a management IP address matching the IP address provided, or no virtual switches exist.
12 X'0C')	The requested version is not supported by CP.
16 (X'10')	The buffer provided is not large enough to contain the virtual switch information. Some information may be returned. The value of *virtual switch count* in the output buffer indicates how may complete sets of virtual switch information are returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.

Usage Notes: None.

Subcode X'00000024'—Return Virtual Port, Virtual NIC or HiperSockets Logical Port Information

Entry Values:

Rx

Contains the address of an input buffer six fullwords in length. This is mapped by the CSIPPLST structure in HCPCSIBK COPY. The first word of the buffer contains the length of the output buffer (Rx +1). The second fullword contains the version. Valid version numbers are defined in HCPCSIBK COPY. The high order half word of the third word contains an input parameter list type which defines the format of the remaining input.

The following formats return Virtual Port or Virtual NIC Information. This information is obtained for guests on the VSWITCH itself.

- 0 (X'0000') - CSIPITYU. Return port information by user ID, optionally filtered by VLAN ID and NIC addresses. The low order half word of the third word contains a VLAN ID or hex zeroes. If the halfword contains zeroes, information is returned for all active VLAN IDs.

  The fourth and fifth fullwords contain the name of the virtual machine ID. Use the user ID of the controller to obtain information about the active virtual switch RDEV ports.

  The sixth fullword contains the coupled guest NIC in the two low-order bytes, or use the RDEV to obtain information about an active virtual switch RDEV port. The two high-order bytes must contain zeroes. If the two low-order bytes contain X'FFFF' (CSIPINON), information is returned for all coupled NICs defined for the specified user and all active RDEV ports for which the virtual machine is a VSWITCH controller.
- 1 (X'0001') - CSIPITYI. Return port information by virtual switch management IP address, optionally filtering by VLAN ID and port number. The low order half word of the third word contains a VLAN ID or hex zeroes. If the halfword contain zeroes, information is returned for all active VLAN IDs defined for the virtual switch.

  The fourth word contains a virtual switch management IP address in hexadecimal format.

  The fifth word must contain hex zeroes.

  The sixth fullword contains the port number in hexadecimal format. If the sixth fullword contains zeroes, information is returned for all ports defined for the specified virtual switch. Ports include all virtual NICs coupled to the virtual switch, as well as the VSWITCH's active RDEVs.
- 2 (X'0002') - CSIPITYN. Return virtual NIC information by user ID, optionally filtered by VLAN ID and NIC address. The low order half word of the third word contains a VLAN ID or hex zeroes. If the halfword contains zeroes, information is returned for all active VLAN IDs.

  The fourth and fifth fullwords contain the name of the virtual machine ID for which virtual NICs are defined.

  The sixth fullword contains the NIC in the two low-order bytes. The two high-order bytes must contain zeroes. If the two low-order bytes also contain x'FFFF' (CSIPINON), information is returned for all NICs defined for the specified user.

The following formats return HiperSockets Logical Port Information. This information is obtained for guests on the HiperSockets channel that are bridge capable.

- 20 (X'0014') - CSIPITYH. Return HiperSockets logical port information by VSWITCH name.

  The low order halfword of the third word is unused.

  The fourth and fifth fullwords contain the name of the VSWITCH.

  The sixth fullword is not used and contains zero.

This address must be aligned on a doubleword boundary.

Rx+1

Contains the address of an output buffer to contain the guest LAN information. This structure is mapped by HCPCSIBK COPY. The output buffer is modified only if Ry+1 contains return code 0 or return code 16 (X'10').

The address must be aligned on a doubleword boundary.

Output Buffer Format

Table 57. Return Virtual Port or Virtual NIC Information (DSECT CSIPRESP). Length CSIPOSIZ, X'08'. This structure also includes CSIPNCTR entries of type CSIPNSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIPOVER - Output buffer format version
4	X'04'	CSIVERS1	CSIPNCTR - Port or NIC count
**	X'08'	CSIVERS1	CSIPNBEG - CSIPNCTR entries of type CSIPNSTR

Table 58. Port or NIC Information (DSECT CSIPNSTR). Length CSIPNL61, X'40'. This structure also includes CSIPDCTR entries of type CSIPDSTR. Length for versions prior to CSI66219 was CSIPNLEN, X'3C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIPNEXT - Offset to next port or NIC entry
8	X'04'	CSIVERS1	CSIPOWNE - Port or NIC owner For a guest NIC, this is the NIC owner. For a virtual switch network connection or a HiperSockets Bridge port the user ID of the controller is contained in this field. For a HiperSockets Logical Port, this is the Logical Guest Port ID.
2	X'0C'	CSIVERS1	CSIPADDR - Port or NIC address. For a guest NIC, this is the NIC address. For a virtual switch network connection or a HiperSockets Bridge port this is the RDEV address. For a HiperSockets Logical Port, this is the device number portion of the Logical Guest Port ID.
1	X'0E'	CSIVERS1	CSIPSTA - Port or NIC status For a guest NIC: X'00' - NIC is not coupled X'01' - Coupled but not active X'02' - Coupled and active For a virtual switch network connection or a HiperSockets Bridge port: X'01' - Attached to a controller but not active X'02' - Attached and active

Table 58. Port or NIC Information (DSECT CSIPNSTR). Length CSIPNL61, X'40'. This structure also includes CSIPDCTR entries of type CSIPDSTR. Length for versions prior to CSI66219 was CSIPNLEN, X'3C'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'0F'	CSIVERS1	CSIPTY - Type 1 - HiperSockets NIC 2 - QDIO NIC 3 - QDIO VSWITCH 4 - Reserved 5 - Reserved 6 - IQD Bridge Port
8	X'10'	CSIVERS1	CSIPLAOW - Guest LAN owner (Blank if guest NIC is not connected)
8	X'18'	CSIVERS1	CSIPLANM - Guest LAN name (Blank if guest NIC is not connected)
8	X'20'	CSIVERS1	CSIPPORT or CSIPPLPR - Port name for a guest NIC: Portname or blanks. For active or standby HiperSockets Bridge port: Active LPAR. Otherwise, blanks.
1	X'28'	CSIVERS1	CSIPPT - Port type 0 - Undefined (VLAN Unaware or guest NIC not coupled) 1 - Access 2 - Trunk
1	X'29'	CSIVERS1	CSIPEST - Extended Port Status 1... .... Isolation status ON (0 if guest NIC not coupled) .1.. .... VEPA status ON ..1. .... Uplink NIC Port (0 if guest NIC not coupled) .... ..11 Transport Protocol (VM65918 +) 00: Undetermined (NIC not Coupled) 01: IP (Layer 3) 10: Ethernet (Layer 2) 11: Reserved
2	X'2A'	CSIVERS1	CSIPVDEV - Base virtual device address
4	X'2C'	CSIVERS1	CSIPPTNM - Portnum (0 if never coupled)

Table 58. Port or NIC Information (DSECT CSIPNSTR). Length CSIPNL61, X'40'. This structure also includes CSIPDCTR entries of type CSIPDSTR. Length for versions prior to CSI66219 was CSIPNLEN, X'3C'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'30'	CSIVERS1	CSIPIFIN - IfIndex (0 if not coupled)
4	X'34'	CSIVERS1	CSIPMAXI - MAXinfo
4	X'38'	CSIVERS1	CSIPDCTR - Count of devices
1	X'3C'	CSI66219	CSIPUPTX - Guest transmission priority level PQUPLINKTX value 0 - z/VM 1 - High 2 - Normal 3 - Low
5	X'3D'	CSI66219	Reserved
**	X'40'	CSIVERS1	CSIPDBEG - CSIPDCTR entries of type CSIPDSTR

Table 59. Device Information (DSECT CSIPDSTR) . Length CSIPDL40, X'1C'. This structure also includes CSIPTCTR entries of type CSIPTSTR. Length for versions prior to CSIVERS4 was CSIPDLEN, X'0C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'00'	CSIVERS1	CSIPDATA - Device address
2	X'02'	CSIVERS1	CSIPUNIT - Device unit
1	X'04'	CSIVERS1	CSIPROLE - Device role 0 - Unassigned 1 - Read control 2 - Write control 3 - Data device 4 - Data-Diag device
3	X'05'	CSIVERS1	Reserved
16	X'08'	CSIVERS4	Reserved
4	X'18'	CSIVERS1	CSIPTCTR - Count of data devices (Currently always 0 or 1)
**	X'1C'	CSIVERS1	CSIPTBEG - One entry of type CSIPTSTR when **device Role=3, data device**

Table 60. Data Device Details (DSECT CSIPTSTR). Length CSIPTLEN, X'10'. This structure also includes CSIPSCTR entries of type CSIPSSTR and CSIPMCTR entries of type CSIPMSTR.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS1	CSIPOP - Options Byte 1 1... .... Broadcast .1.. .... Ethernet ..1. .... IPv4 ...1 .... IPv6 .... 1... Multicast .... .1.. Promiscuous enabled .... ..1. Promiscuous denied ....1 VLAN enabled Bytes 2-4 are reserved for future use. >For a HiperSockets Logical Port, these bytes are not used.
1	X'04'	CSIVERS1	CSIPROU - Router Status 1... .... Primary .1.. .... Secondary ..1. .... Multicast ...1 .... MAC address protection ON The remainder of the byte is reserved for future use.
3	X'05'	CSIVERS1	Reserved
4	X'08'	CSIVERS1	CSIPSCTR - Active segment count
**	X'0C'	CSIVERS1	CSIPSBEG - CSIPSCTR entries of type CSIPSSTR
4	**	CSIVERS1	CSIPMCTR - MAC address count
**	**	CSIVERS1	CSIPMBEG - CSIPMCTR entries of type CSIPMSTR

Table 61. Active Segment Information (DSECT CSIPSSTR). Length CSIPSL40, X'34'. Length for versions prior to CSIVERS4 was CSIPSLEN, X'24'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
2	X'00'	CSIVERS1	CSIPVLAN - VLAN ID or 0. A non-zero VLAN ID is returned only when: • a global VLAN ID has been set for the port, or • the port is on a VLAN AWARE virtual switch with the VLAN_counters attribute set to ON.

Table 61. Active Segment Information (DSECT CSIPSSTR). Length CSIPSL40, X'34'. Length for versions prior to CSIVERS4 was CSIPSLEN, X'24'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'02'	CSIVERS1	CSIPVST4 - IPv4 VLAN status flag when VLAN ID is non-zero. 1... .... Assigned by user .1.. .... Assigned by system ..1. .... Authorization Conflict The remainder of the byte is reserved for future use.
1	X'03'	CSIVERS1	CSIPVST6 - IPv6 VLAN status flag when VLAN ID is non-zero. 1... .... Assigned by user .1.. .... Assigned by system ..1. .... Authorization Conflict The remainder of the byte is reserved for future use.
8	X'04'	CSIVERS1	CSIPRXFM - Received Frames
8	X'0C'	CSIVERS1	CSIPRXDS - Received Frames Discarded
8	X'14'	CSIVERS1	CSIPTXFM - Transmitted Frames
8	X'1C'	CSIVERS1	CSIPTXDS - Transmitted Frames Discarded
8	X'24'	CSI65042	CSIPRXBT - Total number of bytes received
8	X'2C'	CSI65042	CSIPTXBT - Total number of bytes transmitted

Table 62. MAC Address Information (DSECT CSIPMSTR). Length CSIPMLEN, X'1C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
6	X'00'	CSIVERS1	CSIPMAC - MAC address
2	X'06'	CSIVERS1	Reserved
16	X'08'	CSIVERS1	CSIPIPV6, or CSIPIV4 IP address For an IPv4 address, bytes 1-12 are reserved and the IP address is returned in bytes 13-16. Hexadecimal zeroes are returned for MAC addresses associated with Virtual Switch RDEV ports. When the IP address cannot be determined, hexadecimal zeroes are returned for MAC addresses associated with guest ports on ETHERNET virtual switches.

Table 62. MAC Address Information (DSECT CSIPMSTR). Length CSIPMLEN, X'1C'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
1	X'18'	CSIVERS1	CSIPFL - Address version X'04' - IPv4 X'06' - IPv6
1	X'19'	CSIVERS1	Reserved
1	X'1A'	CSIVERS1	CSIPATY - Address type X'00' - Unicast MAC or IP address X'01' - Multicast MAC or IP address X'02' - Broadcast MAC
1	X'1B'	CSIVERS1	CSIPAST - IP or MAC address status 1... .... Local .1.. .... Remote ..1. .... Manual ...1 .... Owner .... 1... Error or External Conflict The remainder of the byte is reserved for future use.

Exit Values:

Table 63. Subcode X'00000024' Return Codes

Return Code in Ry+1	Meaning
0 (X'00')	Virtual port or NIC information was successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	Virtual port or NIC information was not returned. • The specified adapter owner does not exist, or • the specified NIC is invalid, or • the specified NIC is not defined for the adapter owner, or • no NICs are defined for the adapter owner, or • no NICs match the filtering information provided on input, or • no HiperSockets Logical Ports exist, or • the specified VSWITCH does not exist, or • VLAN filtering was requested but the virtual switch management IP address provided is for a VLAN UNAWARE virtual switch, or a VLAN AWARE switch with VLAN_counters set to OFF.
12 (X'0C')	The requested version is not supported by CP.

Table 63. Subcode X'00000024' Return Codes (continued)

Return Code in Ry+1	Meaning
16 (X'10')	The buffer provided is not large enough to contain the information. Some information may be returned. The value of *port* or *NIC count* in the output buffer indicates how many complete sets of information are returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.
24 (X'18')	Incorrect parameter list type (CSIPITYP) or reserved field not zero.
28 (X'1C')	The issuer has G privilege class and is not authorized to obtain information for the virtual machine name specified (CSIPIUSR) in plist CSIPPLST structure.

Usage Notes: Privilege class G users are able to obtain information for their virtual machine. Information pertaining to other users will not be returned.

Subcode X'00000030'—MAC Services

Entry Values:

Rx

Contains the address of a buffer four fullwords in length. This is mapped by the CSIMPLST structure in HCPCSIBK COPY.

The first word of the buffer contains the length of the output buffer (Rx+1). The second fullword contains the version. Valid version numbers are defined in HCPCSIBK COPY.

The high order halfword of the third word contains the MAC services operation code.

X'0000' - Return MAC address

The low order halfword contains the virtual device number in hexadecimal format.

The fourth fullword is reserved.

Rx+1

Contains the address of an output buffer to contain the results of the MAC Service requested. This structure is mapped by HCPCSIBK COPY. The output buffer is modified only if Ry+1 contains return 0 or return code 16 (X'10').

This address must be aligned on a doubleword boundary.

Output Buffer Format

Table 64. MAC Services Return MAC Address (DSECT CSIMRESP). Length CSIMOSIZ, X'0C'.

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
4	X'00'	CSIVERS2	CSIMOVER - Output buffer format version

Table 64. MAC Services Return MAC Address (DSECT CSIMRESP). Length CSIMOSIZ, X'0C'. *(continued)*

Length	Current Version Hexadecimal Offset	Earliest Version available	Contents
6	X'04'	CSIVERS2	CSIMOMAC - MAC address assigned to the virtual device. A MAC address is assigned by concatenating the VMLAN MACPREFIX (as de fined by the system configuration file) with the MACID found first in the following priority list: • The MAC ID from a SET NIC MACID command issued for the device, • the MAC ID specified on a NICDEF directory control statement defining the device, which must be in the USER subset of the VMLAN MACIDRANGE SYSTEM range, or • an available MACID from the SYSTEM section of the VMLAN MACIDRANGE.
2	X'0A'	CSIVERS2	CSIMRSV1 - Reserved

Exit Values:

Table 65. Subcode X'00000030' Return Codes

Return Code in Ry+1	Meaning
0 (X'00')	The MAC service response was successfully returned in the specified output buffer. The fullword at the address pointed to by Rx is modified to contain the total length of the data returned in the buffer.
4 (X'04')	The MAC service was not successful. • Operation Code X'0000' - Return MAC Address – The specified virtual device does not exist, or is not a networking device.
12 (X'0C')	The requested version is not supported by CP.
16 (X'10')	The buffer provided is not large enough to contain the information. Some information may be returned. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer needed to return all information.
20 (X'14')	Error obtaining storage. The buffer size provided is too large. Issue the request using a smaller buffer size. The first fullword at the address pointed to by Rx is modified to contain the size of the buffer required.
24 (X'18')	An unsupported MAC Service operation code was specified.
28 (X'1C')	The MAC service was not successful. • Operation Code X'0000' - Return MAC Address – MACID is not available. No more MACIDs are available in the VMLAN SYSTEM MACIDRANGE or MACID is already in use.
32 (X'20')	The MAC service was not successful. • Operation Code X'0000' - Return MAC Address – MAC address is not available. Unable to assign MAC address due to an unstable SSI mode.

Table 65. Subcode X'00000030' Return Codes (continued)

Return Code in Ry+1	Meaning
36 (X'24')	The MAC service was not successful. • Operation Code X'0000' - Return MAC Address – MAC address is not available. Unable to assign MAC address due to the user currently being relocated.
40 (X'28')	The MAC service was not successful. • Operation Code X'0000' - Return MAC Address – MAC address must be obtained from the OSA-Express directly. The service does not support MAC address assignments for IEDN or INMN.

Usage Notes: None.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'26C' is given incorrect data:

Problem Encountered	Cause
Specification exception	• Ry does not contain a valid subfunction code. • Rx and Ry are the same register. • Rx or Ry is not an even-numbered register. • The input buffer is not on a doubleword boundary. • The output buffer is not on a doubleword boundary.
Access exception	An error occurred trying to: • Fetch the data from the input buffer. • Store data into the output buffer.
Privileged-operation exception	Any of the following: • The virtual machine is in the problem state. • The issuer does not have the appropriate privilege class (class E or class G for subcode X'00000030' only).

DIAGNOSE Code X'270' – Pseudo Timer Extended

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'270' to cause CP to store the number of bytes of time information specified in the Ry register at the address specified in the Rx register. DIAGNOSE code X'270' replaces DIAGNOSE code X'0C' for new applications.

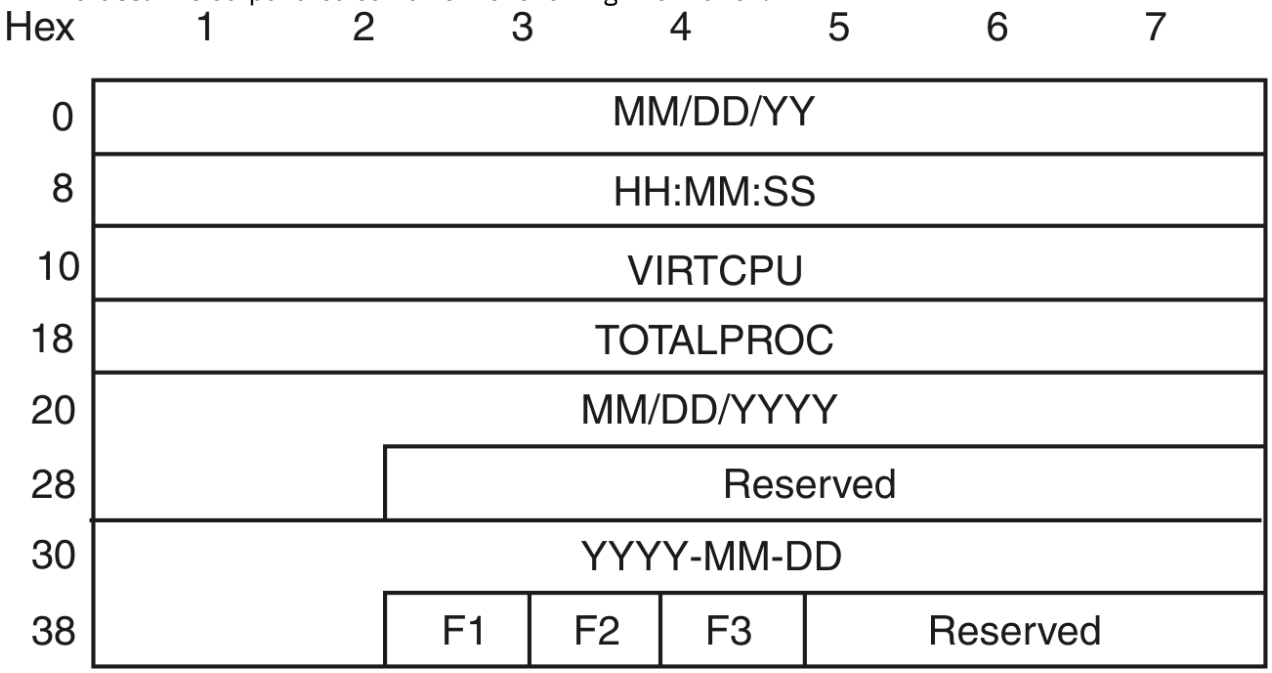
Entry Values:

Rx
Contains the address of an area where the time information is to be stored. The address must be in second-level storage (that is, in the storage that appears real to your virtual machine) and must be on a doubleword boundary.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer where the time information is to be stored. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Ry
Contains the length of the buffer in bytes. If Ry is 0, the same output as the DIAGNOSE X'0C' will be given. 48 bytes are necessary to contain the same output from the DIAGNOSE X'0C' and the date in the full-year format. 64 bytes are needed to contain the same output from the DIAGNOSE X'0C', the date in full-year format and the date in ISO format.

Exit Values: The output area contains the following information:



The byte definitions for F1, F2, and F3 are as follows:

F1
Version of DIAGNOSE 270 (Currently 1)

F2
User's default date format

F3
System default data format

The bit settings in F2 and F3 are as follows:

X'80'
SHORTdate format — mm/dd/yy

X'40'
FULLdate format — mm/dd/yyyy

X'20'
ISOdate format — yyyy-mm-dd

X'10'
Use the system-wide default setting (F2 only). If this bit is on, one of the other bits will be on also. The combination of these bits indicates the value of the system-wide default setting.

The first eight bytes (0 through 7) of the output area contain the date (*mm/dd/yy*) in EBCDIC. The next eight bytes (8 through 15) contain the time of day (*hh:mm:ss*) in EBCDIC. The next eight bytes, VIRTCPU (16 through 23), contain the virtual time consumed by the virtual CPU that issued the DIAGNOSE instruction. The next eight bytes, TOTALPROC (24 through 31), contain the total of the virtual time

(VIRTCPU) and the simulation time spent on behalf of the virtual CPU that issued the DIAGNOSE instruction. Thus, TOTALPROC is always greater than or equal to VIRTCPU. The difference between them represents the time that CP has spent specifically on behalf of the virtual CPU. The next 10 bytes (32 through 41) contain the date in full year format (*mm/dd/yyyy*). The next 6 bytes (42 through 47) are reserved. The next 10 bytes (48 through 57) contain the date in ISO date format (*yyyy-mm-dd*). The next three bytes are three one-byte fields. The first byte is the version of the Diagnose 270. The second byte is the user's default date format. The third byte is the system default date format. The last three bytes are reserved.

These values are also part of the response for the CP INDICATE USER command. Bytes 16 through 31 contain the virtual and total processor time used by the virtual machine that issued the DIAGNOSE instruction. VIRTCPU and TOTALPROC are doubleword, unsigned integers; the time is expressed in microseconds, not as TOD clock units.

Usage Note

For a discussion of how z/VM processes addresses, refer to "How Addresses Are Processed" on page 5.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE X'270' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Any of the following: • The address contained in Rx is not on a doubleword boundary. • Ry is specified as any register except R0 and the user's buffer length is less than or equal to 0. • The user's buffer address is equal to zero. • There is an overlap of Rx and Ry, unless Rx and Ry are both specified as R0.
Privileged-operation exception	The virtual machine is in the problem state.
Access exception (See "Access Exceptions" on page 8.)	An error occurred trying to store into the time-information area.

DIAGNOSE Code X'274' – Set Timezone Interrupt Flag

Privilege Class: G

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'274' to instruct CP to reflect an external interrupt, X'2004', to the issuer's virtual machine when an operator enters the SET TIMEZONE command.

Entry Values:

Rx
contains the subfunction code.

Ry
is reserved and must be specified as 0.

Subfunction Code
Meaning

0
Turns off the normal timezone interrupt flag.

1
Turns on the normal timezone interrupt flag.

2
Turns off the control program timezone interrupt flag. This flag should only be used by the virtual machine's operating system.

3
Turns on the control program timezone interrupt flag. This flag should only be used by the virtual machine's operating system. Starting with CMS Level 11, CMS uses this flag in order to monitor time zone changes on VM.

If either the normal or the control program timezone interrupt flag is turned on and control register 0 bit 19 is on, the floating external interrupt is presented to the virtual machine.

Exit Values:

Guest Cond. Code
Meaning

0
DIAGNOSE completed with no errors.

3
DIAGNOSE failed due to one of the following errors:

- Ry was not specified as 0.
- The subfunction code was not valid.

Usage Notes

1. Interrupt X'2004' is masked by the TOD - clock sync-check subclass mask, control register 0 bit 19. When the interruption is received, DIAGNOSE code X'00' may be used to obtain the new time zone differential.
2. A Subsystem Reset resets both timezone interrupt flags. Such a reset occurs during the processing of commands like IPL, DEFINE STORAGE, SET MACHINE, SYSTEM RESET, SYSTEM CLEAR, and DETACH CPU.
3. HNDEXT SET for interrupt X'2004' can be used by CMS applications wanting to monitor time zone changes. Starting with CMS Level 11, control register 0 bit 19 (CR0.19) is turned on during CMS initialization to enable the X'2004' interruption. Applications which turn CR0.19 off prohibit recognition of the time zone change. The CMS application receives control after CMS processes the X'2004' interrupt.

DIAGNOSE Code X'27C' –Product Enablement Verification

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'27C' to perform product enablement verification. This DIAGNOSE code tests information relating to the product definition within the system. DIAGNOSE code X'27C' requests information regarding the enablement status of a single product or feature. The return code and output area produced by DIAGNOSE code X'27C' contain the following information:

- Whether the product or feature is defined to the system. The product can be defined using a PRODUCT statement within the system configuration file or a SET PRODUCT command.
- The current state of the product (ENABLED or DISABLED).
- The contents of the optional DESCRIPTION operand specified on the PRODUCT statement or SET PRODUCT command.

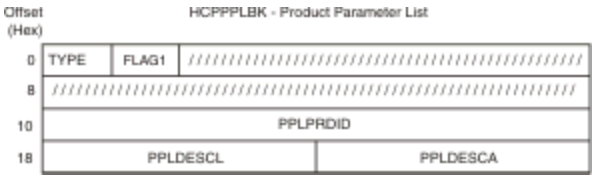
Entry Values:

Rx
Is the guest real address of a product parameter list. This area is mapped by the HCPPPLBK member in HCPGPI MACLIB. This address must be on a doubleword boundary. The required entry values of the product parameter list are described under **Product Parameter List Entry Values**.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the product parameter list and the description return area. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the product parameter list and the description return area are in the host-primary address space.

Ry
Is the length of the product parameter list. This length must be equal to X'20' (PPLBKSZ).

HCPPPLBK — Product Parameter List Format:



HCPPPLBK — Product Parameter List Entry Values:

The required entry values of the product parameter list are:

- The PPLTYPE field must be set to PPLPROD (X'01') to explicitly indicate this is a product query.
- The PPLFLAG1 field must be set to X'00'.
- The reserved fields must be set to X'00'.
- The PPLPRDID field must be set to the VMSES/E product identifier of the product or feature the issuer wanted to query. A seven character product identifier must be padded on the right with a blank.
- To receive any product description data:
  - The PPLDESCL field must contain the number of bytes of the product description the issuer wants to store. It is an unsigned binary number between 0 and 255 (X'00' to X'FF'). If PPLDESCL is not zero, the product description is stored at the guest real address specified by PPLDESCA. If PPLDESCL is zero, the product description is not stored even if one exists.
  - The PPLDESCA field must contain the guest real address where the product description is to be stored. It must be aligned on a doubleword boundary.

Exit Values:

Rx
Is unchanged. The exit values of the product parameter list are described under **Product Parameter List Exit Values**.

Ax
Is unchanged.

Ry
Is the return code from the product enablement verification. The return codes are as follows:

Return Code in Ry	Status
0 (X'00')	The specified product is defined to the z/VM system, and the current status of the product is ENABLED. The fields in the product parameter list are updated as described in **Product Parameter List Exit Values**.
4 (X'04')	The specified product is defined to the z/VM system, but the current status of the product is DISABLED. The fields of the product parameter list are updated as described in **Product Parameter List Exit Values**.

Return Code in Ry	Status
8 (X'08')	The specified product is not defined to the z/VM system. The fields in the product parameter list are not updated.
12 (X'0C')	An unknown error occurred. The fields in the product parameter list are not updated.

HCPPPLBK — Product Parameter List Exit Values:

If the Ry return code is 8 or 12, the fields within the product parameter list are not changed.

If the Ry return code is 0 or 4, the fields within the product parameter list are updated as follows:

- PPLTYPE field is unchanged.
- PPLFLAG1 field is updated as appropriate. Either the PPLENABL bit (X'80') or the PPLDISAB bit (X'40') is set depending on the state of the product within the system. The PPLDESCR bit (X'10') is set if the specified product contains a decription.
- PPLPRDID field is unchanged.
- PPLDESCL field may be changed. It contains the actual number of bytes of the product description that was stored. This is either the number of bytes of the actual product description or the value of the input PPLDESCL, whichever is smaller.
- PPLDESCA field is unchanged.

Usage Note

For a discussion of how z/VM processes addresses , refer to “How Addresses Are Processed” on page 5.

Responses

Program Exceptions: These program exceptions can occur if the DIAGNOSE code X'27C' is given incorrect input data:

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.
Access exception	An error occurred trying to fetch from or store into the parameter list.
Specification exception	Any of the following: • The address of the parameter list specified in Rx is not on a doubleword boundary. • The PPLTYPE field is not X'01' (PPLPROD). • The PPLFLAG1 field is not zero. • The parameter list description length (PPLDESCL) is not in the range of 0 to 255. • The parameter list description return address (PPLDESCA) is not on a doubleword boundary. • Ry contains an invalid length. The only valid length is X'20'. • Reserved fields in the product parmeter list are not zero.

DIAGNOSE Code X'288' - Control Virtual Machine Time Bomb

Privilege Class: Any

Addressing Mode: 31-bit or 64–bit

Use DIAGNOSE code X'288' to set, change, or cancel a virtual machine time bomb. A time bomb explodes after an interval defined by the Diagnose issuer unless it is changed or cancelled. In practice, this interface is used to ensure that the virtual machine remains responsive, as indicated by its issuance of the Diagnose before the interval expires. For example, if the virtual machine issues the Diagnose every 10 seconds and specifies an interval of 15 seconds, it will avoid the explosion indefinitely.

Entry Values:

Rx
is a 32-bit value comprising a 16-bit flag field followed by a 16-bit integer function code. In z/Architecture mode, the high-order word of the register is ignored. The following function codes are supported:

0
Initialize

1
Change

2
Cancel

The following flags are supported:

X'8000'
For the Initialize function, enable SET CONCEAL ON for the virtual machine and arm the time bomb for conditions it detects

Rx+1
For the Initialize or Change function, contains the 32-bit interval in seconds that must elapse before the time bomb explodes. In z/Architecture mode, the high-order word of the register is ignored.

Ry
For the Initialize function, contains the 31-bit or 64-bit real address of the command string to be executed if the time bomb explodes.

Ry+1
For the Initialize function, contains the 32-bit length of the command string. In z/Architecture mode, the high-order word of the register is ignored.

Usage Notes

1. The time bomb interval must not be less than 15 seconds, unless it is zero, in which case the time bomb is disarmed.
2. The command string may consist of one or more commands separated by line-end characters (X'15').
3. Registers that are not used by a particular function code are ignored.
4. Flags that are defined but are not used by a particular function code are ignored.
5. The Initialize function must be the first one used by a virtual machine after it IPLs or issues a Cancel function. Thereafter, the Initialize and Change functions may be used as desired until either a system reset occurs (for example, as part of an IPL) or a Cancel function is used, after which another Initialize function is required.

Responses

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'288' is given incorrect input data:

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.
Addressing exception	The time bomb command string is not addressable.

Problem Encountered	Cause
Specification exception	Any of the following: • An undefined flag bit is turned on. • An undefined function code is used. • The time bomb interval is less than 15 seconds and is not zero (Initialize, Change). • The time bomb command string length exceeds 240 bytes or is zero (Initialize). • No time bomb is defined (Change, Cancel).

DIAGNOSE Code X'290' – Perform Privileged Spool Functions

Privilege Class: D

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'290' to perform the privileged spool functions identified by the following subcodes:

- **X'0000'** fetches the page currently being built in a spool file that is open for creation. This data page contains the records most recently added to the file and has not been written to spool. The spool file may be owned by a user other than the issuer of this Diagnose.
- **X'0004'** fetches the external attribute buffer (XAB) data associated with a particular virtual unit record output device. The owner of the device may be any user currently logged on to the system. For more information on the XAB, refer to "External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'" on page 979.

Entry Values:

Rx
Contains the guest real storage address of the input parameter block. This block must be aligned on a doubleword boundary.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the input parameter block. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the input parameter block is in the host-primary address space.

Ry
(32-bit) Bytes 0 and 1 contain the length of the input parameter block in bytes. Bytes 2 and 3 contain the subcode.

Subcode X'0000' – Fetch Current Page of Open Spool File

Input Parameter Block: The input parameter block for DIAGNOSE code X'290' subcode X'0000' is defined in the D29000 DSECT (in HCPD290P COPY) and contains the following input parameters:

D29000VN
(displacement 0, length 1) Version number (must be X'01').

D29000Q
(displacement 1, length 1) Queue on which the target spool file exists:

D29000PU
X'40' punch queue

D29000PR
X'20' printer queue

D29000ID
(displacement 2, length 2) Spool file ID of the target file.

D29000PN
(displacement 4, length 4) Relative page number of the data page that the program expects is the active page in the open spool file; that is, the page where records are currently being written during creation of the spool file. The first data page in the file is relative page number 0. If the page number of the currently active page does not match this value, a nonzero return code is given (see Responses). This mismatch can occur because new records can be written by the owner between the time the application determines the page number and the time this Diagnose executes.

D29000UI
(displacement 8, length 8) User ID of the owner of the target spool file.

D29000BA
(displacement 16, length 8) Buffer address - the 64-bit guest real address of the output buffer where the data page contents should be stored. This buffer is in the same address space that contains the input parameter block, and it must start on a 4 KB boundary.

D29000BL
(displacement 24, length 4) Buffer length in bytes (must be 4096).

D29000R1
(displacement 28, length 4) Reserved (must be zero for future compatibility).

Exit Values:

- Successful:
  - CC = 0
  - Ry (32-bit) = 0
  - The contents of the current spool file page are in the buffer.
- Unsuccessful:
  - CC = 1
  - Ry (32-bit) = nonzero return code (see Responses)
  - The buffer does not contain the requested data.

Usage Note

For a discussion of how z/VM processes addresses, refer to "How Addresses Are Processed" on page 5.

Responses

Condition Codes:

Condition Code	Meaning
0	The contents of the specified open spool file's currently active page buffer have been stored in the guest's buffer. The return code in Ry is 0.
1	The data has not been stored in the buffer. The exact error is defined by the return code.

Return Codes:

Return Code in Ry	Meaning
0 (X'00')	Successful completion.
4 (X'04') D29000UIDNTLG	The user ID specified is not logged on.
8 (X'08') D29000SFDNTEX	The spool file specified does not exist on the specified queue.

Return Code in Ry	Meaning
12 (X'0C') D29000SFNTOPN	The spool file specified is not currently open.
16 (X'10') D29000RBNTCUR	The requested block number is not the current page.
20 (X'14') D29000BFLNTVAL	The buffer length is not 4096.
24 (X'18') D29000BFADNT4K	The buffer is not on a 4 KB boundary.
28 (X'1C') D29000CPIOERR	A CP paging or I/O error occurred.
32 (X'20') D29000QNTVAL	The spool queue specified is not valid.
36 (X'24') D29000ESMFAIL	DIAGNOSE code X'290' use is not authorized.

Program Exceptions: These program exceptions can occur if DIAGNOSE X'290' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Any of the following: • The address contained in Rx is not on a doubleword boundary. • The subcode specified in Ry is not valid. • The input parameter block length in Ry is incorrect. • The input parameter block version number is not valid. • The reserved fields in the input parameter block are not all zeros.
Privileged-operation exception	The issuer of the instruction does not have class D privileges.
Access exception (See “Access Exceptions” on page 8.)	Any of the following: • An error occurred trying to fetch the input parameter block. • An error occurred trying to store the spool page into the guest's buffer.

Subcode X'0004' – Fetch XAB Data from Virtual Printer

Input Parameter Block: The input parameter block for DIAGNOSE code X'290' subcode X'0004' is defined in the D29004 DSECT (in HCPD290P COPY) and contains the following input parameters:

D29004VN
(displacement 0, length 1) Version number (must be X'01').

D29004R1
(displacement 1, length 1) Reserved (must be zero for future compatibility).

D29004DN
(displacement 2, length 2) Virtual address of the target device.

D29004R2
(displacement 4, length 4) Reserved (must be zero for future compatibility).

D29004UI
(displacement 8, length 8) User ID of the owner of the target device.

D29004BA
(displacement 16, length 8) Buffer address - the 64-bit guest real address of the output buffer where the XAB data should be stored. This buffer is in the same address space that contains the input parameter block, and it must start on a 4 KB boundary.

D29004BL
(displacement 24, length 4) Buffer length in bytes (must be 32768).

D29004R3
(displacement 28, length 4) Reserved (must be zero for future compatibility).

Exit Values:

- Successful:
  - CC = 0
  - Rx (32-bit) = actual length of XAB data stored in the buffer
  - Ry (32-bit) = 0
  - The contents of the XABs are in the buffer.
- Unsuccessful:
  - CC = 1
  - Ry (32-bit) = nonzero return code (see Responses)
  - The buffer does not contain the requested XAB data.

Usage Note

For a discussion of how z/VM processes addresses, refer to "How Addresses Are Processed" on page 5.

Responses

Condition Codes:

Condition Code	Meaning
0	The contents of the specified virtual output device's associated XABs have been stored in the guest's buffer. The return code in Ry is 0.
1	The XAB data has not been stored in the buffer. The exact error is defined by the return code.

Return Codes:

Return Code in Ry	Meaning
0 (X'00')	Successful completion.
4 (X'04') D29004UIDNTLG	The user ID specified is not logged on.
8 (X'08') D29004DEVNTEX	The device specified does not exist.
12 (X'0C') D29004DEVNTPRT	The device specified is not a printer.
16 (X'10') D29004DEVNOXAB	The device has no associated XABs.
20 (X'14') D29004BFLNTVAL	The buffer length is not 32768.

Return Code in Ry	Meaning
24 (X'18') D29004BFADNT4K	The buffer is not on a 4 KB boundary.
28 (X'1C') D29004CPIOERR	A CP paging or I/O error occurred.
36 (X'24') D29004ESMFAIL	DIAGNOSE code X'290' use is not authorized.

Program Exceptions: These program exceptions can occur if DIAGNOSE X'290' is given incorrect input data:

Problem Encountered	Cause
Specification exception	Any of the following: • The address contained in Rx is not on a doubleword boundary. • The subcode specified in Ry is not valid. • The input parameter block length in Ry is incorrect. • The input parameter block version number is not valid. • The reserved fields in the input parameter block are not all zeros.
Privileged-operation exception	The issuer of the instruction does not have class D privileges.
Access exception (See “Access Exceptions” on page 8.)	Any of the following: • An error occurred trying to fetch the input parameter block. • An error occurred trying to store the XAB data into the guest's buffer.

DIAGNOSE Code X'2A8' – Network Diagnose

Privilege Class: Any

Addressing Mode: 31-bit or 64-bit

DIAGNOSE code X'2A8' establishes a network connection on a z/VM simulated network device in order to transmit and receive Ethernet frames. It provides a virtual machine with device-independent access to a simulated NIC created with a CP DEFINE NIC command that is coupled with a CP COUPLE command to either an Ethernet VSWITCH or Ethernet QDIO guest LAN.

Entry Values:

Rx
The general register that contains the guest real address of a DIAGNOSE code X'2A8' request block. The request block must be on a doubleword boundary and cannot cross a 4K boundary. The size and format of the request block is determined by the specified operation code in Ry.

Ax
Is not used. The DIAGNOSE code X'2A8' request block must be in the host-primary address space.

Ry
The operation, as follows:

Bits
Function

0-31
Zero

32-39
Operation code:

X'00'
Query Interface

X'01'
Establish Device Connection

X'02'
Send Data Request

X'03'
Receive Data Request

X'04'
Multicast MAC Registration

X'05'
Network Device Options

40-47
Zero

48-63
Device number

Exit Values:

Rx + 1
Contains the response code indicating the results of the request. Refer to "Responses" on page 283 for the response codes that may be returned.

Operation code X'00' - Query Interface

This function returns the information required to establish and activate a network device connection using an operation code X'01' - Establish Device Connection.

The operation code in bits 32-39 of the Ry operand for this query request is X'00'. The Rx operand must specify the guest real address of a 64-byte storage area in which this operation code will store the following information for the virtual device number specified in Ry operand bits 48-63:

	0	1	2	3	4	5	6	7
00	MAC Address						Format	Features
08	Frames		Below 2GB Frames		Reserved		MAC Limit	
10	CCW							
18	Reserved							
20	Reserved							
28	Reserved							
30	Reserved							
38	Operation Code Mask				Reserved			

Field	Length	Description
MAC Address	6	The MAC address that will be assigned to the device connection when established with an operation code X'01' - Establish Device Connection request.

Field	Length	Description
Format	1	Specifies the layout of the returned Query Interface information: **00** Initial release of DIAGNOSE X'2A8'. **01** Format and operation code mask included in the query response.
Features	1	The features supported by this virtual NIC: **X'80'** A program control interruption (PCI) will be used to indicate there are one or more packets waiting to be received using an operation code X'03' - Receive Data Request. **Notes:** 1. Other features may become available in future releases. 2. An attempt to use a feature that is reported as not available may give inconsistent and unpredictable results.
Frames	2	The total number of guest absolute frames required to establish a device connection with an operation code X'01' - Establish Device Connection request.
Below 2GB Frames	2	The number of guest absolute frames below 2GB required to establish a device connection with an operation code X'01' - Establish Device Connection request. The required number of frames below 2GB is included in the Frames field.
Reserved	2	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.
MAC Limit	2	The maximum number of MAC addresses that may be registered using operation code X'04' - Multicast MAC Address Registration.
CCW	8	The format 1 CCW that must be issued with a SSCH instruction to activate the device connection after a successful operation code X'01' - Establish Device Connection request.
Reserved	32	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

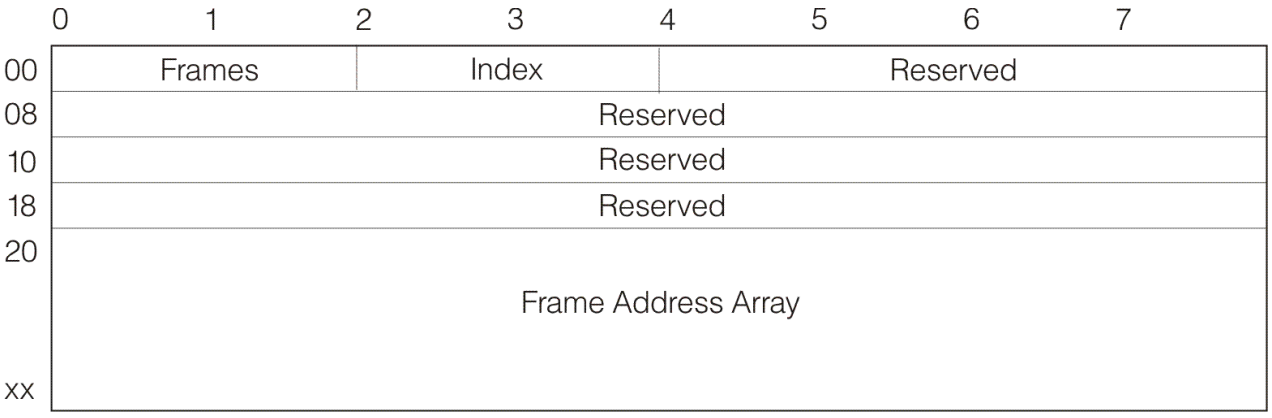
Field	Length	Description
Operation Code Mask	4	Bit mask of supported operation codes: **Bit** **Code – Description** **X'80000000'** 00 – Query Interface **X'40000000'** 01 – Establish Device Connection **X'20000000'** 02 – Send Data Request **X'10000000'** 03 – Receive Data Request **X'08000000'** 04 – Multicast MAC Registration **X'04000000'** 05 – Network Device Options
Reserved	4	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

Operation code X'01' - Establish Device Connection

This function will establish a network connection on a virtual NIC, defined by a CP DEFINE NIC command that is coupled with a CP COUPLE command to either an Ethernet VSWITCH or Ethernet QDIO guest LAN. Prior to establishing a connection, an operation code X'00' - Query Interface must be issued to obtain the information needed to establish and activate this connection.

Prior to issuing an operation code X'01' - Establish Device Connection, the number of guest absolute frames required for Diagnose use (as returned by operation code X'00' - Query Interface) must be provided in guest storage and not modified while the device connection is in use. Unpredictable results can occur if these frames are modified while the device connection is established or activated.

The operation code in bits 32-39 of the Ry operand for this query request is X'01'. The Rx operand must specify the guest real address of a variable-size establish device connection parameter list (EDCPL) that provides the following information for the virtual device number specified in Ry operand bits 48-63:



Once a connection is established, it must be activated by issuing a SSCH instruction to the device specified in Ry with the CCW returned by the operation code X'00' - Query Interface. The I/O request will remain active until it is either terminated by the adapter due to an unexpected error condition or terminated by the guest with either a CSCH or HSCH instruction. As long as the device connection is

active, data can be sent to the LAN by an operation code X'02' - Send Data Request or received with an operation code X'03' - Receive Data Request.

Field	Length	Description
Frames	2	The number of guest real frames given to DIAGNOSE Code X'2A8' to manage the device connection. The number of frames required to establish a device connection is returned by operation code X'00' - Query Interface.
Index	2	When a response code returned in Rx+1 indicates a problem with an entry in the Frame Address Array, the index (0 to x) of the entry in error is set in this field. The field is ignored on entry.
Reserved	28	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.
Frame Address Array	Frames X 8	Variable-size array of 64-bit frames being given to the Network Diagnose to manage the device connection. The size of the array must equal the Frames field multiplied by eight bytes. Each entry must contain the non-zero 64-bit guest absolute address of a frame for the exclusive use of this Diagnose. The frames must remain resident and unmodified in guest storage until the activation CCW terminates. **Note:** The Frame Address Array contains a list of both frames below and above 2GB to be given to the Network Diagnose. The required number of frames below 2GB returned by operation code X'00' Query Interface must be the first frames specified in the Frame Address Array. Failure to specify the frames below 2GB first within the Frame Address Array will result in a response code 32 error. The rest of the frames in the Frame Address Array may reside anywhere in storage.

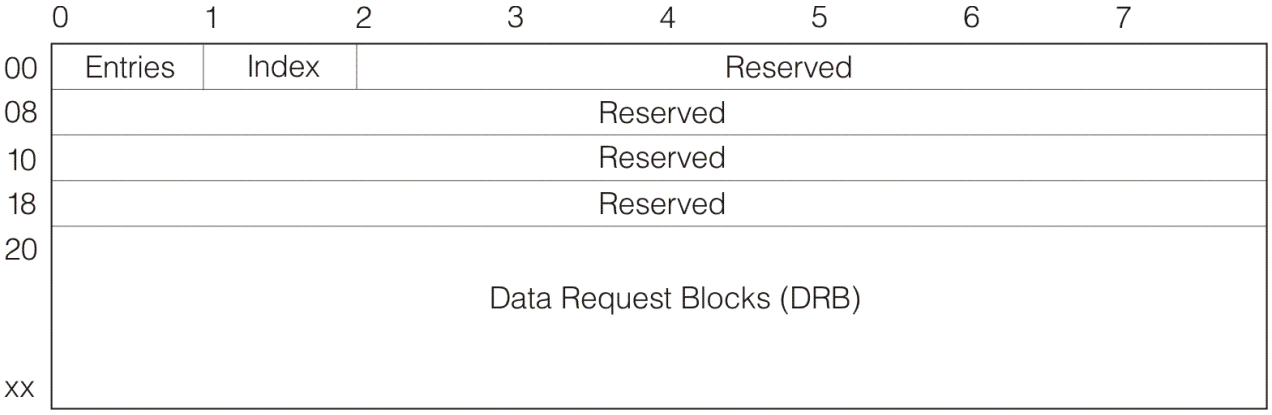
Operation code X'02' - Send Data Request

This function transmits the specified Ethernet frames to the LAN connection established with operation code X'01' - Establish Device Connection and activated with a SSCH of the I/O activation CCW provided by operation code X'00' - Query Interface. As long as the connection is active, data can be sent to the LAN by an operation code X'02' - Send Data Request.

The operation code in bits 32-39 of the Ry operand for this request is X'02'. The Rx operand must specify the 64- or 31-bit guest real address of a network parameter list (NETPL) to transmit Ethernet frames on the device connection activated on the virtual device number specified in Ry operand bits 48-63:

Network Parameter List (NETPL)

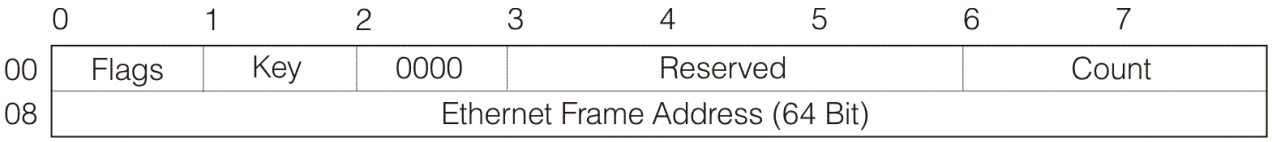
The NETPL is a variable-size control block and is divided into two sections. The first 32 bytes is a fixed-size header that is followed by a variable-size array of data request blocks (DRBs). The address of the NETPL must be on a doubleword boundary and cannot cross a 4K boundary. The NETPL specifies the following information for the virtual device number in Ry operand bits 48-63:



Field	Length	Description
Entries	1	The number of data request blocks (DRBs) associated with a Send or Receive Data Request. A range of 1 to 254 entries may be specified. On completion of an operation code X'03' - Receive Data Request, the Entries field will indicate the number of DGRs filled with Ethernet frames.
Index	1	When a non-zero response code is returned in the Rx+1 operand, the index of the DRB in error is set in this field. The field is ignored on entry.
Reserved	30	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

Data Request Block (DRB)

Appended to the NETPL is an array of data request blocks. Each DRB maps a single packet to be transmitted or received. The number of DRBs to be processed is specified in the Entries field of the NETPL. The maximum number of DRBs that may be specified in a particular NETPL is determined by its starting address. An entire NETPL including the array of DRBs must reside within a single 4K frame. If any portion of the NETPL crosses a 4K boundary, the request will fail.



Field	Length	Description
Flags	1	Processing flags for an operation code X'02' - Send Data Request. Only one of the following flags may be set: **X'01'** Indicates a multicast frame **X'02'** Indicates a broadcast frame **X'04'** Indicates a unicast frame
Key	4 bits	Bits 0-3 of byte 1 contain the storage key to be used when fetching or storing a Ethernet frame.
Reserved	4	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

Field	Length	Description
Count	2	An unsigned half word that specifies the size of the frame to be transmitted for an operation code X'02' - Send Data Request or the size of the buffer for an operation code X'03' - Receive Data Request. For a Receive Data, this count field will indicate the number of bytes filled in the DGRs on completion of the Diagnose.
Ethernet Frame Address	8	Guest real 64-bit address of a Ethernet frame for an operation code X'02' - Send Data Request or an empty buffer for operation code X'03' - Receive Data Request. The Ethernet and VLAN frame headers specified within a SEND Ethernet frame may not cross a 4K boundary. An Ethernet or VLAN frame header crossing a 4K boundary will result in a response code 40 error. **Note:** For performance reasons, it's advisable to keep the entire Ethernet frame within a single 4K frame. Additional Diagnose logic is necessary to process a Ethernet frame which crosses a 4K boundary.

Operation code X'03' - Receive Data Request

This function is used to retrieve Ethernet frames received from the LAN connection established with operation code X'01' - Establish Device Connection. After the network connection is activated, the arrival of inbound frames is signaled by a PCI I/O Interruption on the subchannel on which the I/O CCW provided by operation code X'00' - Query Interface was started via SSCH.

On receipt of a PCI I/O Interruption, an operation code X'03' - Receive Data Request should be issued with an array of empty data request buffers (DGRs). The size of each buffer must be large enough to contain an entire Ethernet frame (equal to or greater than the MTU size for the interface). On completion of the Diagnose instruction, the NETPL Entries field will indicate the number of DGR entries filled with Ethernet frames. The Count field in each filled-in DGR is updated to reflect the number of bytes stored in the Ethernet frame. Operation code X'03' - Receive Data Request should be reissued until all pending Ethernet frames are received. As long as the connection is active, data can be received from the LAN by an operation code X'03' - Receive Data Request.

The operation code in bits 32-39 of the Ry operand for this Receive Data request is X'03'. The Rx operand must specify a 64- or 31-bit guest real address of a network parameter list (NETPL) in order to receive pending Ethernet frames on the device connection activated on the virtual device number specified in Ry operand bits 48-63.

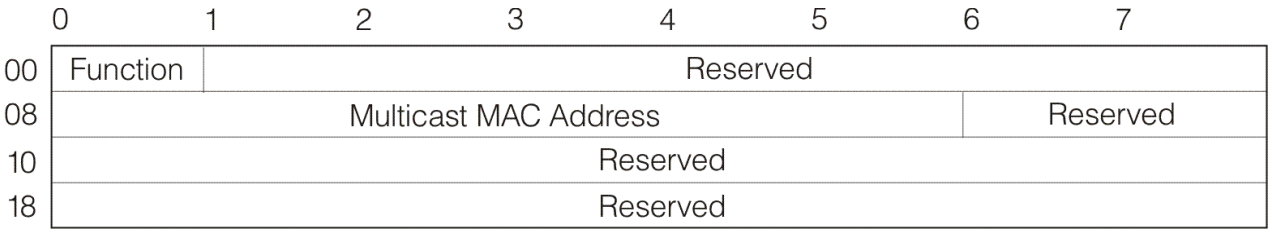
The NETPL is a variable-size control block and is divided into two sections. The first 32 bytes is a fixed-size header and is followed by a variable-size array of Data Request Blocks (DRB). The address of the NETPL must be on a doubleword boundary and cannot cross a 4K boundary. Refer to "Network Parameter List (NETPL)" on page 278 for the format of the NETPL for a Receive Data request.

Operation code X'04' - Multicast MAC Registration

This function is used to register a multicast MAC address on a LAN connection established with an operation code X'01' - Establish Device Connection. After the network connection is activated, inbound multicast Ethernet frames may only be received when their multicast MAC address is registered on the LAN. Operation code X'04' allows a program to either register a multicast MAC address or to remove a previously registered multicast address from an activated network connection.

A multicast MAC address may only be registered after the device connection is activated. A MAC address registration will remain in effect until either it is removed by this operation code or the device connection is deactivated by a CSCH.

The operation code in bits 32-39 of the Ry operand for this registration request is X'04'. The Rx operand must specify a 64- or 31-bit guest real address of a multicast MAC address parameter list (MACPL) in order to receive multicast Ethernet frames on the device connection activated on the virtual device:



Field	Length	Description
Function	1	Function to perform: **X'01'** Assign a multicast MAC address **X'02'** Remove an assigned multicast MAC address All other function codes are reserved for future IBM use.
Reserved	7	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.
Multicast MAC Address	6	The multicast MAC address that will be assigned or unassigned to an active device connection established and activated with an operation code X'01' - Establish Device Connection request.
Reserved	18	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

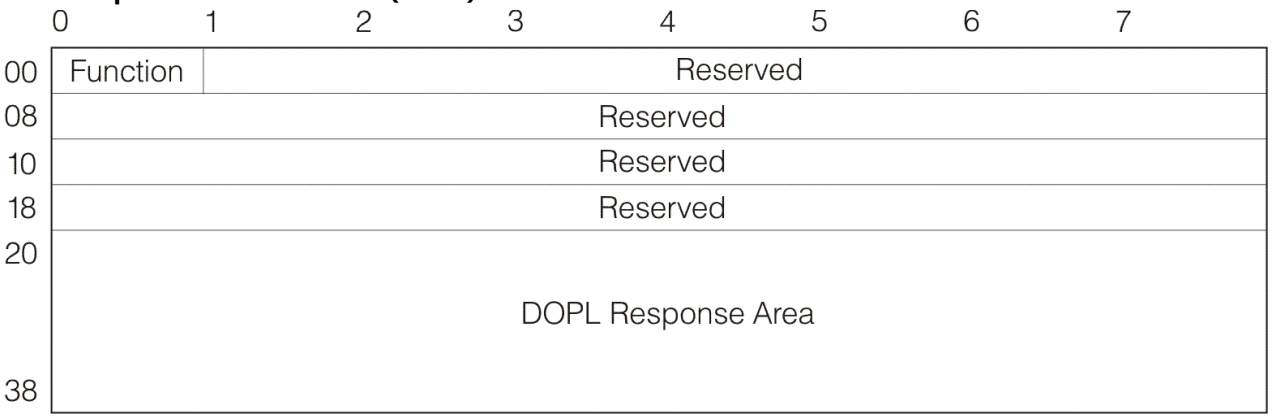
Operation code X'05' - Network Device Options

This function is used to examine or modify options for a network device connection established with an operation code X'01' - Establish Device Connection. A device options parameter list (DOPL) is specified when the operation code is executed. The first 32 bytes of the DOPL specify the function to be performed. The trailing 32 bytes of the DOPL return the active device settings at completion of the operation code X'05' - Network Device Options.

A device option may be modified after the device connection is activated. Any change made will remain in effect until it is either modified by this operation code or the device connection is deactivated.

The operation code in bits 32-39 of the Ry operand for this Network Device Options request is X'05'. The Rx operand must specify the 64- or 31-bit guest real address of a device options parameter list (DOPL).

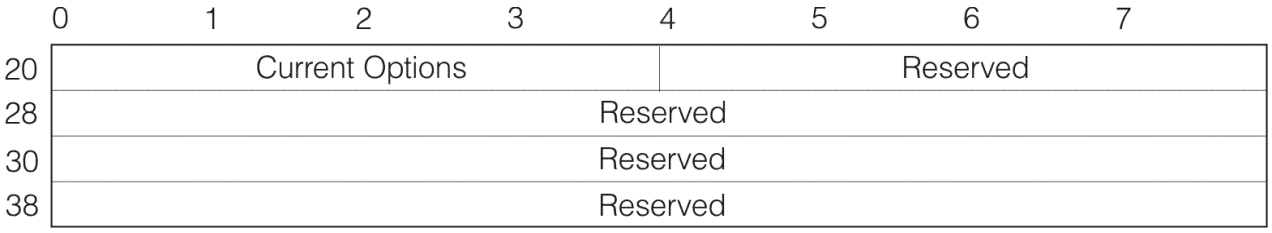
Device Options Parameter List (DOPL)



Field	Length	Description
		Function to perform: **X'00'** Query device options **X'01'** Turn on promiscuous mode **X'02'** Turn off promiscuous mode All other function codes are reserved for future IBM use.
Reserved	31	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

DOPL Response Area

On completion of operation code X'05' - Network Device Options, the DOPL response area contains the device options currently in effect for the active device connection. All fields within the DOPL response area are ignored on entry.



Field	Length	Description
Current Options	4	On a condition code 0 completion of operation code X'05' - Network Device Options, this field will contain the active options for the specified device: **Bit** **Description** **X'80000000'** Promiscuous mode changes are permitted. **X'40000000'** Promiscuous mode is on.

Field	Length	Description
Reserved	28	Reserved (set to binary zeros). Note that programs which place nonzero values in these fields may not operate compatibly in the future.

Responses

Condition Codes:

Upon completion of DIAGNOSE code X'2A8', control is returned to the invoker with a condition code set to indicate the status of both input parameter list processing and the function requested. A response code in Rx+1 further defines the results (see the response code tables that follow).

Condition Code	Meaning
0	Function completed successfully.
1	Function partially completed. Some of the I/O completed successfully. A response code in Rx+1 indicates the condition that caused the partial completion and the Index field in the EDCPL or NETPL indicates the entry in error.
2	Function failed. No I/O has completed successfully. The response code in Rx+1 indicates the reason for the failure.
3	One of the following conditions occurred: • The device number specified in bits 48 to 63 in Ry doesn't exist or its subchannel is not enabled. • The device number is not a virtual NIC coupled to an Ethernet VSWITCH or Ethernet QDIO Guest LAN. • A nonrecoverable error occurred while processing the instruction – a soft ABEND was taken. The response code in Rx+1 indicates the reason for the failure.

Response codes for operation code X'00' - Query Interface are as follows:

Condition Code	Response Code in Rx+1 (decimal)	Meaning
0	0	Query Interface completed successfully.
3	256	The device number specified in bits 48 to 63 in Ry doesn't exist.
3	260	The device number specified in bits 48 to 63 in Ry is not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	264	The Subchannel Enable (E) bit in the Subchannel Information Block (SCHIB) is not enabled for the Device Number specified in bits 48 to 63 in Ry.
3	268	A nonrecoverable error occurred while processing the instruction and a soft ABEND may have been taken. No device information returned. The program should terminate the DIAGNOSE code X'2A8' connection with a CSCH instruction, establish and activate the network connection again to insure the best chance of recovery.

Response codes for operation code X'01' - Establish Device Connection are as follows:

Condition Code	Response Code in Rx+1 (decimal)	Meaning
0	0	Establish Device Connection completed successfully.
2	20	The specified device is currently activated for a network connection or has an outstanding I/O operation.
2	24	The number of frames specified in the EDCPL is not sufficient to establish a device connection.
2	28	An address not on a 4K boundary, an invalid or a zero guest absolute address is specified in a Frame Address Array entry. The Index field in the EDCPL will contain the entry number in error.
2	32	The number of guest absolute frames below 2GB is not sufficient to establish a device connection. The required number of frames below 2GB must be the first frames specified in the Frame Address Array.
2	36	The same frame address was specified more than once in the Frame Address Array. The Index field in the EDCPL will contain the entry number of the duplicate entry.
2	40	The MAC address assigned to this device is currently being used by another device on the LAN.
2	44	The maximum number of multicast MAC addresses allowed to be assigned has been reached for this device connection, The maximum number of MAC addresses allowed to be assigned is returned by operation code X'00' - Query Interface.
3	256	The device number specified in bits 48 to 63 in Ry doesn’t exist.
3	260	The device number specified in bits 48 to 63 in Ry is not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	264	The subchannel enable (E) bit in the subchannel information block (SCHIB) is not enabled for the Device Number specified in bits 48 to 63 in Ry.
3	268	A nonrecoverable error occurred while processing the Diagnose instruction and a soft ABEND may have been taken. The device is in an unpredictable state. The program should terminate the DIAGNOSE code X'2A8' connection with a CSCH instruction, establish and activate the network connection again to insure the best chance of recovery.

Response codes for operation code X'02' - Send Data Request are as follows:

Condition Code	Response Code in Rx+1 (decimal)	Meaning
0	0	Send Data completed successfully.
2	16	The virtual device does not have an activated network connection.
2	20	The virtual device does not have a network device connection established by operation code X'01' - Establish Device Connection.
2	24	An invalid number of entries was specified in the NETPL Entries field.

Condition Code	Response Code in Rx+1 (decimal)	Meaning
1,2	28	Invalid DRB Flag value specified. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	32	Invalid or a zero guest real address is specified in the Address field of a DRB entry. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	36	Storage key violation fetching or storing data in the address specified in a DRB entry. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	40	Insufficient Count value specified in the DRB to hold an Ethernet frame containing both a destination and source MAC address. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	44	A zero count value specified in the DRB. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	48	The MAC address specified in the Ethernet frame header doesn’t match the DRB Flag value specified. The Index field in the NETPL will contain the number of the DRB entry in error.
3	256	The device number specified in bits 48 to 63 in Ry doesn’t exist.
3	260	The device number specified in bits 48 to 63 in Ry is not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	264	The subchannel enable (E) bit in the subchannel information block (SCHIB) is not enabled for the device number specified in bits 48 to 63 in Ry.
3	268	A nonrecoverable error occurred while processing the Diagnose instruction and a soft ABEND may have been taken. The device is in an unpredictable state. The program should terminate the DIAGNOSE code X'2A8' connection with a CSCH instruction, establish and activate the network connection again to insure the best chance of recovery.

Response codes for operation code X'03' - Receive Data Request are as follows:

Condition Code	Response Code in Rx+1 (decimal)	Meaning
0	0	Receive Data completed successfully.
0	4	Receive Data completed successfully. There are additional pending Ethernet frames to retrieve with another Receive Data request.
2	8	No Ethernet frames returned on a Receive Data request.
2	16	The virtual device does not have an activated network connection.
2	20	The virtual device does not have a network device connection established by an operation code X'01' - Establish Device Connection.
2	24	An invalid number of entries was specified in the NETPL Entries field.

Condition Code	Response Code in Rx+1 (decimal)	Meaning
1,2	32	Invalid or a zero guest real address is specified in the Address field of a DRB entry. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	36	Storage key violation fetching or storing data in the address specified in a DRB entry. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	40	Insufficient Count value specified in the DRB to hold the Ethernet frame. The Index field in the NETPL will contain the number of the DRB entry in error.
1,2	44	A zero Count value specified in the DRB. The Index field in the NETPL will contain the number of the DRB entry in error.
3	256	The device number specified in bits 48 to 63 in Ry doesn't exist.
3	260	The device number specified in bits 48 to 63 in Ry is not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	264	The subchannel enable (E) bit in the subchannel information block (SCHIB) is not enabled for the device number specified in bits 48 to 63 in Ry.
3	268	A nonrecoverable error occurred while processing the Diagnose instruction and a soft ABEND may have been taken. The device is in an unpredictable state. The program should terminate the DIAGNOSE code X'2A8' connection with a CSCH instruction, establish and activate the network connection again to insure the best chance of recovery.

Response codes for operation code X'04' - Multicast MAC Registration are as follows:

Condition Code	Response Code in Rx+1 (decimal)	Meaning
0	0	Multicast MAC address registration completed successfully.
0	4	The specified MAC address is already registered for this network connection.
0	8	The specified MAC address being removed is not registered on this network connection.
2	16	The virtual device does not have an activated network connection.
2	20	The virtual device does not have a network device connection established by an operation code X'01' - Establish Device Connection.
2	28	The specified MAC address is not a valid multicast MAC address.
2	32	An invalid function code value was specified.
2	36	The maximum number of multicast MAC addresses allowed to be assigned has been reached for this device connection. The maximum number of multicast MAC addresses allowed to be assigned is returned by an operation code X'00' - Query Interface.

Condition Code	Response Code in Rx+1 (decimal)	Meaning
3	256	The device number specified in bits 48 to 63 in Ry doesn’t exist.
3	260	The device number specified in bits 48 to 63 in Ry is not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	264	The subchannel enable (E) bit in the subchannel information block (SCHIB) is not enabled for the device number specified in bits 48 to 63 in Ry.
3	268	A nonrecoverable error occurred while processing the Diagnose instruction and a soft ABEND may have been taken. The device is in an unpredictable state. The program should terminate the DIAGNOSE code X'2A8' connection with a CSCH instruction, establish and activate the network connection again to insure the best chance of recovery.

Response codes for operation code X'05' - Network Device Options are as follows:

Condition Code	Response Code in Rx+1 (decimal)	Meaning
0	0	Network Device Options completed successfully.
2	16	The virtual device does not have an activated network connection.
2	20	The virtual device does not have a network device connection established by an operation code X'01' - Establish Device Connection.
2	32	An unsupported function code value was specified.
2	36	The user ID is not authorized to set promiscuous mode.
3	256	The device number specified in bits 48 to 63 in Ry doesn’t exist or its subchannel is either not enabled or not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	260	The device number specified in bits 48 to 63 in Ry is not a virtual NIC coupled to a Ethernet VSWITCH or Ethernet QDIO guest LAN.
3	264	The subchannel enable (E) bit in the subchannel information block (SCHIB) is not enabled for the device number specified in bits 48 to 63 in Ry.
3	268	A nonrecoverable error occurred while processing the Diagnose instruction and a soft ABEND may have been taken. The device is in an unpredictable state. The program should terminate the DIAGNOSE code X'2A8' connection with a CSCH instruction, establish and activate the network connection again to insure the best chance of recovery.

Program Exceptions: DIAGNOSE code X'2A8' may result in one of the following program exceptions:

Program Exception	Cause
Privileged-operation	The virtual machine is in the problem state.

Program Exception	Cause
Access	An error occurred trying to: • Store Query Interface information • Fetch or store the establish device connection parameter list (EDCPL) • Fetch or store the network parameter list (NETPL) • Fetch or store the device options parameter list (DOPL).
Specification	One of the following: • Rx is not on a doubleword boundary or it crosses a 4K boundary • Rx equals Ry • Register 15 specified for Rx • Rx+1 overlays Ry • Rx specifies an address of zero
Operand	An invalid operation code is specified in bits 32-39 of Ry.

DIAGNOSE Code X'2CC' – SSI Interface

Privilege Class: B, E

Addressing Mode: 24-, 31-, or 64-bit

Use DIAGNOSE code X'2CC' to perform certain SSI-related functions. DIAGNOSE code X'2CC' provides a virtual machine with access to SSI information.

Entry Values:

Rx
Function code.

Bits	Function
00-31	Ignored
32-63	Function Code: 0 – Obtain Local System SSI Identifiers 1 – Query membership

Ry
Contains a function-specific value and must be an even-numbered register.

Exit Values:

Ry + 1
Contains the response code indicating the results of the request. Refer to the section for each function code to determine the response codes that can be returned.

Responses

Condition Codes and Response Codes:

Upon completion of DIAGNOSE code X'2CC', control is returned to the invoker with the condition code set to indicate the status of both input parameter processing and the function requested. A response code is set in Ry+1 and might further define the results.

If the function code in Rx is not recognized, condition code 3 is set and Ry+1 contains a response code of 256.

Refer to the individual function code descriptions for specific explanations of the meanings of condition codes 0, 1, and 2, as well as for the associated response code values and meanings.

Program Exceptions:

DIAGNOSE code X'2CC' might result in one of the following program exceptions:

Program Exception	Cause
Privileged-operation	Any of the following: • The virtual machine is in the problem state. • The virtual machine does not have privilege class B or E.
Access	An error occurred trying to fetch or store a parameter.
Specification	Any of the following: • Rx and Ry are the same register. • Ry is not an even-numbered register.

Function Code 0: Obtain Local System SSI Identifiers

This function sets the condition code based on whether the local system is a member of the SSI cluster and, if so, returns cluster-related information.

The function code in bits 32-63 of the Rx operand for the Obtain Local System SSI Identifiers request is 0.

The Ry operand must specify the guest logical address of a 32-byte storage area that contains, if the response code is zero:

- The name of the SSI cluster, left-justified and padded with blanks, if necessary, in bytes 0-7
- The name of the local system, left-justified and padded with blanks, if necessary, in bytes 8-15
- A bitmap indicating the slot number of the local system (from left to right), in bytes 16-19
- The slot number of the local system in bytes 20-21
- Zeroes in bytes 22-31.

If the system is not a member of an SSI cluster, condition code 2 is set and the designated 32-byte storage area contains:

- Blanks in bytes 0-7
- The system name, left-justified and padded with blanks, if necessary, in bytes 8-15
- Zeroes in bytes 16-31.

Condition Codes and Response Codes for Obtain Local System SSI Identifiers Function:

Condition code	Response Code in Ry +1	Response Code Meaning
0	0	The local system is an SSI cluster member.
2	248	The local system is not a member of an SSI cluster because it is not configured for an SSI cluster.
2	252	The local system is not a member of an SSI cluster because it was IPLed with the REPAIR parameter, which causes any SSI configuration information to be ignored.

Function Code 1: Query SSI Membership

This function sets the condition code based on whether the system name, whose address is in Ry, is configured as a member of the SSI cluster to which the local system belongs.

The function code in bits 32-63 of the Rx operand for the Query Membership request is 1.

The Ry operand must specify the guest logical address of an eight-byte storage area containing the name of the system whose cluster membership is to be determined. The name of the system must be left-justified and padded with blanks, if necessary.

Condition Codes and Response Codes for Query Membership Function:

Condition code	Response Code in Ry +1	Response Code Meaning
0	0	The specified system is a member of the SSI cluster.
1	4	The specified system is not a member of the SSI cluster.
2	248	The local system is not a member of an SSI cluster because it is not configured for an SSI cluster.
2	252	The local system is not a member of an SSI cluster because it was IPLed with the REPAIR parameter, which causes any SSI configuration information to be ignored.

DIAGNOSE Code X'2E0' – SYSEVENT Query Virtual Server (QVS)

Privilege Class: Any

Addressing Mode: 24-bit, 31-bit, or 64-bit

Use DIAGNOSE code X'2E0' to return information about system, logical partition, and virtual machine capacity. It is equivalent to the MVS™™ SYSEVENT QVS interface, as described in *z/OS® MVS Programming: Authorized Assembler Services Reference, Volume 4 (SET-WTO)*, SA22-7612.

Entry Values:

Rx
Contains the guest logical address of QVS parameter list.

Ry
Contents must be 0.

Exit Values:

Rx
Does not change.

Ry
Contains a return code.

Parameter List: The parameter list consists of an input area followed by an output area. The area must be aligned on a doubleword boundary. This area is mapped by the IRAQVS macro in HCPGPI MACLIB.

Usage Notes

1. SYSEVENT QVS does not report capacity information when z/VM is running on IFL processor engines.
2. Virtual and real specialty engines that may be part of the virtual machine, logical partition, or machine configuration are not included in the capacity values returned by SYSEVENT QVS.

Responses

Upon completion, DIAGNOSE code X'2E0' sets one of the following return codes in Ry:

Return Code
Explanation

0
Successful.

4
Parameter list too short.

The output area is set to zeros with a return code of zero if z/VM is running in a Linux®-only partition, or the DIAGNOSE is issued on a virtual processor that is not a virtual CP.

The virtual machine capacity value returned by DIAGNOSE code X'2E0' is not affected by any SHARE HARDLIMIT setting and reflects the capacity of the lowest-level virtual machine. That is, if the DIAGNOSE is issued by a guest of a z/VM guest, the virtual machine capacity value reflects the capacity of the z/VM guest and not the one issuing the DIAGNOSE instruction.

Program Exceptions: These program exceptions can occur if DIAGNOSE code X'2E0' is given incorrect data:

Problem Encountered	Cause
Specification exception	• The address contained in Rx is not on a doubleword boundary. • The value contained in Ry is not zero.
Access exception	An error occurred trying to store the capacity information into the guest's output area.

DIAGNOSE Code X'2FC' – Obtain Certain Guest Performance Data

Privilege Class: Any, B.

Addressing Mode: 24-bit, 31-bit or 64-bit

Use DIAGNOSE code X'2FC' to obtain certain guest performance data.

Entry Values:

Rx
Contains the guest logical address of the parameter list.

Ax
Is used only for virtual machines in access register mode and contains the ALET for the address space containing the parameter list and the response area. When Rx is general register 0, Ax is not examined; the ALET is assumed to be X'00000000', which indicates the guest host-primary address space.

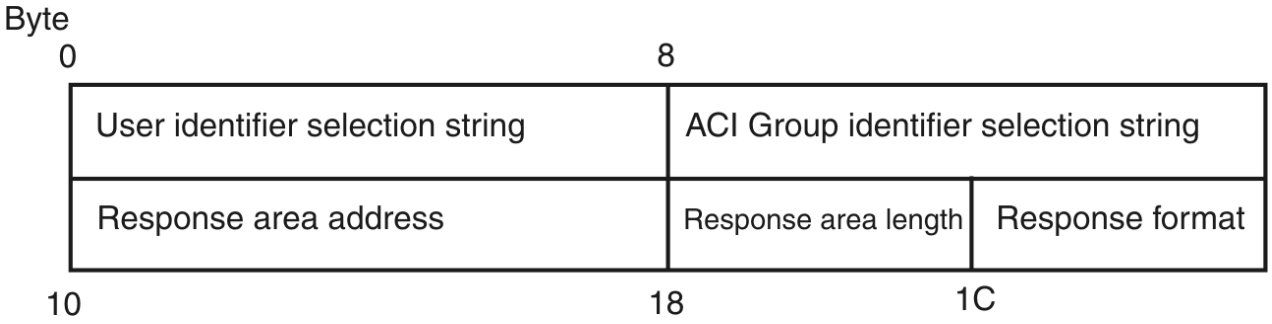
Exit Values:

Rx
Contains the response buffer residual length.

Ax
Does not change.

Ry
Contains a return code.

Parameter List Definition: The parameter list must be aligned on a doubleword boundary. Its format is as follows:



Where:

User identifier selection string (EBCDIC value) This string:

- (Class Any) must be eight blanks to indicate that the issuer's performance data is to be obtained.
- (Class B) must identify a user whose performance data is to be obtained.
- (Class B) must identify a set of users by designating zero or more of the leading characters of their user identifiers, followed by an asterisk (i.e., *) to indicate that any arbitrary subsequent characters are allowed.

ACI Group identifier selection string (EBCDIC value) This string:

- (Class Any) is ignored.
- (Class B) must identify the ACI group name of the users whose performance data is to be obtained.
- (Class B) must identify a set of users by designating zero or more of their ACI group name leading characters, followed by an asterisk (i.e., *) to indicate that any arbitrary subsequent characters are allowed.

Performance data is returned only for users who satisfy both the user identifier and the ACI Group identifier conditions.

Response area address: This is the guest logical address of the doubleword-aligned area in storage where the performance data is to be returned. For a virtual machine in access register mode, the ALET for the address space is contained in Ax. See Ax under Entry Values, for more information.

Response area length: This is the signed 32-bit binary length of the response area.

Response area format: This is the response format and must have the binary integer value 2.

Responses

At the completion of the instruction, the response area contains zero or more sets of performance data. The first word of each set of data in the response is a binary version number indicating its format.

The Format 2, Version 1 response is X'70' (112) bytes long and has the following format:

Offset	Type (Length)	Field Description
0x00	Binary(4)	Version

Offset	Type (Length)	Field Description

0x04	Bit(32)	Guest Flags
		Bits 0-7:
		Primary Virtual CPU type
		`Valid values:`
		`x'00' General Purpose (CP)`
		`x'03' Integrated Facility for Linux (IFL)`
		Bits 8-15:
		Dispatch CPU type
		`Valid values:`
		`x'00' General Purpose (CP)`
		`x'03' Integrated Facility for Linux (IFL)`
		Bits 16-27:
		Reserved (zeros)
		Bit 28:
		Reserved
		Bits 29-30:
		Capping 00-None 01-Soft 10-Hard
		Bit 31:
		Multi-threading 0-Off 1-On

0x08	Binary(8)	Used CPU (uS); total CPU time consumed by all CPUs with the same type as the primary virtual CPU
0x10	Binary(8)	Elapsed Time (uS); time logged on
0x18	Binary(8)	Memory Minimum (KB); reserved pages
0x20	Binary(8)	Memory Maximum (KB); virtual machine storage size
0x28	Binary(8)	Memory Shares (KB); target working set size
0x30	Binary(8)	Used Memory (KB); resident pages
0x38	Binary(4)	Total Active Physical CPUs in CEC
0x3C	Binary(4)	Total Current Logical CPUs in z/VM System
0x40	Binary(4)	Virtual CPUs in Guest with the same type as the primary virtual CPU
0x44	Binary(4)	CPU Minimum; number of guest virtual CPUs with the same type as the primary virtual CPU that are not in the stopped state (0 if guest has ABSOLUTE normal or limit SHARE)
0x48	Binary(4)	CPU Maximum; RELATIVE LIMITSOFT or LIMITHARD SHARE of the dispatch CPU type, if set, 10000 otherwise (0 if guest has ABSOLUTE normal or limit SHARE)
0x4C	Binary(4)	CPU Share; RELATIVE SHARE of the dispatch CPU type (0 if guest has ABSOLUTE normal or limit SHARE)
0x50	Binary(4)	CPU Using Samples; times user found using CPU
0x54	Binary(4)	CPU Delay Samples; times user found waiting for CPU
0x58	Binary(4)	Page Wait Samples; times user found in page wait
0x5C	Binary(4)	Idle Samples; times user found idle

Offset	Type (Length)	Field Description
0x60	Binary(4)	Other Samples; times user found in other state
0x64	Binary(4)	Total Samples; times user state sampled
0x68	Char(8) EBCDIC	User Identifier

Notes:

1. The fields that contain samples (from x'50' through x'64') are calculated only for virtual CPUs of the same type as the primary virtual CPU.
2. For more information about specialty engines and CPU types, see Chapter 1 of *z/VM: Running Guest Operating Systems*.

Condition Code: The condition code is unchanged.

Return Codes: At the completion of the Diagnose instruction, the following return codes are provided in Ry:

Return Code	Meaning
0	SUCCESS Rx contains the residual response area length. Subtract its value from the original response area length to determine the amount of performance data returned. The result could be zero.
-1	FORMAT_NOT_SUPPORTED The Response format value is not 2.
-2	RECEIVE_BUFFER_TOO_SMALL The response area is too small to hold all the performance data and contains only as many complete responses as would fit. Rx contains the residual response area length and will be negative, its complement indicating how much larger the area must be to contain the entire response. Since the size of the response area may change between two invocations of the Diagnose, increasing the area size by this amount might make it either too large or too small to hold a subsequent response. This error will also be returned if the initial response area length is not positive.
-3	INCORRECT_RECEIVE_ADDRESS The response area is not doubleword-aligned, is protected, or is not addressable. The contents of any accessible portion of the response area are unpredictable.
-4	API_EXCEPTION An error occurred fetching the parameter list or storing into the response area. An attempt was made to access another user's performance data without the appropriate (privilege class B) authorization.

Part 2. The Inter-User Communications Vehicle

This part contains the following chapters:

- Chapter 3, “IUCV Overview,” on page 297 which gives an overview of the Inter-User Communications Vehicle (IUCV), and a high-level description of using IUCV to pass information from one virtual machine to another.
- Chapter 4, “IUCV Protocols,” on page 311 which gives reference information on IUCV protocol.
- Chapter 5, “IUCV Function Descriptions,” on page 317 which gives reference information needed to code IUCV functions.

© Copyright IBM Corp. 1991, 2020

Chapter 3. IUCV Overview

The Inter-User Communications Vehicle (IUCV) is a communications facility that allows a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself.

An IUCV communication takes place between a source communicator and a target communicator. The communication takes place over a predefined linkage called a path. Each communicator can have multiple paths, and can receive or send multiple messages on the same path simultaneously.

IUCV provides functions, through the IUCV macro, to:

- Create and dismantle paths
- Send and reply to messages
- Receive or reject messages
- Control the sequence of IUCV events.

Communicators receive information about IUCV events by handling IUCV external interrupts.

To use the IUCV macro, issue the CMS GLOBAL command for HCPGPI MACLIB before assembling your program.

Note: Advanced Program-to-Program Communication/VM (APPC/VM) is based on the IUCV support described in this chapter. Using APPC/VM, a user application program can communicate with a resource manager program in the same system, as with IUCV. With APPC/VM however, an application program can also communicate with a program in another system. This other system could reside in the same TSAF collection (a defined group of z/VM systems), CS collection, or anywhere within a network defined by IBM's Systems Network Architecture (SNA). IUCV connections are restricted to the CS collection, and by default to only virtual machines on the same system.

This chapter only describes communication using IUCV—it does not include information about APPC/VM. Please refer to Part 3, “The Advanced Program-to-Program Communication/VM,” on page 385 for details about APPC/VM and communications outside a single system.

You can write programs that use just the IUCV support described in this chapter, however, CMS IUCV applications should use the CMS support for IUCV and APPC/VM as described in *z/VM: CMS Application Development Guide for Assembler* and *z/VM: CMS Macros and Functions Reference*. The CMS Shared File System, Session Services, private resources, CPI Communications (also known as SAA communications interface), and Coordinated Resource Recovery are just some of the functions and products that require this CMS support.

How Addresses Are Processed

z/VM processes addresses (24-bit or 31-bit) according to the addressing mode being used by the virtual machine. When the guest PSW is in 64-bit addressing mode, IUCV treats addresses as 31-bit addresses.

Note: The IUCV instruction is not supported in access-register mode in an XC virtual machine. It results in a special-operation exception.

The address of the IUCV parameter list is a guest real address in the host-primary address space. All other addresses processed by IUCV are guest absolute addresses in the host-primary address space.

IUCV Paths

The IUCV directory control statement authorizes the establishment of paths between virtual machines, or between a virtual machine and a CP system service. If the maximum number of paths is not specified by

© Copyright IBM Corp. 1991, 2020

the MAXCONN keyword of the OPTION statement in the user's directory, a communicator can establish a maximum of 64 paths.

Once authorized, users establish a path when the source communicator invokes the CONNECT function and the target communicator invokes the ACCEPT function. Either communicator can terminate an established path through the SEVER function. The target communicator can also prevent the establishment of a path by invoking the SEVER function instead of the ACCEPT function. In addition, communication over a path can be temporarily suspended when a communicator invokes the QUIESCE function. The quiesced path can be reactivated when a communicator invokes the RESUME function.

A single communicator can have multiple paths defined, and virtual machines may have multiple paths between them. The communicator could be a source communicator on some of its defined paths, a target communicator on other paths, and both a source and a target communicator on still other paths. Communication over any and all paths can occur simultaneously.

Every path has two ends: the source communicator's end and the target communicator's end. The source communicator has a description of the path from the source's perspective and the target communicator has a description of the same path from the target's perspective.

Each path description has a path ID that is unique for each communicator. IUCV assigns path IDs when communicators invoke the CONNECT and ACCEPT functions. When invoking IUCV functions, the source communicator identifies the path by using the source's path ID. The target communicator identifies the same path to IUCV by using the target's path ID. A path ID is IUCV's method of distinguishing among the paths available to a communicator.

IUCV Messages

An IUCV communication is called a message. The source communicator invoking the SEND function initiates communication and creates a message. The target communicator obtains the message by invoking the RECEIVE function.

The target communicator can optionally request information about messages sent to it by invoking the DESCRIBE function or the INTERRUPT POLL function, and can refuse a message sent to it by invoking the REJECT function. The target communicator can respond to a message through the REPLY function.

Communication is terminated and the message is destroyed when the source communicator issues the TEST COMPLETION function, the INTERRUPT POLL function, or handles an IUCV message complete external interrupt.

Message Data Transfer

When the target communicator issues the RECEIVE function, IUCV moves the message data from the source communicator's SEND virtual address space to the target communicator's RECEIVE virtual address space. When the target communicator issues the REPLY function during a two-way communication, IUCV moves data from the target communicator's REPLY virtual address space to the source communicator's ANSWER virtual address space.

Figure 16 on page 299 illustrates the movement of message data during an IUCV two-way communication.

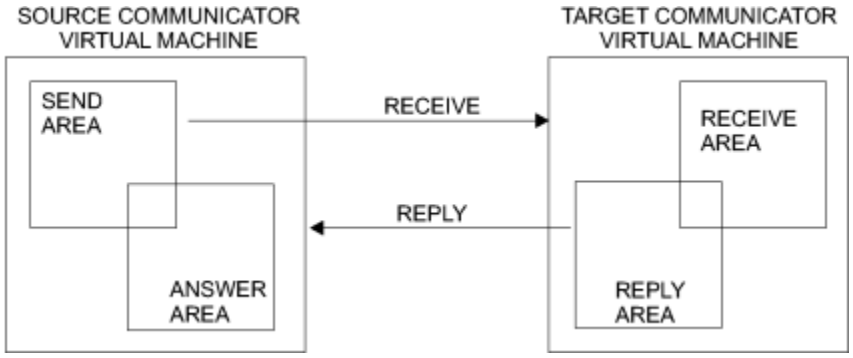


Figure 16. IUCV Two-Way Data Transfer

The source communicator's SEND and ANSWER areas may overlap. Similarly, the target communicator's RECEIVE and REPLY areas may overlap.

CP performs storage protection checking for all data moved during an IUCV communication.

Message Identification

A message is fully identified to a virtual machine by three values. IUCV functions allow one or more of these values to be specified to process messages selectively.

- Message ID

  IUCV assigns a message ID when the source communicator invokes the SEND function. The message ID is generated by a sequential counter value and is unique for the system IPL.
- Message class

  The source communicator identifies a message by using the source message class and target communicator identifies a message by using the target message class. The message classes are arbitrary values that the source communicator specifies when invoking the SEND function. The meaning of the message classes is agreed to in advance by the two communicators. IUCV places no restrictions on the values specified for message class. The communicators can use the message class to handle messages selectively.
- Path ID

  IUCV assigns the path ID when a path is established with the CONNECT function.

There is no defined relationship between the values of the source and target path IDs IUCV assigns, or between the message classes the source and the target communicators use. None of these values need to be the same although they refer to the same message.

The message ID always has the same value for both target and source communicators.

When invoking IUCV functions, the source communicator may refer to a message by a combination of its source path ID, source message class, and message ID. The target communicator may refer to the same message by a combination of its target path ID, target message class, and message ID.

The message tag information may optionally be used by the source communicator to further identify a message. Since IUCV presents the tag to the source communicator when the message completes, the tag may be used to tie the completed message to the original SEND request.

Since a message can be identified as a priority message, the source communicator may also use this as an indication to the target communicator that special handling is required. IUCV queues a priority message ahead of any nonpriority messages and behind any earlier priority messages. A communicator must be authorized to handle priority messages in the IUCV directory control statement.

IUCV External Interrupts

The IUCV external interrupt notifies a virtual machine about IUCV events.

To enable IUCV external interruptions, communicators must:

- Invoke the DECLARE BUFFER function to indicate to IUCV where to store data associated with an external interruption.
- Set bit 7 in the virtual machine's PSW to 1.
- Set submask bit 30 of control register 0 to 1.

IUCV functions generate a type X'4000' external interruption. When a virtual machine in EC mode receives an IUCV external interruption, IUCV places the interruption code in locations X'86' and X'87' of the virtual machine's storage. For a virtual machine in BC mode, IUCV places the code in the external old PSW. In addition, IUCV stores an external interrupt buffer containing information about the message or IUCV function at the address specified when the communicator invoked the DECLARE BUFFER function. One field of this buffer is an external interrupt subtype that indicates why the external interrupt occurred. The possible values of this field are:

- 01 - Connection pending
- 02 - Connection complete
- 03 - Connection severed
- 04 - Connection quiesced
- 05 - Connection resumed
- 06 - Priority message completion
- 07 - Nonpriority message completion
- 08 - Priority message pending
- 09 - Nonpriority message pending.

The first five types are called *control interrupts*, and the last four types are called *message interrupts*.

Whenever there are multiple IUCV interrupts queued for the virtual machine, control interrupts are always reflected to the virtual machine in first-in-first-out (FIFO) order before message interrupts. Message interrupts of the same subtype are reflected in first-in-first-out (FIFO), but message interrupts of different subtypes are reflected in the order shown above.

Interrupts are reflected to the virtual machine in this order regardless of the order in which the interrupts were queued for the virtual machine. There are many conditions which can cause more than one interrupt to be queued for a virtual machine, some of which are beyond the control of the application. For example, if the virtual machine disables for IUCV interrupts for a period of time or if the virtual machine is communicating with multiple partners, then often multiple IUCV interrupts will be on the virtual machine's queue. Also, the relative priorities and time slices given to the communicating virtual machines can affect the order in which IUCV interrupts are presented. For example, if virtual machine A sends a one-way message to virtual machine B, and B receives the message, a message complete interrupt is queued for A. If B then severs the IUCV path, then a sever interrupt is queued for A. If A is not dispatched, or doesn't enable for IUCV interrupts until after the sever interrupt is queued, then A would see the sever interrupt first. If A is dispatched and is enabled for IUCV interrupts before the sever interrupt is queued, then A would see the message complete interrupt first.

A virtual machine can use the SET MASK function to enable or disable external interrupts selectively for IUCV communications. The SET MASK function has mask bits that enable or disable external interruptions for:

- Priority message pending
- Nonpriority message pending
- Priority message completion
- Nonpriority message completion
- IUCV control functions.

To divide and handle the control type interrupts even further, the SET CONTROL MASK function may be used on the IUCV macro. The types of control interrupts may be separately enabled and disabled. These control type interrupts are:

- Connection pending
- Connection complete
- Connection severed
- Connection quiesced
- Connection resumed.

The SET MASK function is interrogated before the SET CONTROL MASK function. If you specify that all control interrupts are disabled using the SET MASK function, then the SET CONTROL MASK settings are not interrogated. If you specify that all control interrupts are enabled using the SET MASK function, then the SET CONTROL MASK settings are interrogated to determine how to handle the individual types of control interrupts.

After IUCV initialization and until you issue the SET MASK or SET CONTROL MASK functions, all IUCV submask bits are on, enabling all IUCV external interrupts.

Avoiding IUCV External Interrupts

A virtual machine can only be notified about an IUCV control function by receiving an external interruption. However, a virtual machine can handle pending messages either by an external interrupts or by using the DESCRIBE function. Message completions can be handled either by an external interrupt or with the TEST COMPLETION function. The INTERRUPT POLL function allows a user to handle message completions and pending messages at the same time.

IUCV also provides the TEST MESSAGE function to determine the presence of any pending messages or message completions. If neither is pending, the virtual machine goes into a wait state until one is pending.

For example, if a source communicator sends a priority message, IUCV queues an external interrupt for the target communicator. If the target virtual machine is enabled for external interrupts, then the target virtual machine receives an external interrupt. However, if the target virtual machine is not enabled, the message remains pending for the virtual machine, and the target virtual machine can issue the DESCRIBE function or the INTERRUPT POLL function to obtain information about the message in the parameter list. The message pending external interrupt is cleared. The target virtual machine can continue processing the message with the RECEIVE or REJECT functions.

Note: If a communicator is enabled for external interrupts and issues the DESCRIBE, INTERRUPT POLL or TEST COMPLETION function, results are unpredictable. However, IUCV supplies information about a message only once.

Two IUCV functions, QUIESCE and RESUME, let a virtual machine control the arrival of message pending external interrupts. The QUIESCE function suspends incoming messages on one or all IUCV paths. Any communicator trying to send a message over a path that has been quiesced receives a return code indicating a quiesced path. No message is created and thus no external interrupt is reflected. The RESUME function restores normal communications.

Security Considerations

Installations control how IUCV is used through the virtual machine directory entries. If the installation has not authorized a user for IUCV communications in the directory, all requests for IUCV communications to virtual machines other than the user's own are denied. Service virtual machines and CP system services defined with the ALLOW (any virtual machine to connect) option do their own authorization checking, and individual directory entries are not needed.

IUCV moves data from one virtual machine address space to another. A virtual machine never has access to the storage or registers of CP or another virtual machine. When the user invokes the RECEIVE or REPLY

functions, the data to be moved is described by a starting address and a length, or a list of starting addresses and lengths. The length specified in the parameter list is the maximum amount of data moved. No requirements are placed on a virtual machine as to the location of these buffers.

IUCV assigns path IDs and records the path ID of each communicator. A given communicator can reference only the paths that have been established for his virtual machine.

IUCV assigns the message ID for each message. IUCV does not use this identifier as a direct reference, but only as an operand in a comparison. It is conceivable that a virtual machine could generate a valid message identifier and use this to request a message. However, when a message ID is used to request a message, a user must also specify a message class and a path ID. If the specified message is not associated with the specified path ID and message class, the user cannot access the messages. If the message ID, path ID, and message class do match, the user could legitimately access the message by specifying simply path ID and/or message class without the generated message ID.

The installation can limit the number of connections for a particular virtual machine by using the MAXCONN parameter of the OPTION control statement in the virtual machine's directory entry.

Virtual Machine-to-Virtual Machine Communication

Three ways of accomplishing virtual machine-to-virtual machine communications are:

- Using data in a buffer
- Using data in a parameter list
- Using control paths.

Using Data in a Buffer

Table 66 on page 302 illustrates a typical sequence of functions invoked when a virtual machine communicates with another virtual machine. The functions include initializing, connecting to another virtual machine, sending and receiving messages, replying to and waiting for messages, severing communications with the other virtual machine, and terminating communications.

Note: Remember, in Table 66 on page 302 data for the messages is stored in a buffer. Refer to “Using Data in a Parameter List” on page 304 for an example of using parameter list data.

Virtual Machine X Communicating to Virtual Machine Y

Table 66. Sequence of Functions

Virtual Machine X	Virtual Machine Y
1. DECLARE BUFFER 2. CONNECT to Y 5. Get External Interrupt 6. SEND to Y 8. TEST COMPLETION 11. Get External Interrupt /or/ TEST COMPLETION 12. SEVER 15. RETRIEVE BUFFER	1. DECLARE BUFFER 3. Get External Interrupt 4. ACCEPT 7. Get External Interrupt /or/ DESCRIBE 9. RECEIVE 10. REPLY 13. Get External Interrupt 14. SEVER 15. RETRIEVE BUFFER

1. Virtual machine X wishes to communicate with virtual machine Y. Both virtual machines must independently invoke the DECLARE BUFFER function. The buffer provides the virtual machine with information about incoming external interrupts concerning IUCV functions.
2. Virtual machine X invokes the CONNECT function, indicating Y as the target. IUCV checks the directory to determine if this connection is authorized. If it is, IUCV queues an external interrupt for Y indicating that there is a connection pending for it. IUCV returns control to X at the next instruction after the CONNECT.
3. The external interrupt queued by step "2" on page 303 is reflected to Y indicating a connection pending. IUCV places the external interrupt information in the buffer that Y provided in step "1" on page 303. IUCV passes control to the external interrupt handler of Y.
4. Virtual machine Y interprets the external interrupt and responds with an ACCEPT to complete the connection. IUCV then completes the connection and queues a Connection Complete external interrupt for X. IUCV returns control to Y at the next instruction after the ACCEPT.
5. The external interrupt queued by step "4" on page 303 is reflected to X, indicating that the connection is complete, and the communication path is available for use. IUCV places the external interrupt information in the buffer that X provided in step "1" on page 303. IUCV passes control to the external interrupt handler of X.
6. Virtual machine X issues a SEND. The SEND function queues an external interrupt for Y indicating that a message is pending. Control returns in X at the next instruction after the SEND.
7. If virtual machine Y is enabled for external interrupts and for IUCV messages (through SET MASK), the external interrupt queued by step "6" on page 303 is reflected to Y, indicating that a message is pending. IUCV places external interrupt information in the buffer specified in step "1" on page 303. IUCV passes control to the external interrupt handler of Y. If virtual machine Y is disabled for external interrupts or IUCV messages and invokes the DESCRIBE function, IUCV places the message information in the DESCRIBE parameter list, and the Message Pending external interrupt for this message is cleared. IUCV passes control to the next instruction after the DESCRIBE.
8. While virtual machine Y is processing the message, virtual machine X can decide to check if the communication has been completed by issuing the TEST COMPLETION function. The condition code indicates that (in this example) the communication is not complete.
9. With the message description from step "7" on page 303, virtual machine Y starts processing the message and issues a RECEIVE. The parameter list associated with RECEIVE specifies where the message data is stored in virtual machine Y.

   If the message was one-way, the RECEIVE function queues an external interrupt for X indicating that the message had completed. REPLY processing in step "10" on page 303 would not be required for one-way messages. Control returns to Y at the next instruction after the RECEIVE.
10. When processing the message is complete, virtual machine Y responds to X by invoking the REPLY function. The REPLY function queues an external interrupt for X indicating that the message has completed. Control returns to Y at the next instruction after the REPLY.
11. If virtual machine X is both enabled for external interrupts and enabled for IUCV replies, the external interrupt queued by step "10" on page 303 is reflected to X, indicating a reply pending. To identify the reply, the external interrupt information is placed in the buffer specified in step "1" on page 303. IUCV passes control to the external interrupt handler of X. If virtual machine X is disabled for external interrupts and issues a TEST COMPLETION, IUCV places the message information in the TEST COMPLETION parameter list, and the Message Completion external interrupt is cleared. IUCV passes control to the next instruction after the TEST COMPLETION.
12. Virtual machine X has now completed its communications with virtual machine Y and issues a SEVER to break the communications path. The SEVER function queues an external interrupt for Y indicating that the communication link has been broken. Control returns in X at the next instruction after the SEVER.
13. The external interrupt queued by step "12" on page 303 is reflected to Y indicating that the path has been broken by virtual machine X. Virtual machine Y can now do any cleanup needed in its storage.
14. After virtual machine Y has completed processing, the virtual machine issues a SEVER notifying IUCV that it also is finished with the communication path. IUCV can then clean up its control blocks.

15. When all communications are complete and all communication paths have been severed, both virtual machines independently invoke the RETRIEVE BUFFER function.

Using Data in a Parameter List

Most IUCV functions require a **parameter list** which contains information necessary for IUCV to perform the requested function. The IUCV macro assists you in filling in the parameter list properly. The parameters used with each function are described with the individual function descriptions later in this chapter.

The parameters let you specify 8 bytes of data in the parameter list. To understand better how data specified in the parameter list is handled, the IUCV functions are covered in a typical scenario.

1. The IUCV DECLARE BUFFER, CONNECT, and ACCEPT sequence must be invoked to establish the user's external interrupt buffer and a path to the target virtual machine (or CP). If you expect to receive data in the parameter list, you must authorize such communication on the CONNECT or ACCEPT by specifying PRMDATA=YES. The external interrupt information to the target communicator includes a bit indicating if PRMDATA=YES was chosen.
2. Issue an IUCV SEND request. When the data is to be passed in the parameter list, the DATA=PRMMSG option is used on the IUCV macro, and the PRMMSG= option is used to move the data into the parameter list. The sender of the message should be prepared to handle a return code indicating that DATA=PRMMSG is not allowed if the target communicator has not specified PRMDATA=YES at connection time. IUCV saves the message data until it is to be presented to the target.
3. If the target is enabled for IUCV Message Pending external interrupts, the target virtual machine receives an IUCV Message Pending external interrupt because of the SEND request in the previous step. The message data is stored in the external interrupt buffer. A flag is set in the IPFLAGS1 field of the buffer (in the IPARML DSECT) indicating that the data is in the parameter list. Since the message data has been presented to the target, the target does not have to issue an IUCV RECEIVE for this message. If the message was a one-way message, communication is complete. There is no asynchronous return of message completion given to the source (sending) virtual machine on a one-way message.
4. If the target is disabled for IUCV Message Pending external interrupts and issues the IUCV DESCRIBE or RECEIVE functions, the message data is stored in the parameter list. A flag is set in the IPFLAGS1 field of the parameter list (in the IPARML DSECT) indicating that the data is in the parameter list. Since the message data is presented to the target on a DESCRIBE, the target does not have to issue an IUCV RECEIVE for this message. If the message was a one-way message, the communication is complete. There is no asynchronous return of message completion given to the source (sending) virtual machine on a one-way message.
5. If the communication in the previous steps was a two-way message, a REPLY is issued by the target virtual machine. When the REPLY data is to be passed in the parameter list, the DATA=PRMMSG option is used on the IUCV macro, and the PRMMSG= option is used to move the data into the parameter list. The REPLYer of the message should be prepared to handle a return code indicating that DATA=PRMMSG is not allowed if the source communicator has not specified PRMDATA=YES at connection time. IUCV saves the message data until it is to be presented to the source communicator.
6. If the source communicator is enabled for IUCV Message Completion external interrupts, the source virtual machine receives an IUCV Message Completion external interrupt because of the REPLY in the previous step. The message data is stored in the external interrupt buffer. A flag is set in the IPFLAGS1 field of the buffer indicating that the data is in the parameter list. The communication is complete.
7. If the target is disabled for IUCV Message Completion external interrupts, and issues the IUCV TEST COMPLETE function, the message data is stored in the parameter list. A flag is set in the IPFLAGS1 field of the parameter list indicating that the data is in the parameter list. The communication is complete.
8. SEVER and RETRIEVE BUFFER cause any messages pending to be destroyed for that virtual machine. Since no asynchronous Message Completion external interrupt is returned to the source communicator for one-way messages using the DATA=PRMMSG option, the source communicator

must realize upon receiving an IUCV Connection Severed external interrupt from the target communicator that messages may not have been received by the target.

Using Control Paths

IUCV control paths and buffers allow a control program (like CMS) running in a virtual machine to use the IUCV functions without interfering with a user application that is also using IUCV. Applications would not be coded using the CONTROL parameter on the IUCV DECLARE BUFFER, IUCV CONNECT, and IUCV ACCEPT functions.

To understand better how control paths would be handled, the IUCV functions are covered in a typical user scenario. In the scenario, CMS is used as the control program running in a virtual machine executing a normal IUCV application.

1. When CMS is IPLed in the virtual machine, CMS issues an IUCV DECLARE BUFFER with the CONTROL=YES parameter. This establishes a control buffer for CMS to use. All IUCV external interrupt information for control paths is presented in this buffer.
2. After CMS has defined a control buffer, CMS may establish control paths to other virtual machines by issuing an IUCV CONNECT with the CONTROL=YES parameter. All paths used by CMS should be specified as control paths.
3. If the target virtual machine accepts the connection request, an IUCV Connection Complete external interrupt is presented to the CMS control program. The IPCNTRL bit in IPFLAGS1 of the external interrupt (in the IPARML DSECT) indicates that a control path was accepted. CMS may now start communications on this path.
4. Just as it can establish control paths through IUCV CONNECT, CMS can also accept connections and specify that they be managed as control paths. CMS does this by issuing IUCV ACCEPT with the CONTROL=YES parameter. All paths accepted by CMS should be accepted as control paths.
5. When CMS allows the application program to run, the application issues an IUCV DECLARE BUFFER with the CONTROL=NO parameter. All application paths are established using IUCV CONNECT with the CONTROL=NO parameter. These are the functions that the application uses today so no changes are required to the application.
6. The application starts communicating over its established paths.
7. Since both CMS and its application have established paths, both are expecting and handling external interrupts.

   If an external interrupt is on a control path, the IUCV information about the interrupt is stored in the control buffer when the interrupt is presented to CMS. CMS interrogates the control buffer, recognizes the path ID as belonging to a control path, and handles the IUCV interrupt. The application's buffer remains unchanged.

   If the external interrupt is on an application path, the IUCV information about the interrupt is stored in the application's buffer when the interrupt is presented to CMS. Since CMS only has access to the control buffer, IUCV stores the path ID in the control buffer and clears (to 0) the remainder of the buffer. CMS interrogates the control buffer, recognizes the path ID as belonging to an application path, and passes the IUCV external interrupt to the application for handling.
8. When the application wishes to terminate a path that it established or to terminate all IUCV communications, it uses the IUCV SEVER or RETRIEVE BUFFER functions. Neither of these functions affects the control paths being used by CMS.

Certain IUCV functions result in an operation exception if executed with only a control buffer declared. These functions are:

- DESCRIBE
- INTERRUPT POLL
- RETRIEVE BUFFER
- SET CONTROL MASK
- SET MASK

- TEST COMPLETION
- TEST MESSAGE.

The ALL=YES parameter on the IUCV functions of SEVER, QUIESCE, and RESUME does not affect control paths.

When handling IUCV messages with the IUCV functions of RECEIVE, REPLY, REJECT, and PURGE on control paths, the message must be fully qualified. The message ID, path ID, and class of the message must be specified in the parameter list to reference the message.

The IUCV functions affecting IUCV external interrupts do not operate on interrupts for control paths. These functions are TEST MESSAGE, DESCRIBE, INTERRUPT POLL, TEST COMPLETION, SET MASK, and SET CONTROL MASK. These functions are never used by a control program since they have no affect on control paths.

Since IUCV cannot tell what part of the virtual machine issued an IUCV function, it is possible for an application to issue an IUCV function on a control path. This reference to a control path by an application, whether intentional or accidental, is considered a user application error. For example, the SEVER function specifying a control path terminates that path even though the function was issued by the application program.

Invoking IUCV Functions

You can invoke all IUCV functions through the IUCV macro. In general, specify the name of the IUCV function you wish to perform, the address of a parameter list to contain input to the function, and keyword parameters. IUCV moves the values specified on the keyword parameters into the specified parameter list.

The parameter list must be defined on a doubleword boundary.

You can specify IUCV parameters in two ways:

- By coding keyword parameters on the IUCV macro. IUCV stores values in the parameter list based on values you specify on the macro.
- By storing required input to the function in the function parameter list before invoking the IUCV macro. To store input in an IUCV parameter list, use labels generated by the IPARML DSECT in HCPGPI MACLIB.

You may use a combination of these methods to supply input to a single IUCV function. If you specify any optional parameters on the IUCV macro, you are responsible for providing the USING for the IPARML DSECT when the macro is invoked. If you do not specify an optional parameter to initialize the parameter list, the macro assumes that you have stored a value in the parameter list before invoking the IUCV macro.

Note: The IUCV macro does NOT clear parameter list fields since values may have been stored by the user already. Therefore, it is the user's responsibility to insure that all unused fields are cleared (set to 0). All reserved fields in the parameter list should always be set to 0.

An advantage of using the IUCV macro is that IUCV provides extensive error checking of parameter combinations when input is supplied on the macro. Many invalid parameter combinations can be detected by IUCV when you assemble the program.

For more information on the CMS IUCV applications, see *z/VM: CMS Application Development Guide for Assembler* and *z/VM: CMS Macros and Functions Reference*.

General Description of IUCV Functions

In the description of IUCV functions the following terms are used:

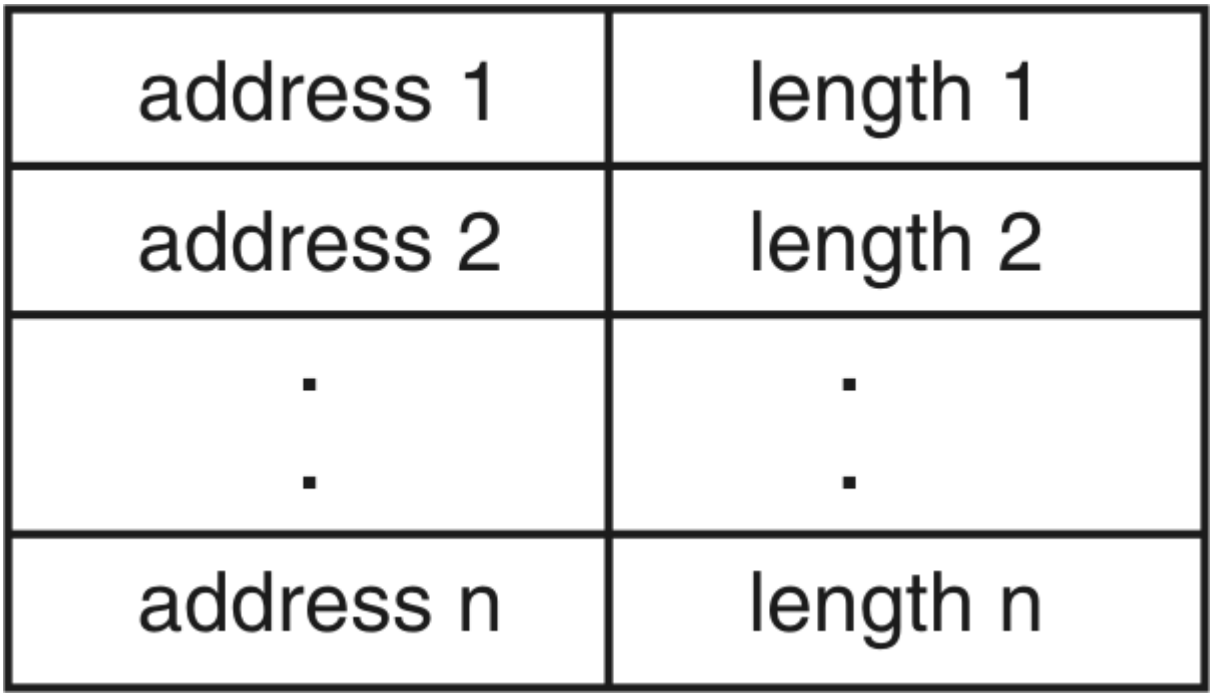
Address
A guest real address (real to the virtual machine). It can be specified on the IUCV macro in one of the following ways:

- *Label* of the storage location
- Number of a register in parentheses that contains the address, (*reg*).

Every address field in the IUCV parameter list is a 4-byte reserved field. The address of the parameter list must be a guest real address, and it must be on a doubleword boundary.

Specify the address of the parameter list as a relocatable label or the number of a register that contains the address.

Address List
A virtual machine defined area used on an IUCV SEND, RECEIVE, or REPLY that allows data to be moved from discontiguous areas. The address list must be on a doubleword boundary in the following format:



Each entry contains two fullwords; the guest real address of the data to be transferred and the number of bytes to be transferred from that address.

Label
An addressable label in the user's program. The IPARML DSECT provides common labels for referencing fields in an IUCV parameter list.

Length
The amount of data to be transferred on an IUCV request. It can be specified on the IUCV macro in one of the following ways:

- *Label* of the storage location containing the length
- Number of a register that contains the length, (*reg*).

The IUCV macro assumes a halfword value for the length at the storage location, or the low-order halfword of the register specified. A length modifier of 2 or 4 may be used, (*label*,2) or ((*reg*),2), or (*label*,4) or ((*reg*),4). If a length modifier of 4 is used, the macro uses the fullword value for the length at the storage location or in the register specified.

The descriptions of the IUCV functions are presented in alphabetic order in Chapter 5, “IUCV Function Descriptions,” on page 317:

ACCEPT — Complete a path
CONNECT — Establish a path

DECLARE BUFFER* — Initialize for IUCV communications
DESCRIBE* — Avoid Message pending interrupt
IPOLL* — Check for pending replies or incoming messages
PURGE — Cancel a message
QUERY — Get IUCV information
QUIESCE — Suspend message pending interrupts
RECEIVE — Receive a message
REJECT — Refuse a message
REPLY — Respond to a message
RESUME — Restore message pending interrupts
RETRIEVE BUFFER* — Terminate all IUCV communications
SEND — Transmit a message
SET CONTROL MASK* — Disable all IUCV control interrupts
SET MASK* — Disable all types of IUCV interrupts
SEVER — Terminate a path
TEST COMPLETION* — Avoid Message complete interrupt
TEST MESSAGE* — Check for interrupts or wait.

*These functions have different meanings in a virtual MP environment. For more information about a virtual MP environment, see the following section on "Virtual MP Considerations for IUCV Applications".

Virtual MP Considerations for IUCV Applications

IUCV applications can be written to work in a virtual MP environment. The following list is intended to provide some guidance on using IUCV in a virtual MP environment.

- IUCV functions may be invoked by any virtual processor in the virtual configuration as long as one of the processors has issued a DECLARE BUFFER function.
- The DECLARE BUFFER function defines an interrupt buffer for the virtual processor that invokes it.
- In the virtual MP environment, IUCV interrupts are treated as "floating" interrupts. Any virtual processor that has:
  - issued an IUCV Declare Buffer
  - enabled to receive IUCV interrupts with the CR0 setting
  - enabled for IUCV interrupts with the SETMASK and SETCMASK functions

  may receive an IUCV interrupt.
- The IUCV RETRIEVE BUFFER function will only retrieve the buffer for the currently running virtual processor. IUCV paths will not be SEVERed until the last virtual processor issues a RETRIEVE BUFFER.
- The SETMASK and SETCMASK functions will apply only to the virtual processor on which they are invoked. This will allow an application to force different types of IUCV interrupts to different virtual processors in the complex, if so desired.
- The following are associated with the virtual configuration:
  - IUCV directory specifications
  - IUCV paths
  - IUCV interrupts
  - IUCV messages.
- The following are associated with the virtual CPU:
  - the application buffer, the control buffer, and the interrupt buffer extension
  - interrupt enablement masks in the virtual PSW and virtual control register 0 (bit 30)
  - the interrupt enablement masks of SETMASK and SETCMASK.

- The DESCRIBE, TEST COMPLETE, and IPOLL functions will complete on any processor in the virtual complex as long as one virtual processor has issued a DECLARE BUFFER (it does not have to be the virtual processor that issued the DESCRIBE, TEST COMPLETE, or IPOLL function).
- If multiple virtual processors in the complex issue the TEST MESSAGE function, it is unpredictable in which order the virtual processors will be taken out of their wait states.
- All addresses specified with IUCV parameter lists are guest absolute addresses.
- Without appropriate guest operating system support, it is difficult or impossible to use IUCV in a virtual MP environment. This support would allow your application to:
  - declare buffers on different processors
  - enable for IUCV interrupts on the needed processors
  - handle the interrupts and route them to the appropriate virtual processor

  Note that CMS does not currently support IUCV virtual MP functions.

IUCV in a Distributed Environment

Distributed IUCV is supported across a Communication Services (CS) collection. Participating systems must include the DISTRIBUTE IUCV statement in their SYSTEM CONFIG files. Depending on what you specify, distributed IUCV will either:

- attempt to satisfy a CONNECT on the local system, then attempt to locate the target on a system within the CS collection. The only exceptions being if the application specifies that the connect must be satisfied either locally or on a particular target system.
- or, be supported only when an application explicitly specifies the target system for a CONNECT.

IUCV applications will behave the same in a distributed environment as they do on a local system with the following exceptions:

- PURGE and REJECT will only be honored on the local system. Once a message is sent to the other system it is considered to be delivered.
- The PRIORITY and MSGLIMIT directory specifications must be present on both systems if they are to be honored.
- The default maximum data length is 16M per message. The maximum can be altered via the SYSTEM CONFIG file by using the DISTRIBUTE statement.

As with distributed APPC/VM through ISFC, distributed IUCV will SEVER both sides of the conversation if it receives an IPRCODE or IPAUDIT code on function completion. Because ISFC is a transport mechanism, it can not rely on information received on a function that completes with an error and because ISFC is unsure of the disposition of the conversation, it SEVERs the path. This is the normal procedure for an application that receives an IUCV error.

At this time, no system services are able to operate in a distributed IUCV environment.

See *z/VM: CP Planning and Administration* for information on the DISTRIBUTE IUCV system configuration statement.

Chapter 4. IUCV Protocols

The following protocols are defined for IUCV data communication:

CONNECT/ACCEPT
Establishes a path between two communicators

SEND/RECEIVE
Sends one-way messages with data in one or more buffers

SEND/RECEIVE/REPLY
Sends two-way messages with data in one or more buffers

SEND
Sends one-way messages with data in a parameter list

SEND/REPLY
Sends two-way messages with data in a parameter list.

IUCV CONNECT/ACCEPT Protocol: A virtual machine uses the CONNECT/ACCEPT protocol (shown in Figure 17 on page 311) to establish a connection with another virtual machine or with a CP system service. The IUCV parameter list specified on the CONNECT request indicates the user ID of the intended target virtual machine or CP system service and, optionally:

- A message limit that indicates the number of outstanding messages permitted for the path
- The path's ability to handle priority communications
- Whether data is to be sent in a parameter list or in one or more buffers
- A user doubleword of data.

The condition code and the return code for the CONNECT request indicate to the source virtual machine the success or failure of the sending of the request. IUCV also returns the (IUCV-assigned) path ID and message limit to the source virtual machine in the IUCV parameter list.

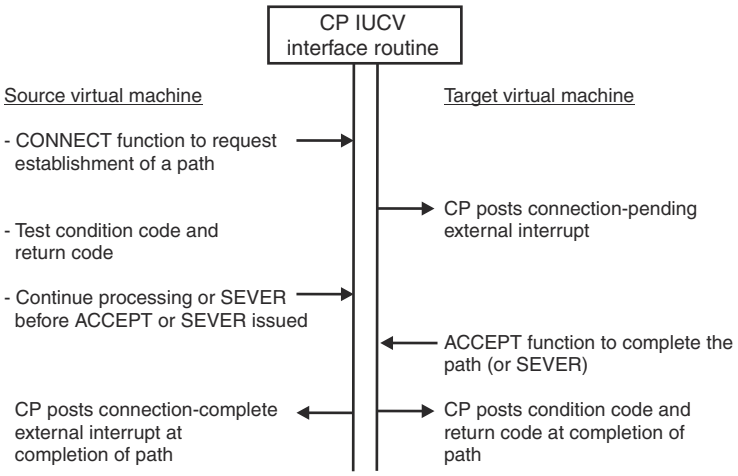


Figure 17. Flow of the IUCV CONNECT/ACCEPT Protocol

The target communicator specified in a successful CONNECT request has a connection-pending external interrupt queued for it by CP. Identification of the request (the source virtual machine's path ID and user ID) is placed in the external interrupt buffer that is stored in the target virtual machine's virtual storage at the time the target virtual machine receives the connection-pending external interrupt. The target virtual machine can then send an ACCEPT function that specifies the path ID of the path to be accepted and optionally modifies the characteristics of the path, such as its message limit and priority handling.

If the ACCEPT function is incorrectly specified, the target virtual machine is notified immediately by way of a condition code and return code and the acceptance of the path does not take place. If correctly

© Copyright IBM Corp. 1991, 2020

specified, the ACCEPT function causes CP to establish the path. When the path is established, CP notifies the target virtual machine by way of a condition code and return code for the ACCEPT function that indicates a successful or unsuccessful operation.

The source virtual machine is notified of the completion of its CONNECT request by way of a connection-complete external interrupt. The external interrupt buffer identifies the completed request by way of its path ID, indicates the path's characteristics, and optionally, contains a user doubleword of data specified by the target virtual machine.

If the target virtual machine to which a CONNECT request is directed does not wish to establish the path, it can enter a SEVER function that specifies the path being rejected. CP then posts a connection-severed external interrupt for the source virtual machine. The external interrupt buffer indicates the path ID of the severed path, and optionally, a user doubleword of data specified by the target virtual machine. The source virtual machine can also sever a path by using the SEVER function.

IUCV SEND/RECEIVE Protocol: The SEND/RECEIVE protocol defines a one-way transfer of data from virtual storage of the source virtual machine to virtual storage of the target virtual machine, as shown in Figure 18 on page 312. The IUCV parameter list specified on the SEND request indicates:

- Path ID of the path on which the message is to be sent
- This is a one-way message
- Data is in one or more buffers
- Length of the data
- Address of the data or the address of a list of buffers that contain the data in the source virtual machine.

Optionally, the priority of the message, the source message class, the target message class, and a message tag may be specified. The condition code and the return code for the SEND request indicate to the source virtual machine the success or failure of the sending of the request. IUCV also returns the (IUCV-assigned) message ID to the source virtual machine in the IUCV parameter list. The source virtual machine can then continue with other processing while the data transfer operation takes place.

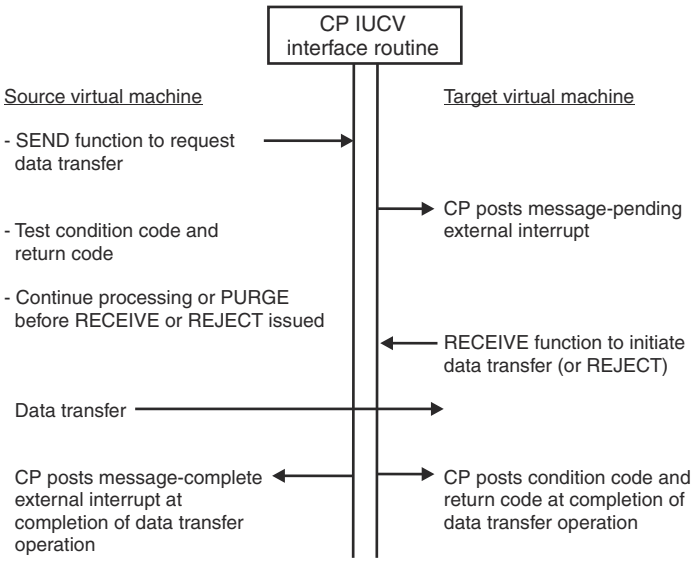


Figure 18. Flow of the IUCV SEND/RECEIVE Protocol

The target communicator specified in a successful SEND request has a message-pending external interrupt queued for it by CP. Identification of the request (path ID and message ID) and data length are placed in the external interrupt buffer that is stored in the target virtual machine's virtual storage when the target virtual machine receives the message-pending external interrupt. The target virtual machine can then send a RECEIVE function that specifies the path ID and message ID the request to be received and the address and length of the buffer in its own virtual storage in which the data to be received is to be placed. If more than one buffer is required, the address of a list of discontiguous buffers may be specified

instead. Use of the message ID in the RECEIVE function enables a virtual machine with more than one data transfer request queued to process them in the order desired.

If the RECEIVE function is incorrectly specified, the target virtual machine is notified immediately by way of a condition code and a return code. The requested data transfer does not take place. If correctly specified, the RECEIVE function causes CP to begin transferring the data from virtual storage of the source to virtual storage of the target virtual machine. When the data transfer is completed, CP notifies the target virtual machine by way of a condition code and a return code for the RECEIVE function that indicates a successful or unsuccessful operation.

The source virtual machine is notified of the completion of its SEND request by way of a message-complete external interrupt. The external interrupt buffer:

- Indicates whether the data transfer occurred successfully (by way of the audit trail bits)
- Identifies the completed request (by way of its path ID and message ID)
- Contains a residual count for a partial transfer operation (as a result of a data transfer error).

If the target virtual machine to which a SEND request is directed does not wish to receive the data, it can issue a REJECT function that specifies the request being rejected. The external interrupt buffer stored for the message-complete external interrupt for the source virtual machine indicates that the request was rejected by way of an audit trail bit. The source virtual machine can purge a SEND request using the PURGE function.

IUCV SEND/RECEIVE/REPLY Protocol: The SEND/RECEIVE/REPLY protocol provides the means for a virtual machine to perform a send and receive operation using a single request. That is, while data is being transferred from the source virtual machine to the target virtual machine, data can also be transferred from the target virtual machine to the source virtual machine.

As shown in Figure 19 on page 314, the source virtual machine issues a SEND function to initiate the two-way transfer request. The IUCV parameter list of the SEND function specifies the following:

- The path ID of the path on which the message is to be sent
- That this is a two-way message
- That the data is in one or more buffers
- The address and length of the data to be sent (or list of buffers that contain the data) in the source virtual machine
- The address and length of the reply area (or list of reply buffers) in the source virtual machine.

Optionally, the priority of the message, the source message class, the target message class, and a message tag may be specified.

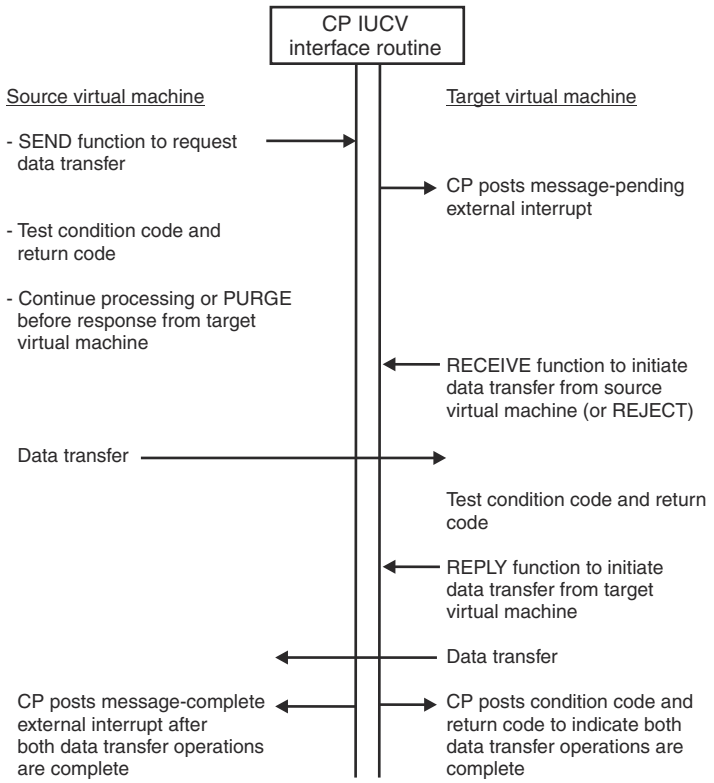


Figure 19. Flow of the IUCV SEND/RECEIVE/REPLY Protocol

The condition code and the return code for the SEND request indicate whether the request was sent successfully. IUCV also returns the (IUCV-assigned) message ID to the source virtual machine in the IUCV parameter list and processing in the source virtual machine continues. A message-pending external interrupt is queued for the target virtual machine by CP. When the target virtual machine receives the interrupt, the request is identified in the external interrupt buffer by its path ID and message ID. The length and location of the buffer in the source virtual machine that is to receive data sent by the target virtual machine by way of a REPLY (or the address of a list of buffers) is also stored.

After receiving the message-pending external interrupt, the target virtual machine can respond with a RECEIVE or REJECT function, with parameters specified as for a SEND/RECEIVE transaction. The REJECT function causes the entire SEND/RECEIVE/REPLY transaction to be cancelled. The RECEIVE function causes a data transfer from the source virtual machine to the target virtual machine, as for a SEND/RECEIVE transaction. However, after the RECEIVE function is sent, the target virtual machine receives control to send a REPLY function that specifies the path ID and message ID for this REPLY request, and the address and length of the data to be sent to the source virtual machine. This causes CP to start transferring data from the target virtual machine to the source virtual machine. The data is placed in the reply buffers in the source virtual machine that were specified in the SEND function.

When both data transfer operations have completed, the source virtual machine receives a message-complete external interrupt and the target virtual machine receives a condition code and return code in response to the REPLY function. The source virtual machine can use the REPLY data length field in the stored external interrupt buffer to determine the amount of data sent by the target virtual machine.

A SEND/RECEIVE/REPLY request can be terminated by the source virtual machine by way of a PURGE function.

IUCV SEND Protocol: The SEND protocol defines a one-way transfer of data from a source virtual machine to a target virtual machine without the use of the RECEIVE function, as shown in Figure 20 on page 315. The data that may be transferred in this way is limited to 8 bytes and is stored in the IUCV parameter list. The path to be used must be authorized to handle data in the parameter list during the CONNECT/ACCEPT protocol that establishes it. If a SEND is directed to a virtual machine that has not authorized receipt of data in a parameter list on this path, it is indicated by a return code to the SEND request and the request is cancelled by CP.

The IUCV parameter list for a SEND request with data in the parameter list specifies the path ID of the path on which the message is to be sent, that this is a one-way message, and that the data is in the parameter list. Optionally, the priority of the message, the source message class, the target message class, and a message tag may be specified. When correctly specified, the SEND causes CP to transfer data from the source to the external interrupt buffer of the target virtual machine. After the data transfer is complete, a message-pending external interrupt is queued for the target virtual machine. Note that no message-complete external interrupt is queued for the source virtual machine.

A SEND request can be purged by the source virtual machine through the PURGE function. The target virtual machine can reject a SEND request through the REJECT function.

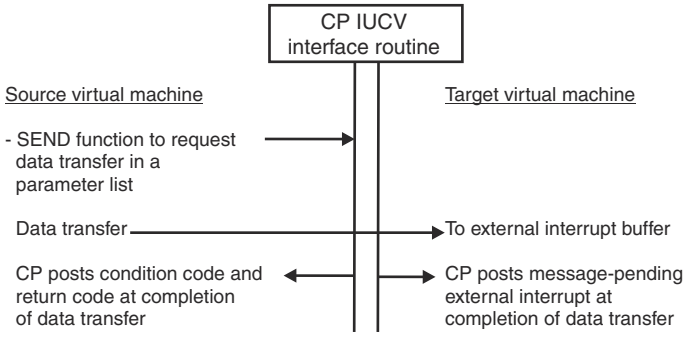


Figure 20. Flow of the IUCV SEND Protocol

IUCV SEND/REPLY Protocol: The SEND/REPLY protocol defines a two-way transfer of data from a source virtual machine to a target virtual machine without the use of the RECEIVE function, as shown in Figure 21 on page 315. The data that may be transferred in this way is limited to 8 bytes and is stored in the IUCV parameter list. The path to be used must be authorized to handle data in the parameter list during the CONNECT/ACCEPT protocol that establishes it. If a SEND/REPLY request is directed to a virtual machine that has not authorized receipt of data in a parameter list on this path, it is indicated by a return code to the SEND/REPLY request and the request is cancelled by CP.

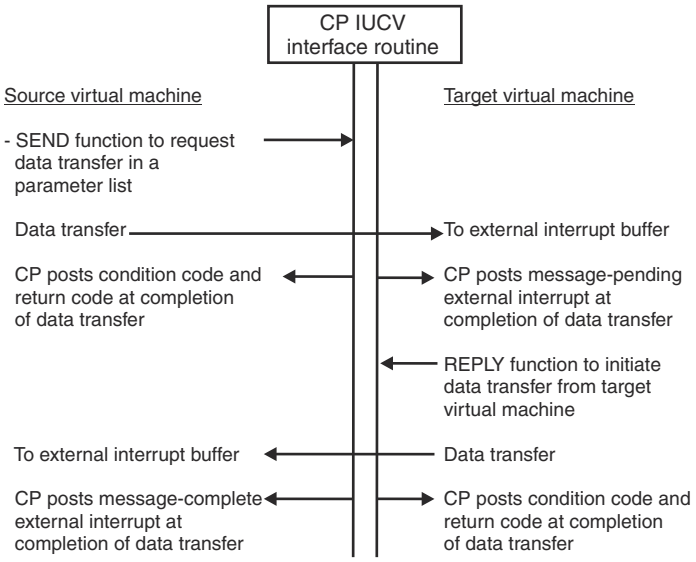


Figure 21. Flow of the IUCV SEND/REPLY Protocol

The IUCV parameter list for a SEND/REPLY request with data in the parameter list specifies the path ID of the path on which the message is to be sent, that this is a two-way message, and that the data is in the parameter list. Optionally, the priority of the message, the source message class, the target message class, and a message tag may be specified. When correctly specified, the SEND causes CP to transfer data from the source virtual machine to the external interrupt buffer of the target virtual machine. After the data transfer is complete, a message-pending external interrupt is queued for the target virtual machine.

Note that no message-complete external interrupt is queued for the source virtual machine after the SEND.

After receiving the message-pending external interrupt, the target virtual machine receives control to send a REPLY function that specifies the path ID and message ID for this REPLY request, and that the reply is in the parameter list. CP transfers the data from the target virtual machine's parameter list to the external interrupt buffer of the source virtual machine. When the data transfer from the REPLY is complete, the source virtual machine receives a message-complete external interrupt and the target virtual machine receives a condition code and return code in response to the REPLY function.

The target virtual machine can also REPLY with data in one or more buffers as is described under the SEND/RECEIVE/REPLY protocol.

A SEND request can be purged by the source virtual machine using the PURGE function. The target virtual machine can reject a SEND request using the REJECT function.

Chapter 5. IUCV Function Descriptions

This chapter contains information on the following IUCV functions and external interrupts:

- Accept and the Connection Complete Interrupt
- Connect and the Connection Pending External Interrupt
- Declare Buffer
- Describe
- Interrupt Poll
- Purge
- Query
- Quiesce and the Connection Quiesced External Interrupt
- Receive
- Reply and the Message Complete External Interrupt
- Resume and the Connection Resumed External Interrupt
- Retrieve Buffer
- Send and the Message Pending External Interrupt
- Set Control Mask
- Set Mask
- Sever and the Connection Severed External Interrupt
- Test Completion
- Test Message.

If you are unfamiliar with reading syntax diagrams, see "Syntax, Message, and Response Conventions" on page xxxv.

CP System Services

IUCV treats communications with CP as if CP were a single virtual machine. IUCV gathers information about a message and routes it to the specified system service for processing.

IUCV provides:

- Routing of connections to CP system services
- Routing of messages to CP system services
- Routing of message completions to the CP system service that issued the SEND
- Severing of connections to CP system services.

Each CP system service that interfaces with virtual machines is uniquely defined to IUCV. The following table shows the corresponding user ID for each of the CP system services. This user ID must be specified on the USERID= parameter when invoking the IUCV CONNECT function.

Note: Use of any IUCV service within a transaction will cause the transaction to abort with either a restricted-instruction transaction-abort code or a transaction-constraint exception. (See z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf) for details on transactional execution.)

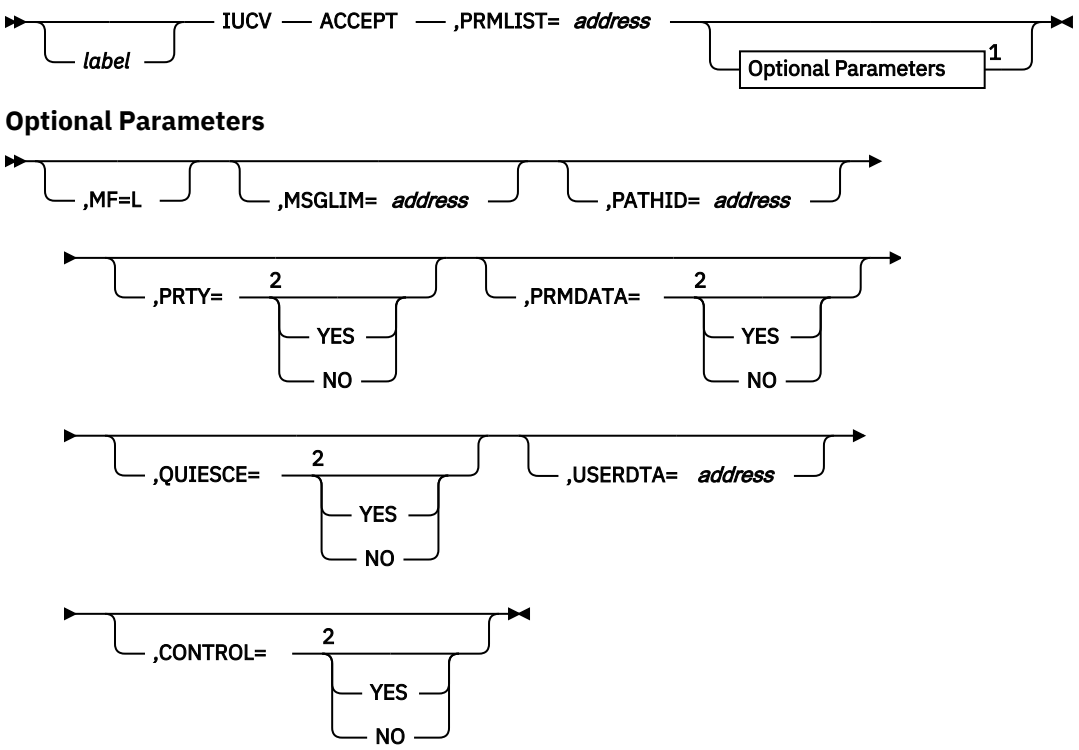
CP system services and their user IDs are as follows:

© Copyright IBM Corp. 1991, 2020

Table 67. CP System Services and Their User IDs

System Service User ID	System Service
*ACCOUNT	Accounting System Service (See page Chapter 12, “Account System Service (*ACCOUNT),” on page 699.)
*ASYNCMD	Asynchronous CP Command Response System Service (See page Chapter 13, “Asynchronous CP Command Response System Service (*ASYNCMD),” on page 719.)
*BLOCKIO	DASD Block I/O System Service (See page Chapter 14, “DASD Block I/O System Service (*BLOCKIO),” on page 721>.)
*IDENT	Identify System Service (See page Chapter 16, “Identify System Service (*IDENT),” on page 731.)
*LOGREC	Error Recording System Service (See page Chapter 15, “Error Logging System Service (*LOGREC),” on page 729.)
*MONITOR	Monitor System Service (See *z/VM: Performance*.)
*MSG	Message System Service (See page Chapter 17, “Message System Service (*MSG),” on page 739.)
*MSGALL	Message All System Service (See page Chapter 18, “Message All System Service (*MSGALL),” on page 741.)
*RPI	Access Verification System Service (See page Chapter 11, “Access Verification System Service (*RPI),” on page 591.)
*SIGNAL	Signal System Service (See page Chapter 20, “Signal System Service (*SIGNAL),” on page 747.)
*SPL	Spool System Service (See page Chapter 21, “Spool System Service (*SPL),” on page 751.)
*SYMPTOM	Symptom System Service (See page Chapter 22, “Symptom System Service (*SYMPTOM),” on page 773.)
*VMEVENT	VM Event System Service (See page Chapter 23, “VM Event System Service (*VMEVENT),” on page 775.)

ACCEPT Function



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The ACCEPT function is issued after the user receives a Connection Pending external interrupt and now wishes to complete the IUCV communication path.

Parameters

Required Parameters:

ACCEPT
Requests that CP perform the IUCV ACCEPT function.

PRMLIST=
Specifies the *address* of the ACCEPT parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

MSGLIM=
Specifies the limit of outstanding messages to be allowed on the path completing this ACCEPT. The *address* of the MSGLIM points to a 2-byte field.

Upon executing the IUCV instruction, the message limit specified is checked to insure that the maximum limit of 65535 has not been exceeded. The actual limit assigned to the invoker's path established by this connection depends on the value specified for MSGLIM and the value specified for MSGLIMIT (if any) on the IUCV control statement of the invoker's directory entry. The IUCV control statement authorizes the invoker to establish the path.

When the ACCEPT function is invoked, the directory entries are searched in a definite order:

1. The invoker's IUCV control statements are searched for an entry for the target's user ID.
2. The invoker's IUCV control statements are searched for an ANY entry.

The first entry found, that applies establishes the message limit for the path according to the following table:

MSGLIM specified on ACCEPT	MSGLIMIT specified on applicable IUCV control statement	Actual message limit
No	No	Default value of 10
No	Yes	Directory entry value
Yes	No	MSGLIM value
Yes	Yes	Lower value of the two specified

After executing the IUCV instruction, the IPMSGLIM field in the output parameter list reflects the actual message limit established for the path.

PATHID=
Specifies the path identification number on which you wish to communicate. This path ID is presented to the virtual machine in the Connection Pending external interrupt.

PRTY=
Specifies if you want to send priority messages on this path. It does not affect the program's ability to receive priority messages.

PRTY=YES indicates that you want to send priority messages. Priority must be authorized in the IUCV directory control statement for this parameter to be effective. After executing the IUCV instruction, the IPPRTY bit in IPFLAGS1 should be checked to insure that priority messages were authorized.

PRTY=NO indicates that you cannot send priority messages.

If your program is unauthorized or if PRTY=NO is specified, IUCV prevents your program from sending priority messages.

PRMDATA=
Specifies whether your program can handle message data in the parameter list.

PRMDATA=YES indicates that your program can handle message data in the parameter list (those messages sent using the parameter DATA=PRMMSG on an IUCV SEND).

PRMDATA=NO indicates that your program can only handle message data presented in a buffer (sent using the parameter DATA=BUFFER on an IUCV SEND).

QUIESCE=
Specifies whether you want to quiesce the path being established.

QUIESCE=YES prevents messages from coming across the path until your program is ready to process them. You can restore the path to full communication by invoking the IUCV RESUME function.

QUIESCE=NO indicates that the path will become active as soon as the IUCV ACCEPT completes.

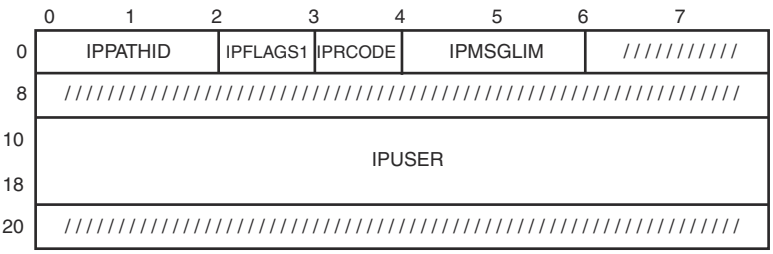
USERDTA=
Specifies the data area containing the 16 bytes of user data that IUCV is to reflect to the source virtual machine. The user data is reflected as part of the IUCV Connection Complete external interrupt.

CONTROL=
Specifies whether this connection is to be associated with the external interrupt buffer for control paths or application paths.

CONTROL=YES indicates that this path is to be associated with the external interrupt buffer for control paths. CONTROL=NO indicates that this path is to be associated with the external interrupt buffer for application paths.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path identification number of the path you are completing.

IPFLAGS1
Contains options for the ACCEPT function.

IPRMDATA (X'80')
Indicates that you are prepared to handle message data in the parameter list.

IPQUSCE (X'40')
Indicates that you do not want to receive messages on this path until an IUCV RESUME is issued.

IPPRTY (X'20')
Indicates that you want to send priority messages on this path.

IPCNTRL (X'04')
Indicates that you want this to be a control path.

IPMSGLIM
Contains the limit of outstanding messages that IUCV is to allow the invoker to send on the path.

IPUSER
Contains the user data that IUCV reflects to the target virtual machine.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored in IPRCODE.

Parameter List Output Fields:

IPMSGLIM
Contains the message limit for this path.

IPFLAGS1
Contains specific information about this connection.

IPPRTY (X'20')
Indicates that you may send priority messages.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Connection is not pending on this path
20 - X'14' - Originator has severed this path
30 - X'1E' - IPAPPC flag in IPFLAGS1 not 0.

Note: If you get a return code that is not documented here, it is an APPC/VM return code. An APPC/VM return code can result if the IPAPPC bit is set on during an IUCV CONNECT. For a description of the APPC/VM return code, refer to "IUCV ACCEPT" on page 524.

Program Exceptions

The program exceptions for IUCV ACCEPT are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

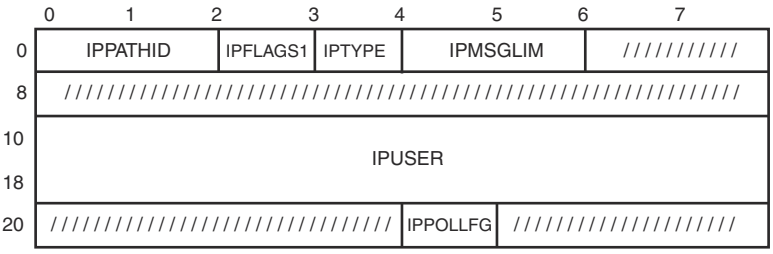
Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

Completion Conditions

Connection Complete External Interrupt: To notify the source virtual machine that you have accepted the connection and completed a new IUCV path, IUCV reflects an IUCV Connection Complete external interrupt to the source virtual machine.

The source virtual machine receives this external interrupt if it is enabled for IUCV interrupts in Control Register 0 and the PSW. The functions of SET MASK and SET CONTROL MASK also control the presentation of this type of interrupt.



IPPATHID
Contains the path ID of the path which has now been established.

IPFLAGS1
Contains options for this path.

IPRMDATA (X'80')
Indicates that the connecting virtual machine can handle message data in the parameter list.

IPQUSCE (X'40')
Indicates that IUCV will not allow messages to be sent on this path until an IUCV RESUME is issued by the connecting virtual machine.

IPPRTY (X'20')
Indicates that the virtual machine may receive priority messages on this path.

IPCNTRL (X'04')
Indicates that this is a control path.

IPTYPE
Indicates a Connection Complete external interrupt with a value of X'02'.

IPMSGLIM
Contains the maximum number of messages that IUCV allows the virtual machine that issued the ACCEPT function to send on this path.

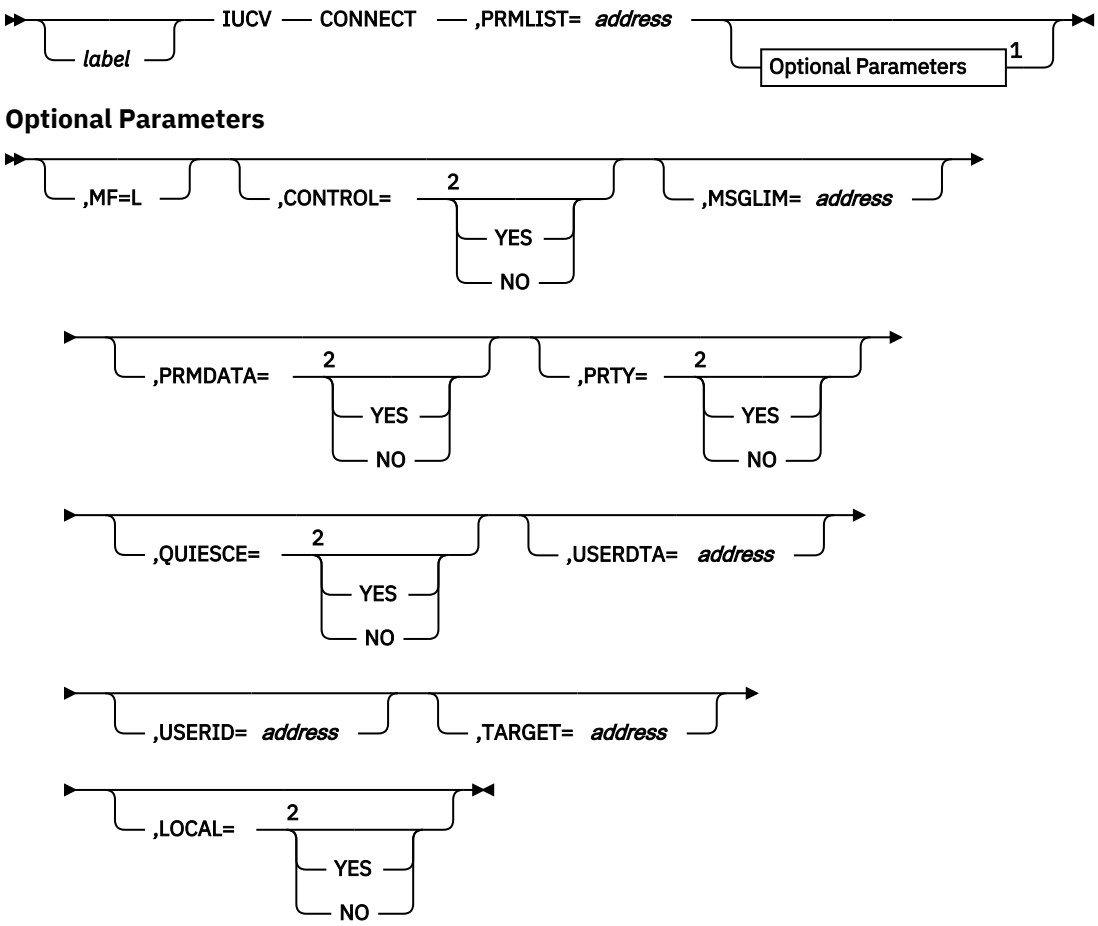
IPUSER
Contains the user data specified by the target virtual machine when it accepted this connection.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

CONNECT Function



Notes:

[1] Optional parameters can be entered in any order.
[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The CONNECT function establishes an IUCV path to another virtual machine. Although the CONNECT may complete successfully, you are not able to use the path until you receive an IUCV Connection Complete external interrupt (the target has accepted your connection) for this path.

If you receive an IUCV Connection Severed external interrupt (the target has severed your connection) for this path, you may not use this path since the connection has been refused by the target virtual machine.

If the CONNECT function completes successfully, the count of active connections is incremented for both virtual machines. If a virtual machine is connecting to itself, the active connection count is incremented by two. The count is not decremented until the virtual machine issues an IUCV SEVER for a particular path.

Note: If an external security manager is installed on your system, you may not be authorized to use this function. For additional information, contact your security administrator.

Parameters

Required Parameters:

CONNECT
Requests that CP perform the IUCV CONNECT function.

PRMLIST=
Specifies the *address* of the CONNECT parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

CONTROL=
Specifies whether this connection is to be associated with the external interrupt buffer for control paths or application paths.

CONTROL=YES indicates that this path is to be associated with the external interrupt buffer for control paths. For a complete discussion on using control paths and buffers, see "Using Control Paths" on page 305.

CONTROL=NO indicates that this path is to be associated with the external interrupt buffer for application paths.

LOCAL=

YES
Allows an application to force the partner to be on the local system. LOCAL=YES and TARGET=*address* cannot be specified together.

NO
Indicates that if the partner is not found on the local system, IUCV may try to locate the partner within the CS collection if DISTRIBUTE IUCV YES is specified in the system configuration file.

MSGLIM=
Specifies the limit of outstanding messages to be allowed on the path established by this CONNECT. The *address* of the MSGLIM points to a 2-byte field.

Upon executing the IUCV instruction, the message limit specified is checked to insure that the maximum limit of 65535 has not been exceeded. The actual limit assigned to the invoker's path established by this connection depends on the value specified for MSGLIM and the value specified for MSGLIMIT (if any) on the IUCV control statement of the invoker's directory entry. The IUCV control statement authorizes the invoker to establish the path.

When the CONNECT function is invoked, the directory entries are searched in a definite order:

1. The invoker's IUCV control statements are searched for an entry for the target's user ID.
2. The invoker's IUCV control statements are searched for an ANY entry.
3. The target's IUCV control statements are searched for an ALLOW entry.

The first entry found, that applies establishes the message limit for the path according to the following table:

MSGLIM specified on CONNECT	MSGLIMIT specified on applicable IUCV control statement	Actual message limit
No	No	Default value of 10
No	Yes	Directory entry value

MSGLIM specified on CONNECT	MSGLIMIT specified on applicable IUCV control statement	Actual message limit
Yes	No	MSGLIM value
Yes	Yes	Lower value of the two specified

After executing the IUCV instruction, the IPMSGLIM field in the output parameter list reflects the actual message limit established for the path.

If the SEND function completes successfully, the count of active messages is incremented for the sending virtual machine. The count is not decremented until the message complete external interrupt is returned. For one-way messages using data in the parameter list, the count is decremented when the message pending external interrupt is presented to the target virtual machine.

PRMDATA=
Specifies whether your program can handle message data in the parameter list.

PRMDATA=YES indicates that your program can handle message data in the parameter list (those messages sent using the parameter DATA=PRMMSG on an IUCV SEND).

PRMDATA=NO indicates that your program can only handle message data presented in a buffer (sent using the parameter DATA=BUFFER on an IUCV SEND).

PRTY=
Specifies if you want to send priority messages on this path. It does not affect the program's ability to receive priority messages.

PRTY=YES indicates that you want to send priority messages. Priority must be authorized on the IUCV directory control statement for this parameter to be effective. After executing the IUCV instruction, the IPPRTY bit in IPFLAGS1 should be checked to insure that priority messages were authorized.

If the IUCV CONNECT is routed via ISFC, PRIORITY status for the CONNECT is determined by the initial CONNECT invoker's IUCV directory control statement.

PRTY=NO indicates that you do not want to send priority messages.

If your program is unauthorized or if PRTY=NO is specified, IUCV prevents your program from sending priority messages.

TARGET=
Specifies which system the target must be on. LOCAL=YES and TARGET=*address* cannot be specified together.

This option allows a remote connection within the CS collection if DISTRIBUTE IUCV YES or TOLERATE has been specified in the system configuration file. If DISTRIBUTE IUCV NO has been specified, a remote connection is allowed only if the source and target node are members of the same SSI cluster.

QUIESCE=
Specifies whether you want to quiesce the path being established.

QUIESCE=YES prevents messages from coming across this path until your program is ready to process them. You can restore the path to full communication by invoking the IUCV RESUME function.

QUIESCE=NO indicates that the path is to become active as soon as the corresponding IUCV ACCEPT is done by the target communicator.

USERDTA=
Specifies the data area containing the 16 bytes of user data that IUCV is to reflect to the target virtual machine. The user data is reflected as part of the IUCV Connection Pending external interrupt.

USERID=
Specifies the 8-character user ID of the target virtual machine or the IUCV system service to which you want to establish this path.

By default USERID is resolved within the local VM system.

When TARGET is specified, USERID is resolved on the designated target VM system. This requires:

- Configure DISTRIBUTE IUCV YES or TOLERATE on both systems. For more information on the SYSTEM CONFIG statement, see *z/VM: CP Planning and Administration*. Note that if the source and target node reside within the same SSI cluster, the DISTRIBUTE IUCV restrictions do not apply.
- Configure ISFC links connecting both VM systems. For more information on the ACTIVATE ISLINK, see *z/VM: CP Commands and Utilities Reference* and *z/VM: CP Planning and Administration*.
- The LOCAL=YES option must not be specified.

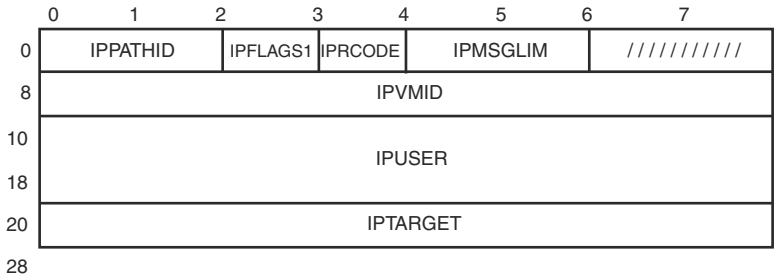
When TARGET is not specified, USERID is resolved according to the following rules:

- If USERID is logged on the local VM system (the one the invoker is on) a local connection is initiated.
- If an ISFC link is active, and DISTRIBUTE IUCV YES is configured, ISFC will perform a search for USERID logged on within the ISFC collection. If USERID is found on a node within the collection, a connection is attempted via ISFC.

  If DISTRIBUTE IUCV NO or TOLERATE is configured and USERID is a single-configuration user online within the same SSI cluster, a remote connection is initiated because the user is configured to operate on any node in the cluster. If DISTRIBUTE IUCV NO or TOLERATE is configured and USERID is a multiconfiguration (IDENTITY) user, the IUCV CONNECT fails without a TARGET node.
- Otherwise, the IUCV CONNECT fails.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPFLAGS1
Contains options for the CONNECT function.

IPRMDATA (X'80')
Indicates that your program can handle message data in the parameter list.

IPQUSCE (X'40')
Indicates that you do not want to receive messages on this path until an IUCV RESUME is issued.

IPPRTY (X'20')
Indicates that you want to send priority messages on this path.

IPAPPC (X'08')
Indicates the protocol to be used on this path. This bit must be set to 0.

IPCNTRL (X'04')
Indicates that you want this to be a control path.

IPLOCAL (X'01')
Indicates that the connect can only be satisfied on the local system

IPMSGLIM
Contains the limit of outstanding messages that IUCV is to allow the invoker to send on the path.

IPVMID
Contains the user ID of the virtual machine or IUCV system service to which you want to establish this path.

IPUSER
Contains the user data that IUCV reflects to the target virtual machine.

IPTARGET
Contains the system name where the connect is to be satisfied.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored in IPRCODE.

Parameter List Output Fields:

IPPATHID
Contains the path ID that IUCV assigns the new path.

IPMSGLIM
Contains the message limit for this path.

IPFLAGS1
Contains specific information about this connection.

IPPRTY (X'20')
Indicates that you may send priority messages on this path.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
11 - X'0B' - Target communicator is not logged on.
Also see note 2 below.
12 - X'0C' - Target communicator has not invoked the DECLARE BUFFER function
13 - X'0D' - Maximum number of connections for this communicator exceeded
14 - X'0E' - Maximum number of connections for the target exceeded
15 - X'0F' - No authorization found (See Note 3.)
16 - X'10' - Invalid IUCV system service name

Notes:

1. If you get a return code that is not documented here, it is an APPC/VM return code. An APPC/VM return code can result if the IPAPPC bit is set on during an IUCV CONNECT. For a description of the APPC/VM return code, refer to "APPCVM CONNECT" on page 412.
2. IPRCODE 11 - X'0B' may also be returned if Distributed IUCV is active (either DISTRIBUTE IUCV TOLERATE or DISTRIBUTE IUCV YES is specified in the SYSTEM CONFIG file for the z/VM system) and one of the following occurs:
   - IPLOCAL is specified but IPTARGET is non-zero.
   - IPLOCAL is specified but the target VM id (IPVMID) is not logged on locally.
   - IPTARGET is blanks.

- IPTARGET specifies a remote system but no ISFC link exists.
- DISTRIBUTE IUCV TOLERATE is specified in SYSTEM CONFIG and IPTARGET is zero but the VM id (IPVMID) is not logged on locally.
- DISTRIBUTE IUCV YES is specified in SYSTEM CONFIG and IPTARGET is zero but the VM id (IPVMID) is not logged on locally or remotely.

3. IPRCODE 15 - X'0F' may also be returned if a problem was detected with internal CP control structures. Consequently, a CP soft abend will be generated.

Program Exceptions

The program exceptions for IUCV CONNECT are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

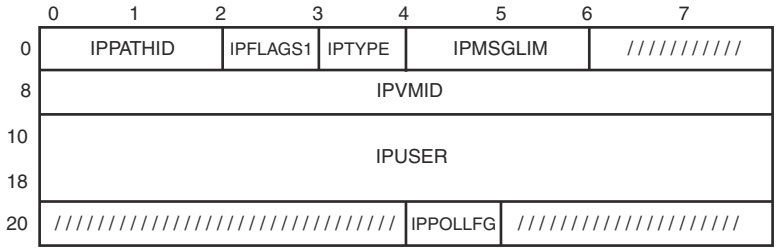
Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

Completion Conditions

Connection Pending External Interrupt: To notify the target virtual machine that you wish to establish a new path (through the CONNECT), IUCV reflects an IUCV Connection Pending external interrupt to the target virtual machine.

The target virtual machine receives this external interrupt if it is enabled for IUCV interrupts in Control Register 0 and the PSW. The functions of SET MASK and SET CONTROL MASK also control the presentation of this type of interrupt.

The external interrupt contains the information that the target virtual machine needs to either ACCEPT or SEVER the pending connection.



IPPATHID
Contains the path ID that IUCV assigns the new path.

IPFLAGS1
Contains options for this path.

IPRMDATA (X'80')
Indicates that the connecting virtual machine can handle message data in the parameter list.

IPQUSCE (X'40')
Indicates that IUCV will not allow messages to be sent on this path until an IUCV RESUME is issued by the connecting virtual machine.

IPPRTY (X'20')
Indicates that the virtual machine may receive priority messages on this path.

IPTYPE
Indicates a Connection Pending external interrupt with a value of X'01'.

IPMSGLIM
Contains the maximum number of messages that IUCV allows the virtual machine that issued the CONNECT function to send on this path.

IPVMID
Contains the user ID of the virtual machine or IUCV system service specified by the virtual machine that wants to establish this path.

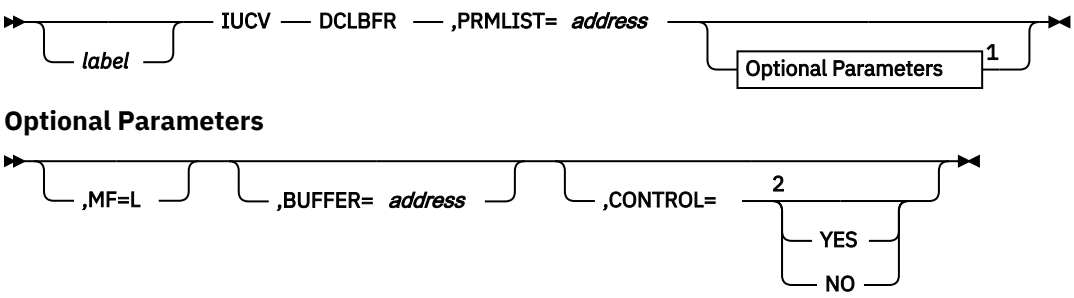
IPUSER
Contains the user data specified by the virtual machine that wants to establish this path.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

DECLARE BUFFER Function



Notes:

[1] Optional parameters can be entered in any order.
[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The DECLARE BUFFER function specifies the guest real address of an external interrupt buffer where IUCV can store information. When a virtual machine receives an IUCV external interruption, IUCV stores in this buffer information about the message, reply, or control function that caused the interruption.

The DECLARE BUFFER function must be invoked before any other IUCV function can be used (except QUERY).

After invoking the DECLARE BUFFER function and if enabled for IUCV interrupts, the virtual machine can now start receiving IUCV external interrupts.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see "Virtual MP Considerations for IUCV Applications" on page 308.

Parameters

Required Parameters:

DCLBFR
requests that CP perform the IUCV DECLARE BUFFER function.

PRMLIST=
specifies the *address* of the DECLARE BUFFER parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

BUFFER=
specifies the *address* of the external interrupt buffer.

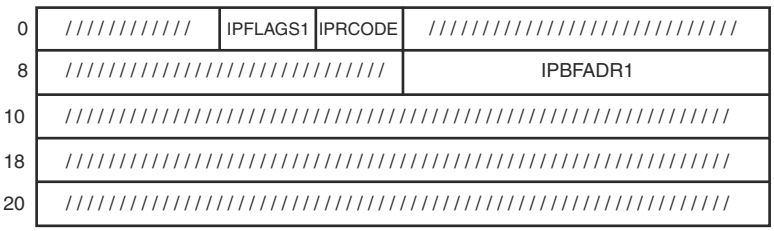
CONTROL=
specifies whether this buffer is to be used with control paths or application paths.

CONTROL=YES indicates that this buffer is to be used with control paths. For a complete discussion on using control buffers and paths, see "Using Control Paths" on page 305.

CONTROL=NO indicates that this buffer is to be used with application paths.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPFLAGS1
contains options for the DECLARE BUFFER function.

IPCNTRL (X'04')
indicates that you want this to be a control buffer.

IPBFADR1
contains the address of your external interrupt buffer.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE
3 - Errors encountered in reading directory.

Parameter List Output Fields:

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
10 - X'0A' - Invalid length for the interrupt buffer extension.
19 - X'13' - A previously declared buffer is still in use
or an IUCV RETRIEVE BUFFER is in progress.
62 - X'3E' - Two of the following buffers overlap:
- Control buffer
- External interrupt buffer
- Interrupt buffer extension

92 - X'5C' - A paging or storage error was detected.

Program Exceptions

The program exceptions for IUCV DECLARE BUFFER are:

Specification Exception
The parameter list is not on a doubleword boundary.

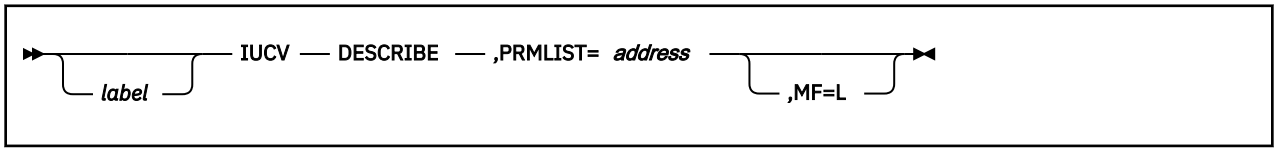
Addressing Exception
The parameter list or buffer address that you specified is outside the virtual machine's storage.

Operation Exception
Your virtual machine is not in supervisor state.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

The buffer is outside the user's address space when CP checks to see that the user is entitled to access the area of space assigned to the buffer.

DESCRIBE Function



Purpose

The DESCRIBE function determines whether you have a message pending for your virtual machine. If there is a message pending, information about the message is returned in the parameter list. Since you now have the message information, there is no need for IUCV to reflect an IUCV Message Pending external interrupt and you do not receive one for this message.

Since IUCV normally informs you of the message by reflecting a Message Pending external interrupt, you should not use the DESCRIBE function unless you have disabled for this type of interrupt. The IUCV SET MASK function can be used to disable your virtual machine for Message Pending external interrupts.

A message is described only once, either in the parameter list of a DESCRIBE or by a Message Pending external interrupt.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Parameters

Required Parameters:

DESCRIBE
Requests that CP perform the IUCV DESCRIBE function.

PRMLIST=
Specifies the *address* of the DESCRIBE parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameter: If you do not specify this parameter, the macro assumes that you have stored the desired value into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

Parameter List Format:

IPARML DSECT

	0	1	2	3	4	5	6	7
0	IPPATHID		IPFLAGS1	IPRCODE	IPMSGID			
8	IPTRGCLS				IPRMMSG1			
10	IPBFLN1F / IPRMMSG2				//////////////////////////////			
18	//							
20	IPBFLN2F				IPPOLLFG	//////////////////		

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
2 - No message found.

Parameter List Output Fields:

IPPATHID
Contains the path on which the message was sent.

IPFLAGS1
Contains specific information about the message.

IPRMDATA (X'80')
Indicates that the 8-byte message is in the parameter list at IPRMMSG.

IPPRTY (X'20')
Indicates that this is a priority message.

IPNORPY (X'10')
Indicates that this is a one-way message and no REPLY is expected.

IPFGMID (X'04')
Is always set to 1 indicating that the message ID has been stored at IPMSGID.

IPFGPID (X'02')
Is always set to 1 indicating that the path ID has been stored at IPPATHID.

IPFGMCL (X'01')
Is always set to 1 indicating that the target message class has been stored at IPTRGCLS.

IPMSGID
Contains the message ID.

IPTRGCLS
Contains the target message class.

IPRMMSG1/IPRMMSG2
Contains the message when it is stored in the parameter list (indicated by IPRMDATA in IPFLAGS1). The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPBFLN1F
Contains the length of the message.

IPBFLN2F
Contains the length of the maximum expected reply.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that another iteration of this function will probably not find a message waiting at this time.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return.

Program Exceptions

The program exceptions for IUCV DESCRIBE are:

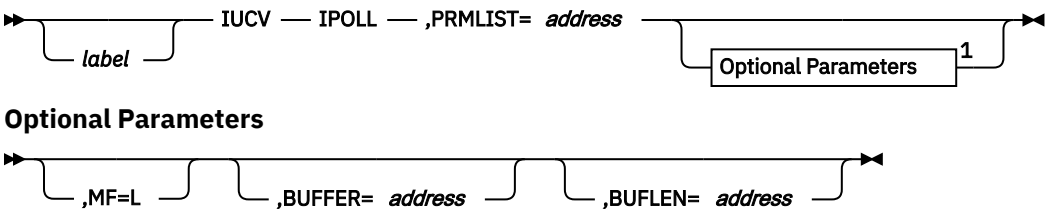
Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

INTERRUPT POLL Function



Notes:

[1] Optional parameters can be entered in any order.

Purpose

The INTERRUPT POLL (IPOLL) function determines whether you have any replies or incoming messages pending. If IUCV finds any replies or incoming messages pending, it returns the information about them in the buffer provided. The maximum number of pending interrupts that can be retrieved on a single request is the number of IUCV external interrupt buffers which can fit on one 4K page.

Notes:

1. Unless you disable your virtual machine for IUCV message-complete and message-pending interrupts, you should not use the INTERRUPT POLL function. When the virtual machine is enabled for these interrupts, IUCV automatically informs you of message completion or arrival of an incoming message by reflecting an external interrupt to your virtual machine.
2. No external interrupt will occur for a reply represented by a message-complete returned by the INTERRUPT POLL function.
3. No external interrupt will occur for a message represented by a message-pending returned by the INTERRUPT POLL function. It is your responsibility to use the RECEIVE or REJECT function to process a message obtained using the INTERRUPT POLL function.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Parameters

Required Parameters:

IPOLL
Requests that CP perform the IUCV INTERRUPT POLL function.

PRMLIST=
Specifies the *address* of the INTERRUPT POLL parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

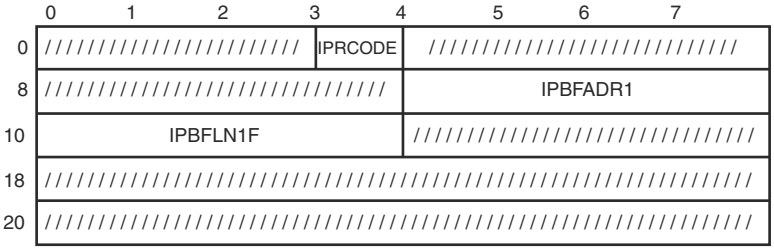
MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

BUFFER=
Specifies the address of the INTERRUPT POLL buffer for interrupt data.

BUFLEN=
Specifies the length of the INTERRUPT POLL buffer for interrupt data. This length must be at least the size of an IUCV interrupt buffer and not more than 4096 bytes, and the buffer may not cross a 4K page boundary. The length need not be an exact multiple of the length of an IPARML.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPBFADR1
contains the address of the input buffer.

IPBFLN1F
contains the length of the input buffer.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE
2 - No message found.

Parameter List Output Fields:

IPBFLN1F
contains the length of the output data returned by the INTERRUPT POLL function. This value will always be a multiple of the length of an IUCV external interrupt buffer (IPARML).

IPRCODE
contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
92 - X'5C' - A paging or storage error was detected.

Program Exceptions

The program exceptions for IUCV INTERRUPT POLL are:

Specification Exception
The parameter list is not on a doubleword boundary.

The buffer length specified is less than the size of an IUCV interrupt buffer.

The buffer specified spans a 4K page boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

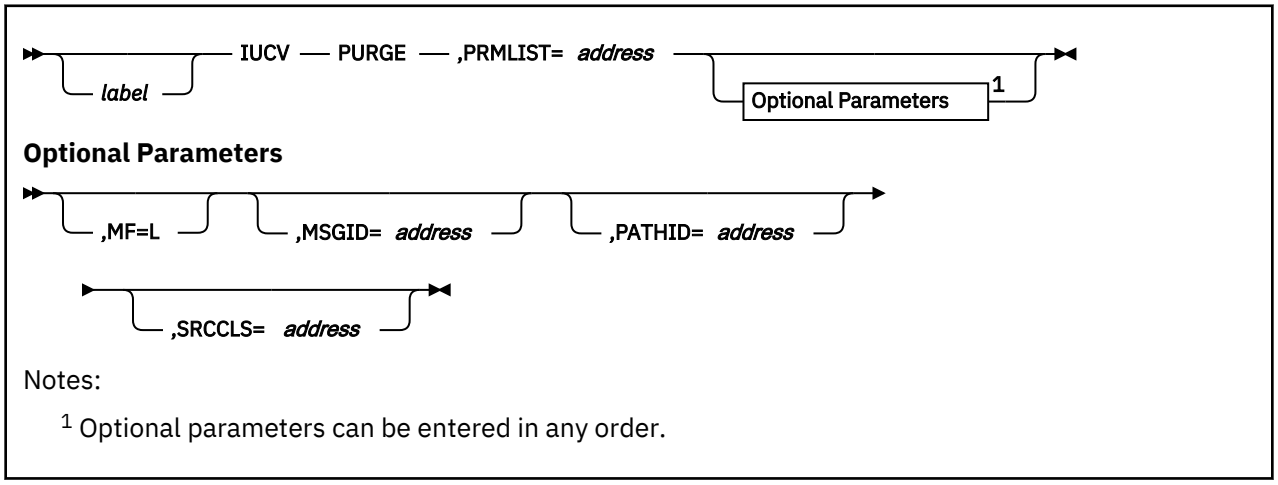
Completion Conditions

Output Buffer Format: When the condition code is zero, the buffer contains one or more interrupt data areas for replies and messages. See Message Complete External Interrupt, and Message Pending External Interrupt. The remainder of the buffer not occupied by the external interrupt data remains unchanged.

If no more replies or messages are pending for the invoker, the last interrupt placed in the output buffer will have the IPNOPOLL flag set.

If INTERRUPT POLL is issued in an APPC/VM environment you may receive interrupt information both for IUCV and APPC/VM paths.

PURGE Function



Notes:

[1] Optional parameters can be entered in any order.

Purpose

The PURGE function cancels a message that you have sent. When you purge a message, one of the following actions takes place:

- If you purge a message before the target virtual machine has received an IUCV Message Pending external interrupt for this message, the target virtual machine is never aware that you sent the message.
- If you purge a message after the target virtual machine has received the IUCV Message Pending external interrupt, but before the target has completed handling the message, the target receives a return code indicating that the message has been purged the next time it references the message (normally, on a RECEIVE or a REPLY). The target is given only one such indication about the purged message. Any future references to the purged message results in a *no message found* condition.
- If you purge a message on which the target virtual machine has already completed its processing, the IUCV Message Complete external interrupt is avoided, but the target virtual machine is never aware that the message was purged.

When purging a message, you can completely identify the message by specifying the message ID, path ID, and source message class. You can also identify the message by the path ID with or without the message class.

Parameters

Required Parameters:

PURGE
Requests that CP perform the IUCV PURGE function.

PRMLIST=
Specifies the *address* of the PURGE parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

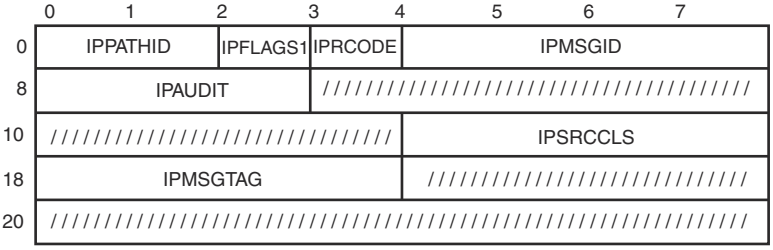
MSGID=
Specifies the message ID of the message to be purged. If the message ID is used to identify the message, the path ID and the source class must also be correctly specified in the parameter list.

PATHID=
Specifies the path ID on which the message was sent. The *address* of the PATHID is a halfword value.

SRCCLS=
Specifies the source message class associated with a message.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path ID of the message you are purging.

IPFLAGS1
Specifies options for the PURGE function.

IPFGMID (X'04')
Indicates that you have specified a message ID (IPMSGID) for the message you are purging.

IPFGPID (X'02')
Indicates that you have specified a path ID (IPPATHID) for the message you are purging.

IPFGMCL (X'01')
Indicates that you have specified a source message class (IPSRCCLS) to identify the message you are trying to purge.

IPMSGID
Contains the message ID of the message you are purging.

IPSRCCLS
Contains the source message class of the message you are purging.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE
2 - No message found.

Parameter List Output Fields:

IPPATHID
Contains the path ID of the message you purged.

IPFLAGS1
Contains specific information about the message purged.

IPNORPY (X'10')
Indicates that the message purged is a one-way message.

IPPRTY (X'20')
Indicates that the message purged is a priority message.

IPMSGID
Contains the message ID of the message you purged.

IPAUDIT
Contains information about possible asynchronous error conditions which may have affected the normal completion of this message. If this field is 0, the message has completed successfully.

The meanings of the bits in the audit trail are:

IPADRPLE (X'800000')
Reply too long for buffer

IPADSNPX (X'400000')
Protection exception on send buffer

IPADSNAX (X'200000')
Addressing exception on send buffer

IPADANPX (X'100000')
Protection exception on answer buffer

IPADANAX (X'080000')
Addressing exception on answer buffer

IPADRJCT (X'040000')
Message was rejected

IPADPRMD (X'020000')
Reply specified DATA=PRMMSG, but this path cannot handle data in the parameter list.

IPADPGNR (X'010000')
Message purged on send or receive queue.

IPADRCPX (X'008000')
Protection exception on receive buffer

IPADRCAX (X'004000')
Addressing exception on receive buffer

IPADRPPX (X'002000')
Protection exception on reply buffer

IPADRPAX (X'001000')
Addressing exception on reply buffer

IPADSVRD (X'000800')
Path was severed

IPADRLST (X'000400')
Invalid RECEIVE or REPLY address list

(X'000200')
Reserved

(X'000100')
Reserved

IPADBLEN (X'000080')
Bad length in SEND buffer list

IPADALEN (X'000040')
Bad length in SEND answer list

IPADBTOT (X'000020')
Invalid total SEND buffer length

IPADATOT (X'000010')
Invalid total SEND answer length

(X'000008')
Reserved

(X'000004')
Reserved

(X'000002')
Reserved

(X'000001')
Reserved.

IPSRCCLS
Contains the message class of the message you purged.

IPMSGTAG
Contains the message tag of the message you purged.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
8 - X'08' - Message found but message class invalid
31 - X'1F' - IUCV function specified on an APPC/VM path.

Program Exceptions

The program exceptions for IUCV PURGE are:

Specification Exception
The parameter list is not on a doubleword boundary, or the message ID was specified without the path ID or the message class.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

QUERY Function

Purpose

The QUERY function determines how large an external interrupt buffer IUCV requires to store information, and determines the maximum number of communication paths you can establish in your virtual machine.

QUERY can be issued before DECLARE BUFFER to determine the buffer size and allocate the buffer before it is declared to IUCV. The maximum number of paths facilitates the allocation of a user-defined path table. This function is useful to virtual machines that have dynamic storage allocations routines. For those programs that must allocate fixed storage, the buffer size is 40 bytes (X'28'), and the maximum number of paths available would be the default of four or the number on the user's OPTION directory control statement, the MAXCONN option.

Parameters

Required Parameter:

QUERY
Requests that CP perform the IUCV QUERY function.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
2 - Error - IUCV RETRIEVE BUFFER in progress.
3 - Errors were encountered reading directory.

Output from QUERY: When you invoke the QUERY function, IUCV returns:

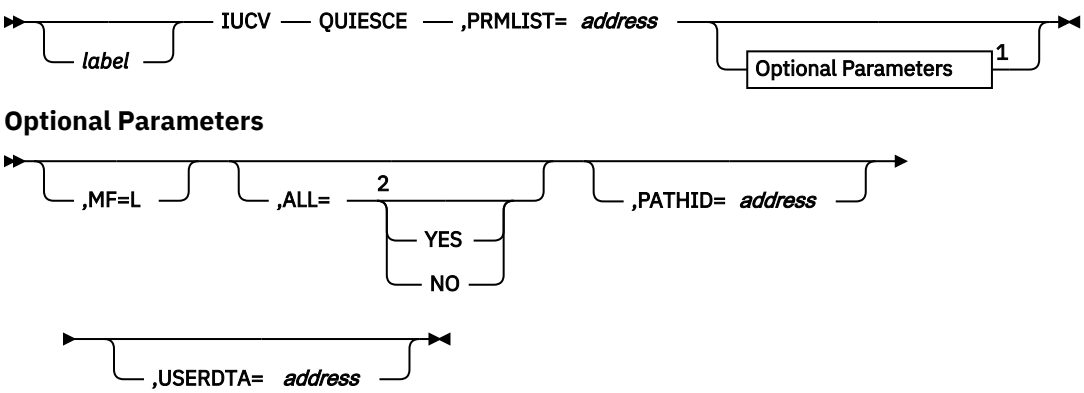
- The size of the IUCV external interrupt buffer in general register 0.
- The maximum number of connections that can be outstanding for this virtual machine in general register 1.

Program Exceptions

The program exceptions for IUCV QUERY are:

Operation Exception
Your virtual machine is not in supervisor state.

QUIESCE Function



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The QUIESCE function temporarily suspends incoming messages on an IUCV path. You can later reactivate the path by invoking the RESUME function or you may leave the path quiesced, making it a one-way path. You are still allowed to send messages as the path is only quiesced for incoming messages.

If a message is sent to you on a quiesced path, a return code is returned to the sender, and the message is not sent. Each end of a path can be quiesced independently.

Parameters

Required Parameters:

QUIESCE
Requests that CP perform the IUCV QUIESCE function.

PRMLIST=
Specifies the *address* of the QUIESCE parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

ALL=
Specifies whether all paths for this virtual machine are to be quiesced.

ALL=YES indicates that all of your paths are to be quiesced.

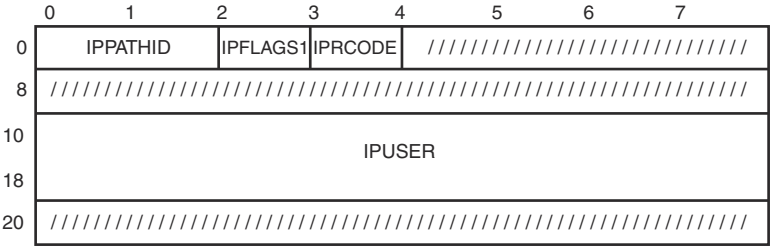
ALL=NO indicates that you do not want all of your paths quiesced, only the one specified by PATHID.

PATHID=
Specifies the path ID of the path you want to quiesce.

USERDTA=
Specifies the data area containing the 16 bytes of user data to be reflected across the path. The user data is reflected as part of the IUCV Connection Quiesced external interrupt.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path ID of the path you are quiescing.

IPFLAGS1
Contains options for the QUIESCE function.

IPALL (X'80')
Indicates that you want to quiesce all paths for this virtual machine.

IPUSER
Contains the user data that is reflected across the path.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE

Parameter List Output Fields:

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
31 - X'1F' - IUCV function specified on an APPC/VM path.
48 - X'30' - Partner system service does not support this function.

Program Exceptions

The program exceptions for IUCV QUIESCE are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

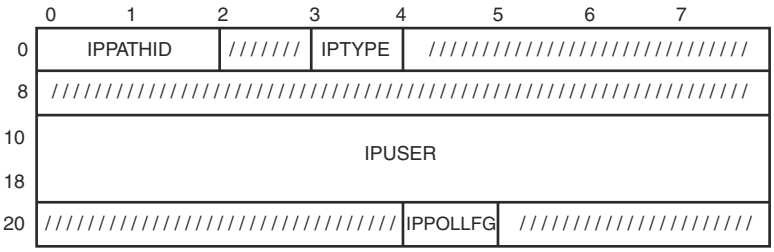
Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

Completion Conditions

Connection Quiesced External Interrupt: To notify the other side of the path that the path has been quiesced, IUCV reflects an IUCV Connection Quiesced external interrupt.

The target virtual machine receives this external interrupt if it is enabled for IUCV interrupts in Control Register 0 and the PSW. The functions of SET MASK and SET CONTROL MASK also control the presentation of this type of interrupt.



IPPATHID
Contains the path ID of the path quiesced.

IPTYPE
Indicates a Connection Quiesced external interrupt with a value of X'04'.

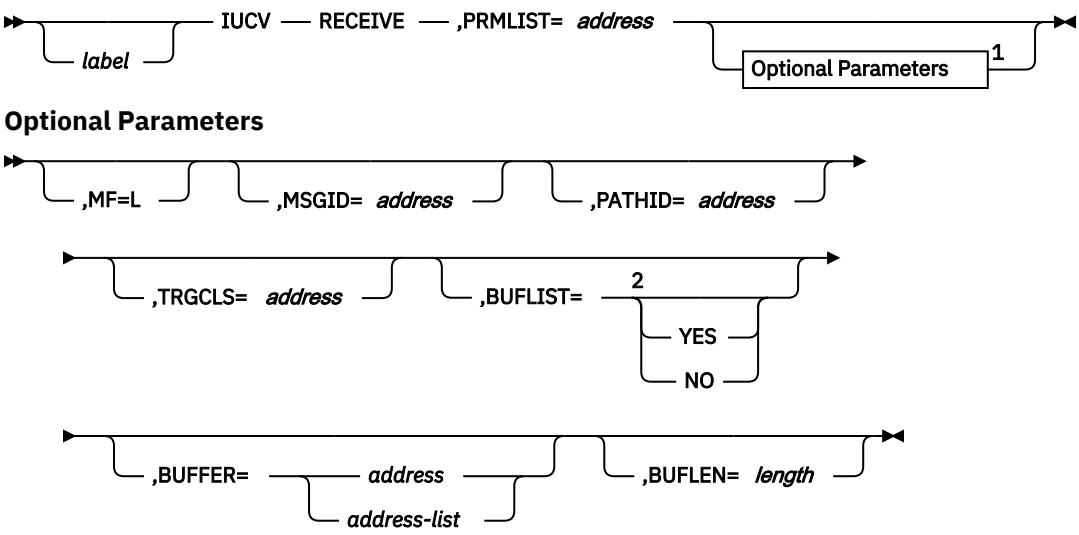
IPUSER
Contains the user data specified by the virtual machine that quiesced the path.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

RECEIVE Function



Notes:

[1] Optional parameters can be entered in any order.
[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The RECEIVE function receives messages that are being sent to you over established paths. The RECEIVE function moves the message data from the source virtual machine to your virtual machine.

When receiving a message, you can completely identify the message by specifying the message ID, path ID, and target class. You can also identify the message by either the path ID or the target class, or both. If you do not specify any identifiers when invoking the RECEIVE function, you receive the first message that has not been partially received.

If your receive area cannot contain the complete message, you can issue another RECEIVE to obtain the remainder of the message. If you use the same parameter list on the subsequent RECEIVE, the message ID, path ID, and message class that are already stored in the parameter list completely identify the message. These fields are required on any subsequent RECEIVEs for the same message. You must initialize the receive area and length for the subsequent RECEIVEs.

The RECEIVE function completes a one-way communication when all the data has been received.

Parameters

Required Parameters:

RECEIVE
Requests that CP perform the IUCV RECEIVE function.

PRMLIST=
Specifies the *address* of the RECEIVE parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

MSGID=
Specifies the message ID of the message to be received. If the message ID is used to locate the message, the path ID and the target class must also be correctly specified in the parameter list.

PATHID=
Specifies the path over which you wish to receive the message. The *address* of the PATHID is to a halfword value.

TRGCLS=
Specifies the target message class associated with this message.

BUFLIST=
Specifies that the list format is being used.

BUFLIST=NO indicates that the list format is not being used. The BUFFER parameter is the address of the complete message.

BUFLIST=YES indicates that the address on the BUFFER parameter identifies the address of a list of addresses and lengths of discontiguous buffers that hold the message text.

BUFFER=
Specifies the *address* or the *list* of addresses into which IUCV moves the message.

BUFLEN=
Specifies the total length of the message to RECEIVE. If BUFFER specifies an address list (BUFLIST=YES), the value specified with BUFLEN is the total of the individual buffer lengths in the list.

Parameter List Format:

IPARML DSECT

	0	1	2	3	4	5	6	7
0	IPPATHID		IPFLAGS1	IPRCODE	IPMSGID			
8	IPTRGCLS				IPBFADR1 / IPRMMSG1			
10	IPBFLN1F / IPRMMSG2				//////////////////////////////			
18	//							
20	IPBFLN2F				//////////////////////////////			

Parameter List Input Fields:

IPPATHID
Contains the path ID of the path to receive the message.

IPFLAGS1
Contains options for the RECEIVE function.

IPBUFLST (X'40')
Indicates that you are using an address list for the message data.

IPAPPC (X'08')
Indicates the protocol to be used on this path. This bit must be set to 0.

IPFGMID (X'04')
Indicates that you have specified a message ID (IPMSGID) for the messages you are trying to receive.

IPFGPID (X'02')
Indicates that you have specified a path ID (IPPATHID) for the message you are trying to receive.

IPFGMCL (X'01')
Indicates that you have specified a target message class (IPTRGCLS) to identify the message you are trying to receive.

IPMSGID
Contains the message ID of the message you are trying to receive.

IPTRGCLS
Contains the target message class of the message you are trying to receive.

IPBFADR1
Contains the address of the receive buffer.

IPBFLN1F
Contains the length of the receive buffer. Use this label with a fullword value. Use IPBFLN1 with a halfword value.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE
2 - No message found.

Parameter List Output Fields:

IPPATHID
Contains the path ID of the message you received.

IPFLAGS1
Contains specific information about the message received.

IPRMDATA (X'80')
Indicates that the 8-byte message is contained in the parameter list at IPRMMSG.

IPPRTY (X'20')
Indicates that this is a priority message.

IPNORPY (X'10')
Indicates that this is a one-way message and no reply is expected.

IPFGMID (X'04')
Is always set to 1 indicating that the message ID has been stored at IPMSGID.

IPFGPID (X'02')
Is always set to 1 indicating that the path ID has been stored at IPPATHID.

IPFGMCL (X'01')
Is always set to 1 indicating that the target message class has been stored at IPTRGCLS.

IPMSGID
Contains the message ID.

IPTRGCLS
Contains the target message class.

IPBFADR1
If BUFLIST=NO, contains the address of the buffer updated by the number of bytes you have received. If BUFLIST=YES, the address points to the current list entry IUCV is working on.

IPRMMSG1/IPRMMSG2
Contains the message when it is stored in the parameter list (indicated by IPRMDATA in IPFLAGS1). The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPBFLN1F
Contains one of the following values, if the receive buffer is:

The same length as the message, this field contains 0.
Longer than the message, this field contains the number of bytes remaining in the buffer.
Shorter than the message, this field contains a residual count (that is, the number of bytes remaining in the message that does not fit into the buffer).

IPBFLN2F
Contains the length of the expected reply. Use this label with a fullword value. Use IPBFLN2 with a halfword value.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
5 - X'05' - Receive buffer too short to contain message
6 - X'06' - Fetch protection exception on send buffer
7 - X'07' - Addressing exception on the send buffer
8 - X'08' - Message ID found but message class or path ID invalid
9 - X'09' - Message has been purged
10 - X'0A' - Message length is negative
22 - X'16' - Send buffer list invalid
23 - X'17' - Negative length in buffer list
24 - X'18' - Incorrect total length of buffer list lengths
26 - X'1A' - Buffer list not on a doubleword boundary
30 - X'1E' - IPAPPC flag in IPFLAGS1 not 0
31 - X'1F' - IUCV function specified on an APPC/VM path
91 - X'5B' - A paging or storage error was detected in the SEND data area
92 - X'5C' - A paging or storage error was detected in the RECEIVE data area.

Note: If you get a return code that is not documented here, it is an APPC/VM return code. An APPC/VM return code can result if the IPAPPC bit is set on during an IUCV CONNECT. For a description of the APPC/VM return code, refer to “APPCVM RECEIVE” on page 451.

Program Exceptions

The program exceptions for IUCV RECEIVE are:

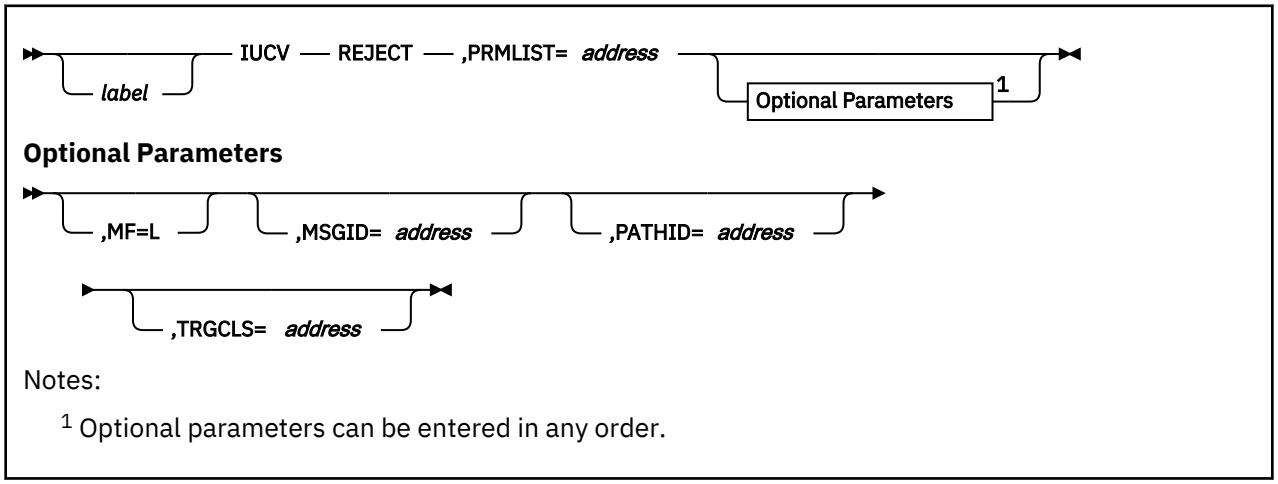
Specification Exception
The parameter list is not on a doubleword boundary, or the message ID was specified without the path ID and the message class.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list, the buffer address, or the buffer list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address, buffer list address, or buffer address in the parameter list or the buffer list does not match the key of the user.

REJECT Function



Notes:

[1] Optional parameters can be entered in any order.

Purpose

The REJECT function refuses a specified message. Between the time that you are notified of a message and the time that you complete the message, the message may be rejected.

When a message is rejected, an IUCV Message Complete external interrupt is reflected to the source virtual machine with an indication in the audit trail (IPAUDIT) that the message was rejected. Depending on when the message was rejected and the type of message, message data may or may not have been moved. When a message is rejected, the sender has no way to determine if any data has been transmitted.

When rejecting a message, you can completely identify the message by specifying the message ID, path ID, and target message class. You can also identify the message by either the path ID or the target message class, or both.

Parameters

Required Parameters:

REJECT
Requests that CP perform the IUCV REJECT function.

PRMLIST=
Specifies the *address* of the REJECT parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

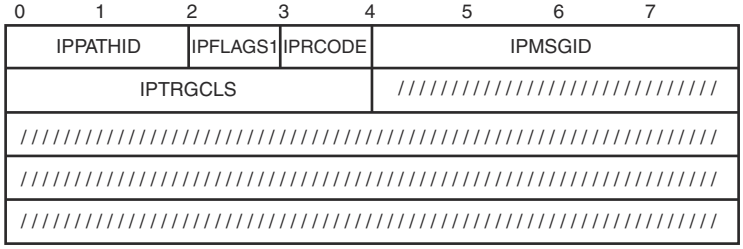
MSGID=
Specifies the message ID of the message to be rejected. If the message ID is used to locate the message, the path ID and the target class must also be correctly specified in the parameter list.

PATHID=
Specifies the path ID of the message to be rejected.

TRGCLS=
Specifies the target message class of the message to be rejected.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path ID of the message you are rejecting.

IPFLAGS1
Contains options for the REJECT function.

IPFGMID (X'04')
Indicates that you have specified a message ID (IPMSGID) for the message you are rejecting.

IPFGPID (X'02')
Indicates that you have specified a path ID (IPPATHID) for the message you are rejecting.

IPFGMCL (X'01')
Indicates that you have specified a target message class (IPTRGCLS) for the message you are rejecting.

IPMSGID
Contains the message ID of the message you are rejecting.

IPTRGCLS
Contains the target message class of the message you are rejecting.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored in IPRCODE
2 - No message found.

Parameter List Output Fields:

IPPATHID
Contains the path ID of the message you rejected.

IPMSGID
Contains the message ID of the message you rejected.

IPTRGCLS
Contains the target message class of the message you rejected.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
8 - X'08' - Message ID found but message class or path ID invalid
31 - X'1F' - IUCV function specified on an APPC/VM path.

Program Exceptions

The program exceptions for IUCV REJECT are:

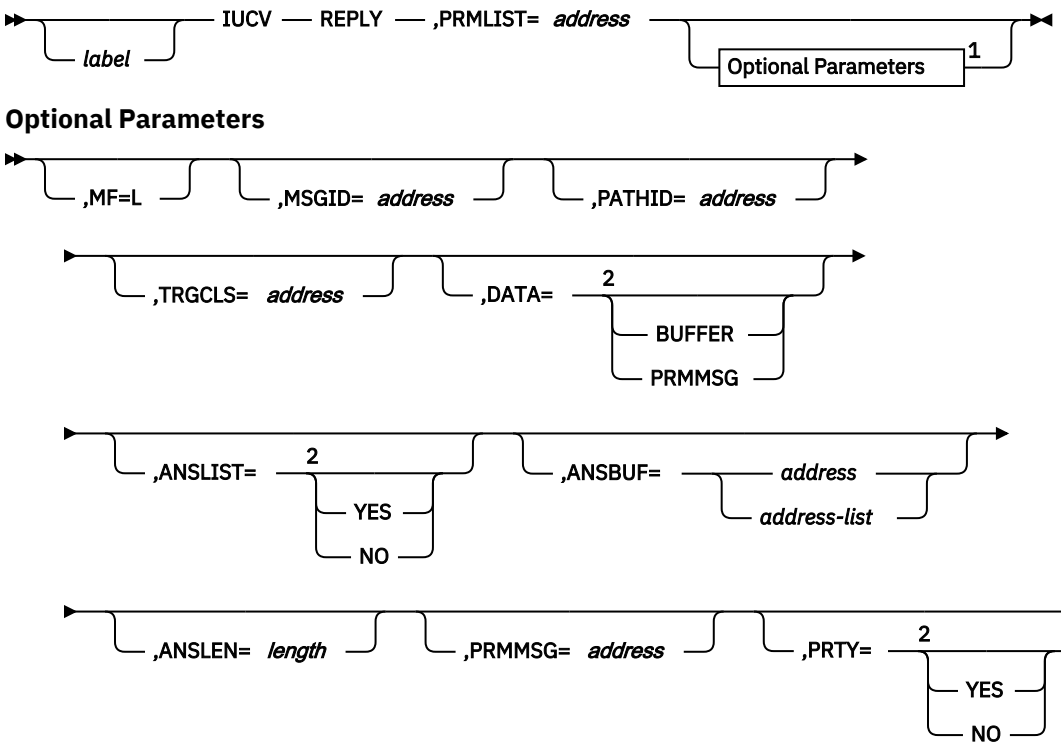
Specification Exception
The parameter list is not on a doubleword boundary, or the message ID was specified without the path ID and the message class.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

REPLY Function



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The REPLY function responds to the two-way messages that you receive. The previous IUCV functions will have identified the message to which you are replying by providing the message ID, the path ID, and the target class. You must identify completely the message to which you wish to reply.

The REPLY function moves the reply data from your virtual machine to the source virtual machine.

Parameters

Required Parameters:

REPLY
Requests that CP perform the IUCV REPLY function.

PRMLIST=
Specifies the *address* of the REPLY parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

MSGID=
Specifies the message ID of the message to which you are replying.

PATHID=
Specifies the path associated with the message.

TRGCLS=
Specifies the target message class associated with the message.

With the reply function you have an option of replying to the message in a buffer or in the parameter list. The size of a parameter list message is very limited, but then the originator of the message (sender) does not have to maintain an answer buffer (ANSBUF parameter on SEND). The protocol you chose may be different from that used to send you the message.

DATA=
Specifies the location of your message data for this IUCV communication.

DATA=BUFFER indicates that your reply data is in a buffer. You can use the ANSBUF, ANSLEN, or ANSLIST macro options to help you fill in the parameter list.

DATA=PRMMSG indicates that your reply data is in the parameter list. Use the PRMMSG parameter if you want the macro to fill in the parameter list.

ANSLIST=
Specifies whether the list format is being used.

ANSLIST=NO indicates that the list format is not being used. The ANSBUF parameter is the address of the complete reply.

ANSLIST=YES indicates that the address on the ANSBUF parameter identifies a list of addresses and lengths of discontiguous buffers that contains the reply data.

ANSBUF=
Specifies the *address* or the address of a list of addresses (*address-list*) from which IUCV moves the reply data.

Since the data is moved as part of the REPLY function, all buffer areas may be reused when the REPLY function completes.

ANSLEN=
Specifies the total *length* of the reply data. If ANSBUF specifies an address list (ANSLIST=YES), the value specified with ANSLEN is the total of the individual buffer lengths in the list.

PRMMSG=
Specifies the 8 bytes of message data that are moved into the parameter list.

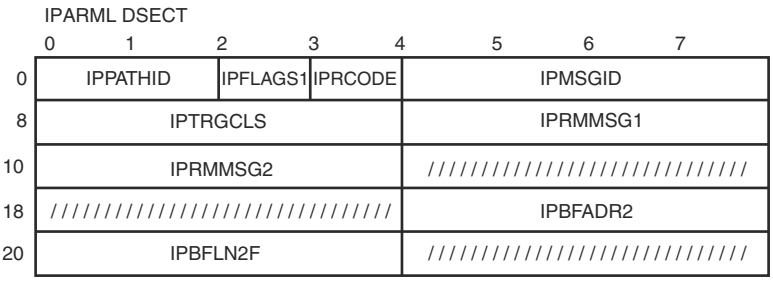
An additional option on REPLY lets you define priority messages.

PRTY=
Specifies whether this response is a priority message.

PRTY=YES indicates that this response is a priority message.

PRTY=NO indicates that this response is not a priority message.

Parameter List Format:



Parameter List Input Fields:

IPPATHID
Contains the path ID of the message to which you are replying.

IPFLAGS1
Contains the options for the REPLY function.

IPRMDATA (X'80')
Indicates that the reply is in the parameter list.

IPPRTY (X'20')
Indicates that this is a priority response.

IPANSLST (X'08')
Indicates that you are using an address list for the reply data.

IPMSGID
Contains the message ID of the message to which you are replying.

IPTRGCLS
Contains the message class of the message to which you are replying.

IPRMMSG1/IPRMMSG2
Contain the reply data when it is stored in the parameter list rather than a buffer. The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPBFADR2
Contains the address of the reply data.

IPBFLN2F
Contains the length of the reply data. Use this label with a fullword value. Use IPBFLN2 with a halfword value.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored in IPRCODE
2 - No message found.

Parameter List Output Fields:

IPBFADR2
Contains the address of the reply data, when ANSLIST=NO, updated by the number of bytes of data that IUCV moved. If ANSLIST=YES is specified, the address points to the current list entry IUCV is working on.

IPBFLN2F
Contains one of the following values:

If the answer buffer is the same length as the reply, this field contains 0.

If the answer buffer is longer than the reply, this field contains the number of bytes remaining in the buffer.

If the answer buffer shorter than the reply, this field contains a residual count (that is, the number of bytes remaining in the reply that does not fit into the buffer).

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
5 - X'05' - Answer buffer too short to contain message
6 - X'06' - Storage protection exception on answer buffer
7 - X'07' - Addressing exception on answer buffer
8 - X'08' - Message ID found but message class or path ID invalid
9 - X'09' - Message has been purged
10 - X'0A' - Message length is negative
21 - X'15' - Parameter list data not allowed on this path
22 - X'16' - Send/Answer buffer list invalid
23 - X'17' - Negative length in buffer list
24 - X'18' - Incorrect total length of buffer list lengths
25 - X'19' - PRMMSG option invalid with ANSLIST option
27 - X'1B' - Answer list not on a doubleword boundary
31 - X'1F' - IUCV function specified on an APPC/VM path
93 - X'5D' - A paging or storage error was detected in the ANSWER data area
94 - X'5E' - A paging or storage error was detected in the REPLY data area.

Program Exceptions

The program exceptions for IUCV REPLY are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address, the answer list address, the answer buffer address, or an answer buffer address in the answer list is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address, answer list address, answer buffer address, or an answer buffer address in the answer list does not match the key of the user.

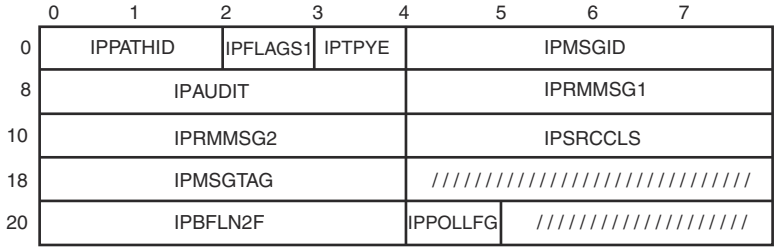
Completion Conditions

Message Complete External Interrupt:

To notify the originator of the message (the sender) that you have replied to the message, IUCV reflects an IUCV Message Complete external interrupt to the originator's virtual machine. If the sender had reserved buffers or address lists for use with this message, they can now be reused.

Notes:

1. Even though the user may not be dependent on the information contained in this interrupt, it should be processed as the message is not considered complete until the interrupt is reflected to the virtual machine.
2. A message complete interrupt is also reflected to the sender of the message when a one-way message with data specified in a buffer is received by the target. No message complete external interrupt is reflected for a one-way message with data specified in the parameter list.



IPPATHID
Contains the path on which the message was sent.

IPFLAGS1
Contains specific information about the message.

IPRMDATA (X'80')
Indicates that the 8-byte reply is in the interrupt information.

IPPRTY (X'20')
Indicates that this is a priority reply.

IPNORPY (X'10')
Indicates that this is a one-way message.

IPTYPE
Indicates a Message Complete external interrupt. If this is an incoming priority message completion, the interrupt type is X'06'. If this is an incoming nonpriority message completion, the interrupt type is X'07'.

IPMSGID
Contains the message ID.

IPAUDIT
Contains information about possible asynchronous error conditions which may have affected the normal completion of this message. If this field is 0, the message has completed successfully.

The meanings of the bits in the audit trail are:

IPADRPLE (X'800000')
Reply too long for buffer

IPADSNPX (X'400000')
Protection exception on send buffer

IPADSNAX (X'200000')
Addressing exception on send buffer

IPADANPX (X'100000')
Protection exception on answer buffer

IPADANAX (X'080000')
Addressing exception on answer buffer

IPADRJCT (X'040000')
Message was rejected

IPADPRMD (X'020000')
Reply specified DATA=PRMMSG, but this path cannot handle data in the parameter list.

IPADPGNR (X'010000')
Message purged on send or receive queue.

IPADRCPX (X'008000')
Protection exception on receive buffer

IPADRCAX (X'004000')
Addressing exception on receive buffer

IPADRPPX (X'002000')
Protection exception on reply buffer

IPADRPAX (X'001000')
Addressing exception on reply buffer

IPADSVRD (X'000800')
Path was severed

IPADRLST (X'000400')
Invalid RECEIVE or REPLY address list

(X'000200')
Reserved

(X'000100')
Reserved

IPADBLEN (X'000080')
Bad length in SEND buffer list

IPADALEN (X'000040')
Invalid Send/Answer buffer list such as bad address or length

IPADBTOT (X'000020')
Invalid total SEND buffer length

IPADATOT (X'000010')
Invalid total SEND answer length

(X'000008')
Reserved

(X'000004')
Reserved

(X'000002')
Reserved

(X'000001')
Reserved.

The fourth byte of IPAUDIT, IPASYRC, may contain on of the following error codes (for which an appropriate IPRCODE was given to your communications partner.

IPMCSNDA, 91 (X'00005B')
Paging or storage error was detected in SEND data area

IPMCRECA, 92 (X'00005C')
Paging or storage error was detected in RECEIVE data area

IPMCANSA, 93 (X'00005D')
Paging or storage error was detected in ANSWER data area

IPMCRPYA, 94 (X'00005E')
Paging or storage error was detected in REPLY data area.

IPRMMSG1/IPRMMSG2
Contains the message when it is stored with the interrupt information (indicated by IPRMDATA in IPFLAGS1). The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPSRCCLS
Contains the source message class.

IPMSGTAG
Contains the tag data of the message.

IPBFLN2F
Contains one of the following values, if the answer buffer is:

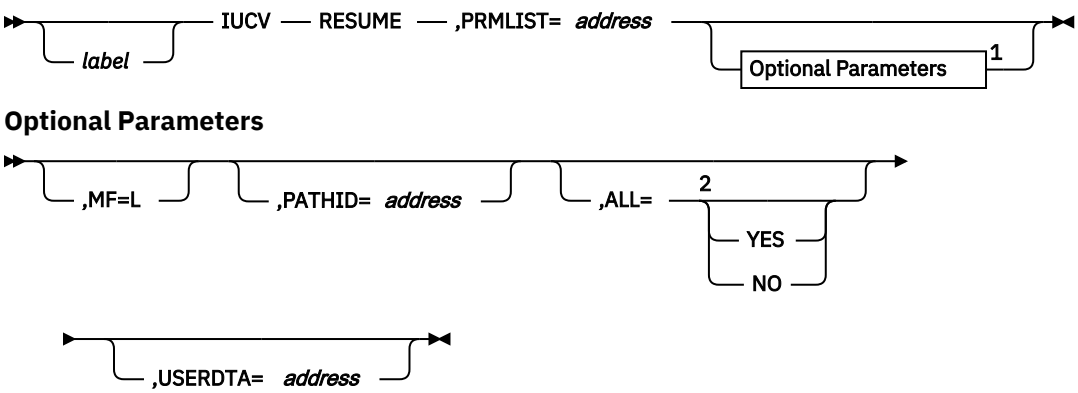
The same length as the reply, this field contains 0.
Longer than the reply, this field contains the number of bytes remaining in the buffer.
Shorter than the reply, this field contains a residual count (that is, the number of bytes remaining in the reply that does not fit into the buffer). The IPADRPLE bit is set in the audit trail on this condition.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

RESUME Function



Notes:

[1] Optional parameters can be entered in any order.
[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The RESUME function restores communications over a quiesced path.

Parameters

Required Parameters:

RESUME
Requests that CP perform the IUCV RESUME function.

PRMLIST=
Specifies the *address* of the RESUME parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

PATHID=
Specifies the path ID of the path on which you want to resume getting messages.

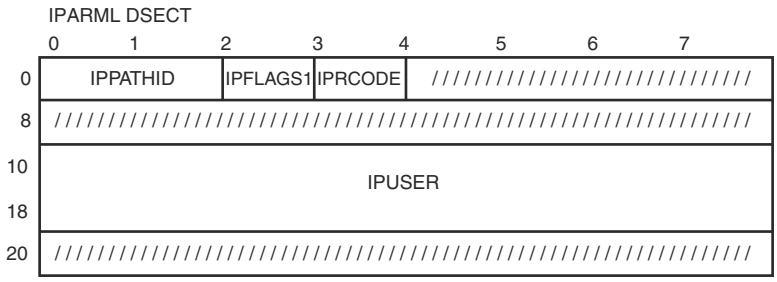
ALL=
Specifies whether communications should be restored for all paths for this virtual machine.

ALL=YES indicates that communications should be restored on all paths.

ALL=NO indicates that communications should be restored only on the path specified by PATHID.

USERDTA=
Specifies the data area containing the 16 bytes of user data that is to be reflected across the path. The user data is reflected as part of the IUCV Connection Resumed external interrupt.

Parameter List Format:



Parameter List Input Fields:

IPPATHID
Contains the path ID of the path you are resuming.

IPFLAGS1
Contains options for the RESUME function.

IPALL (X'80')
Indicates that you want to resume all paths for this virtual machine.

IPUSER
Contains the user data that is reflected across the path.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE.

Parameter List Output Fields:

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
31 - X'1F' - IUCV function specified on an APPC/VM path.
48 - X'30' - Partner system service does not support this function.

Program Exceptions

The program exceptions for IUCV RESUME are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

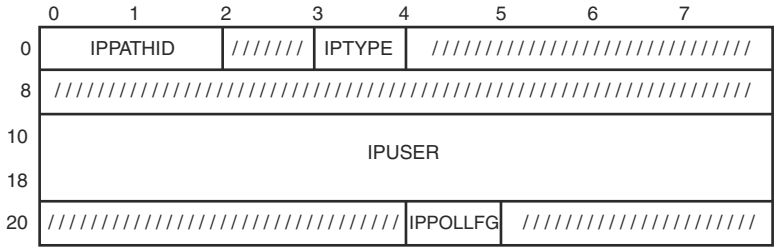
Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

Completion Conditions

Connection Resumed External Interrupt: To notify the other side of the path that the path has been resumed, IUCV reflects an IUCV Connection Resumed external interrupt.

The target virtual machine receives this external interrupt if it is enabled for IUCV interrupts in Control Register 0 and the PSW. The functions of SET MASK and SET CONTROL MASK also control the presentation of this type of interrupt.



IPPATHID
Contains the path ID of the path quiesced.

IPTYPE
Indicates a Connection Resumed external interrupt with a value of X'05'.

IPUSER
Contains the user data specified by the virtual machine that quiesced the path.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

RETRIEVE BUFFER Function

Purpose

The RETRIEVE BUFFER function terminates all use of IUCV. After the RETRIEVE BUFFER function completes, you may reuse the storage allocated for the IUCV external interrupt buffer since you will no longer receive IUCV external interrupts.

Since this function results in a sever of all IUCV paths, all outstanding IUCV communications are terminated as if each path has been individually severed.

When issued by a virtual machine, the RETRIEVE BUFFER function causes all paths except control paths to be severed. For example, if a program using CMSIUCV support issues HNDIUCV CLR, CMS issues the RETRIEVE BUFFER function and all paths are severed **except control paths**.

When issued by CP, the RETRIEVE BUFFER function severs **all** paths.

Note: Be aware that CP issues the RETRIEVE BUFFER function for the following commands:

- SYSTEM RESET
- IPL, which issues SYSTEM RESET
- LOGOFF.

This severs **all** paths, including control paths.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Parameters

Required Parameter:

RTRVBFR
Requests that CP perform the IUCV RETRIEVE BUFFER function.

Condition Codes and Return Codes

CONDITION CODES

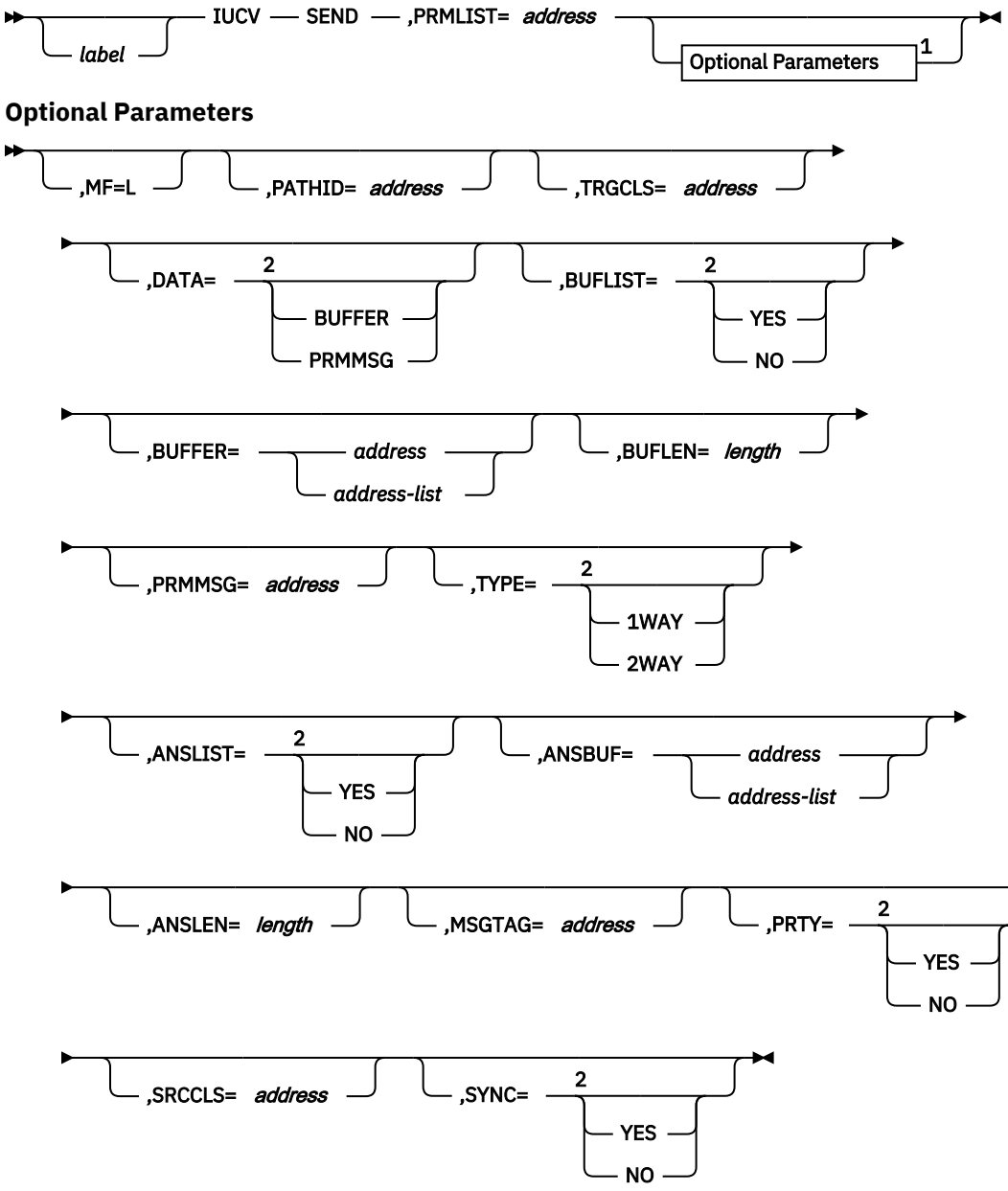
0 - Normal completion.

Program Exceptions

The program exceptions for IUCV RETRIEVE BUFFER are:

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

SEND Function



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The SEND function transmits data to another virtual machine. This data is called a *message* and may be specified in the parameter list or in a buffer. The message is sent over a path that has been established by the CONNECT function.

Parameters

Required Parameters:

SEND
Requests that CP perform the IUCV SEND function.

PRMLIST=
Specifies the *address* of the SEND parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

PATHID=
Specifies the path over which you wish to send the message. The *address* of the PATHID is to a halfword value.

TRGCLS=
Specifies the target message class associated with this message. This value, which is user defined, is considered part of the message identification (along with the PATHID and the MSGID field returned by IUCV). It can be used by the target virtual machine to receive particular messages.

With the SEND function you have an option of sending the message in a buffer or in the parameter list. The size of a parameter list message is very limited (only 8 bytes), but the overhead of an IUCV RECEIVE is avoided, which simplifies the communication process. The protocol that you chose on SEND does not affect the protocol chosen if the target must REPLY.

DATA=
Specifies the location of your message data for this IUCV communication.

DATA=BUFFER indicates that your message is in a buffer. You can use the BUFFER, BUFLEN, or BUFLIST macro parameter to help you fill in the parameter list.

DATA=PRMMSG indicates that your message is in the parameter list. Use the PRMMSG parameter if you want the macro to fill in the parameter list.

BUFLIST=
Specifies if the list format is being used.

BUFLIST=NO indicates that the list format is not being used. The BUFFER parameter is the address of the complete message.

BUFLIST=YES indicates that the address on the BUFFER parameter identifies a list of addresses and lengths of discontiguous buffers that hold the message text.

BUFFER=
Specifies the *address* or the address of a list of addresses (*address-list*) from which IUCV moves the message. Any message buffers should not be reused until you receive a Message Complete external interrupt for this message.

BUFLEN=
Specifies the total *length* of the message. If BUFFER specifies an address list (BUFLIST=YES), the value specified with BUFLEN is the total of the individual buffer lengths in the list.

PRMMSG=
Specifies the eight bytes of message data that are moved into the parameter list.

With the SEND function you have an option of sending the message with and without a reply. If you depend on the target virtual machine processing the message (for example, updating a database), you should use the 2-WAY protocol with a REPLY.

TYPE=
Specifies whether a reply is expected to this message.

TYPE=1WAY indicates that this is a one-way message and that the receiver will not reply to the message.

TYPE=2WAY indicates that this is a two-way messages and that the receiver is expected to reply to the message. You can use the ANSBUF, ANSLEN, or ANSLIST macro parameter to help you fill in the parameter list.

ANSLIST=
Specifies whether the list format is being used.

ANSLIST=NO indicates that the list format is not being used. The ANSBUF parameter is the address to contain the complete reply.

ANSLIST=YES indicates that the address on the ANSBUF parameter identifies a list of addresses and lengths of discontiguous buffers that contains the reply.

ANSBUF=
Specifies the *address* or the address of a list of addresses (*address-list*) into which IUCV moves the reply to this message. You do not know that the reply has been stored into this area until you receive a Message Complete external interrupt.

ANSLEN=
Specifies the total *length* of the expected reply. If ANSBUF specifies an address list (ANSLIST=YES), the value specified with ANSLEN is the total of the individual buffer lengths in the list.

Additional options for SEND include tagging the messages, sending priority messages, and defining a source class.

MSGTAG=
Specifies a tag to be associated with this message. This tag is returned to you on the IUCV Message Complete external interrupt. IUCV does not reference this tag so it may be used for any purpose that you desire. The tag information is not presented to the target user.

PRTY=
Specifies whether this is a priority message.

PRTY=YES indicates that this is a priority message.

PRTY=NO indicates that this is not a priority message.

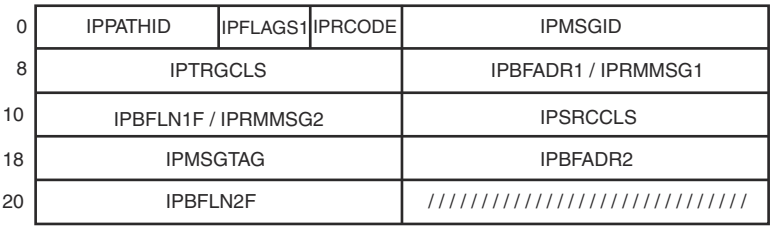
SRCCLS=
Specifies the source message class associated with this message.

The tag information is not presented to the target user. The source class is returned on the Message Complete external interrupt. You can use this field to identify different types of messages.

SYNC=
Designates a synchronous send request.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path ID on which to send the message.

IPFLAGS1
Contains options for the SEND function.

IPRMDATA (X'80')
Indicates that the message is in the parameter list.

IPBUFLST (X'40')
Indicates that you are using an address list for the message data.

IPPRTY (X'20')
Indicates that you are sending a priority message.

IPNORPY (X'10')
Indicates that this is a one-way message. No reply expected.

IPANSLST (X'08')
Indicates that you are using an address list for the reply data.

IPSYNC (X'04')
Indicates that you are requesting a synchronous send.

IPAPPCSN (X'02')
Indicates the protocol to be used on this path. This bit must be set to 0.

IPTRGCLS
Contains the target message class.

IPBFADR1
Contains the address of the message.

IPBFLN1F
Contains the length of the message buffer. Use this label with a fullword value. Use IPBFLN1 with a halfword value.

IPRMMSG1/IPRMMSG2
Contains the message when it is stored in the parameter list rather than a buffer. The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPSRCCLS
Contains the source message class.

IPMSGTAG
Contains the tag data of the message.

IPBFADR2
Contains the address to hold the reply.

IPBFLN2F
Contains the length of the reply area. Use this label with a fullword value. Use IPBFLN2 with a halfword value.

Condition Codes and Return Codes

The condition codes are:

0	Normal completion
1	Nonzero value stored in IPRCODE
2	Synchronous send request (IPSYNC) to the DASD Block I/O System Service has completed successfully.

Parameter List Output Fields:

IPMSGID
Contains a message ID that IUCV assigns the message.

IPRCODE
Contains the return code describing how this function completed.

0 - X'00'	Normal return

1 - X'01'	Path ID specified is not an established path
2 - X'02'	Path quiesced - SENDs not allowed
3 - X'03'	Message limit exceeded
4 - X'04'	Priority messages not allowed on this path
10 - X'0A'	Message length is negative
21 - X'15'	Message in parameter list not allowed on this path
25 - X'19'	PRMMSG invalid with BUFLIST parameter
26 - X'1A'	Buffer list not on a doubleword boundary
27 - X'1B'	Answer list not on a doubleword boundary
30 - X'1E'	IPAPPCSN flag in IPFLAGS1 not 0
31 - X'1F'	IUCV function specified on an APPC/VM path
48 - X'30'	Partner system service does not support this function.
96 - X'60'	The send was going to be routed via ISFC but the length was greater than the maximum allowed
97 - X'61'	The send was going to be routed via ISFC but the answer area was greater than the maximum allowed

Note: If you get a return code that is not documented here, it is an APPC/VM return code. An APPC/VM return code can result if the IPAPPC bit is set on during an IUCV CONNECT. For a description of the APPC/VM return code, refer to “APPCVM SENDDATA” on page 475.

Program Exceptions

The program exceptions for IUCV SEND are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list or buffer address that you specified is outside the virtual machine's storage.

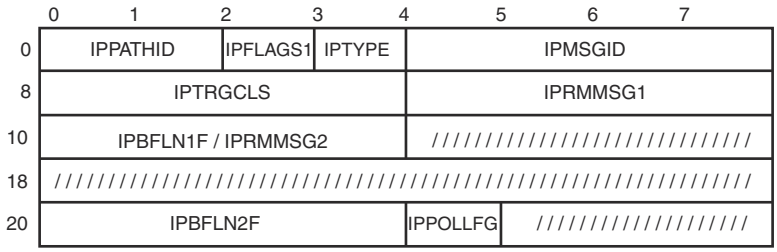
Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

Completion Conditions

Message Pending External Interrupt: To notify the target virtual machine that you have sent a message, IUCV reflects an IUCV Message Pending external interrupt to the target virtual machine.

The target virtual machine receives this external interrupt if it is enabled for IUCV interrupts in Control Register 0 and the PSW. The SET MASK function also controls the presentation of this type of interrupt.

The external interrupt contains the information that the target virtual machine needs to continue processing the message. If the message is being sent in a buffer, the target continues processing by issuing a RECEIVE or a REJECT. If the message is contained in the interrupt information (IPRMDATA in IPFLAGS1 is on), a RECEIVE is not needed. If the interrupt indicates a one-way message with the data in the parameter list, no further IUCV processing is necessary. Therefore, no message complete external interrupt is generated for the sender of the message.



IPPATHID
Contains the path on which the message was sent.

IPFLAGS1
Contains options for this message.

IPRMDATA (X'80')
Indicates that the 8-byte message is in the interrupt information.

X'40'
This value is reserved for IBM use only.

IPPRTY (X'20')
Indicates that this is a priority message.

IPNORPY (X'10')
Indicates that this is a one-way message and no REPLY is expected.

IPFGMID (X'04')
Is always set to 1 indicating that the message ID has been stored at IPMSGID.

IPFGPID (X'02')
Is always set to 1 indicating that the path ID has been stored at IPPATHID.

IPFGMCL (X'01')
Is always set to 1 indicating that the target message class has been stored at IPTRGCLS.

IPTYPE
Indicates an Incoming Message external interrupt. If this is an incoming priority message, the interrupt type is X'08'. If this is an incoming nonpriority message, the interrupt type is X'09'.

IPMSGID
Contains the message ID.

IPTRGCLS
Contains the target message class.

IPRMMSG1/IPRMMSG2
Contains the message when it is stored with the interrupt information (indicated by IPRMDATA in IPFLAGS1). The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPBFLN1F
Contains the length of the message.

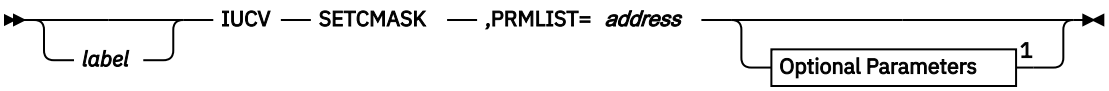
IPBFLN2F
Contains the length of the maximum expected reply.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

SET CONTROL MASK Function



Optional Parameters

Notes:

[1] Optional parameters can be entered in any order.

Purpose

The SET CONTROL MASK function enables or disables the following IUCV external interruptions:

- Pending Connection
- Connection Complete
- Connection Severed
- Connection Quiesced
- Connection Resumed.

IUCV external interrupts are controlled by several masks in the following priority order:

1. Submask bit 30 of Control Register 0
2. Bit 7 of the virtual machine PSW
3. Bits defined by the SET MASK function
4. Bits defined by the SET CONTROL MASK function.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Parameters

Required Parameters:

SETCMASK
Requests that CP perform the IUCV SET CONTROL MASK function.

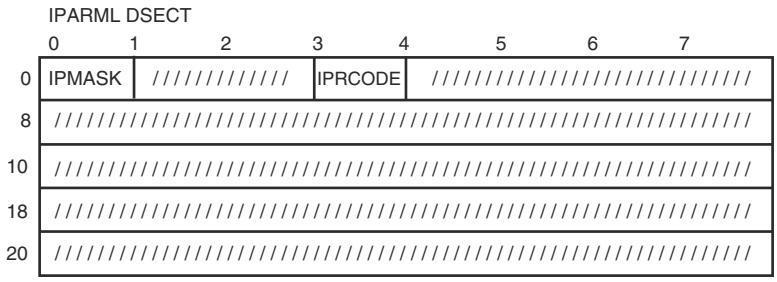
PRMLIST=
Specifies the *address* of the SET parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

MASK=
Specifies the mask byte to determine for which, if any, IUCV external interrupts a virtual machine is enabled.

Parameter List Format:



Parameter List Input Fields:

IPMASK
Contains the mask that specifies for which, if any, IUCV interrupts your virtual machine is enabled. The meanings of the bits in the mask are:

IPCLPC (X'80')
Enable for pending connections interrupts

IPCLCC (X'40')
Enable for connection complete interrupts

IPCLPS (X'20')
Enable for connection severed interrupts

IPCLPQ (X'10')
Enable for connection quiesced interrupts

IPCLPR (X'08')
Enable for connection resumed interrupts.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion

Parameter List Output Fields:

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return

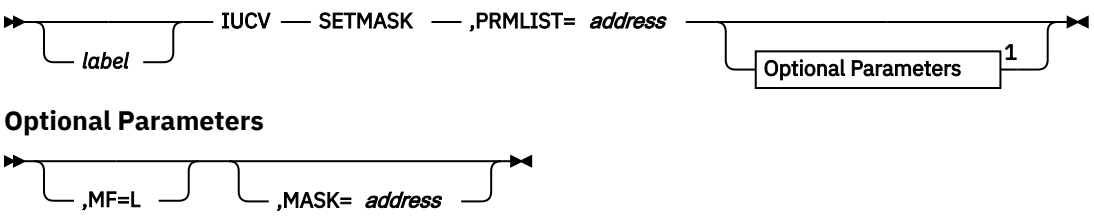
Program Exceptions

The program exceptions for IUCV SET CONTROL MASK are:

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Protection Exception
The storage key of the specified parameter list does not match the key of the user.

SET MASK Function



Notes:

[1] Optional parameters can be entered in any order.

Purpose

The SET MASK function enables or disables the following IUCV external interruptions:

- Nonpriority message interrupts
- Priority message interrupts
- Priority reply interrupts
- Nonpriority reply interrupts
- IUCV control interrupts.

Individual IUCV control interrupts can be controlled by using the SET CONTROL MASK function.

IUCV external interrupts are controlled by several masks in the following priority order:

1. Submask bit 30 of Control Register 0
2. Bit 7 of the virtual machine PSW
3. Bits defined by the SET MASK function
4. Bits defined by the SET CONTROL MASK function.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Parameters

Required Parameters:

SET MASK
Requests that CP perform the IUCV SET MASK function.

PRMLIST=
Specifies the *address* of the SET parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

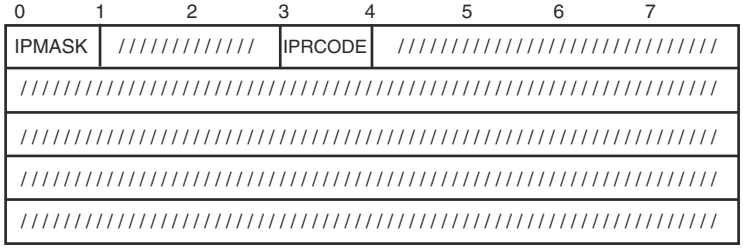
Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

MASK=
Specifies the mask byte to determine for which, if any, IUCV external interrupts a virtual machine is enabled.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPMASK
Contains the mask that specifies for which, if any, IUCV interrupts your virtual machine is enabled. The meanings of the bits in the mask are:

IPSNDN (X'80')
Enable for nonpriority message interrupts

IPSNDP (X'40')
Enable for priority message interrupts

IPRPYN (X'20')
Enable for nonpriority message completion interrupts

IPRPYP (X'10')
Enable for priority message completion interrupts

IPCTRL (X'08')
Enable for IUCV control interrupts

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion.

Parameter List Output Fields:

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return.

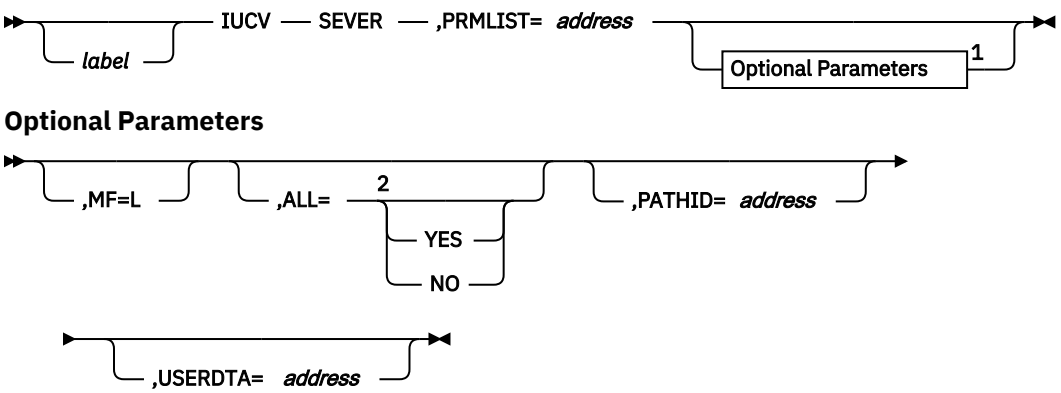
Program Exceptions

The program exceptions for IUCV SET MASK are:

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Protection Exception
The storage key of the specified parameter list does not match the key of the user.

SEVER Function



Notes:

[1] Optional parameters can be entered in any order.
[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The SEVER function terminates an IUCV path to another virtual machine. You can terminate an established path or a pending connection. When you SEVER an established path, all of your outstanding messages on that path are purged, and any incoming messages on that path are rejected. When you have severed a path, communications on that path are no longer allowed and no communication, in either direction, can occur. The path ID may be reused by IUCV when new paths are established.

When you receive an IUCV Connection Severed external interrupt, you can no longer send on that path, but you can process outstanding messages. You should always issue a SEVER in response to a Connection Severed interrupt to terminate your use of the path.

Parameters

Required Parameters:

SEVER
Requests that CP perform the IUCV SEVER function.

PRMLIST=
Specifies the *address* of the SEVER parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

ALL=
Specifies whether all paths for this virtual machine are to be severed.

ALL=YES indicates that all of your paths are to be severed.

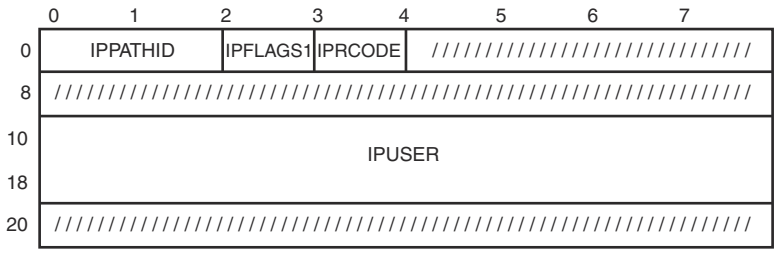
ALL=NO indicates that you do not want all of your paths severed, only the one specified by PATHID.

PATHID=
Specifies the path ID to be severed.

USERDTA=
Specifies the data area containing the 16 bytes of user data that IUCV is to reflect across the path. The user data is reflected as part of the IUCV Connection Severed external interrupt.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path ID of the path you want to sever.

IPFLAGS1
Contains options for the SEVER function.

IPALL (X'80')
Indicates that you want to sever all paths for this virtual machine.

IPAPPC (X'08')
Indicates the protocol to be used on this path. This bit must be set to 0.

IPUSER
Contains the user data that IUCV reflects across the path.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE.

Parameter List Output Fields:

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
30 - X'1E' - IPAPPC flag in IPFLAGS1 not 0.

Note: If you get a return code that is not documented here, it is an APPC/VM return code. An APPC/VM return code can result if the IPAPPC bit is set on during an IUCV CONNECT. For a description of the APPC/VM return code, refer to "APPCVM SEVER" on page 509.

Program Exceptions

The program exceptions for IUCV SEVER are:

Specification Exception
The parameter list is not on a doubleword boundary.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

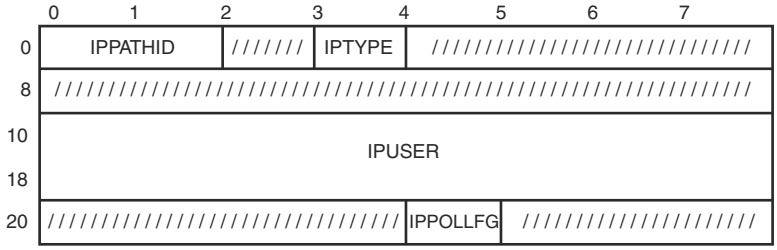
Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

Completion Conditions

Connection Severed External Interrupt: To notify the other side of the path that you wish to terminate communication on a path, IUCV reflects an IUCV Connection Severed external interrupt.

The target virtual machine receives this external interrupt if it is enabled for IUCV interrupts in Control Register 0 and the PSW. The functions of SET MASK and SET CONTROL MASK also control the presentation of this type of interrupt.



IPPATHID
Contains the path ID of the path being severed.

IPTYPE
Indicates a Connection Severed external interrupt with a value of X'03'.

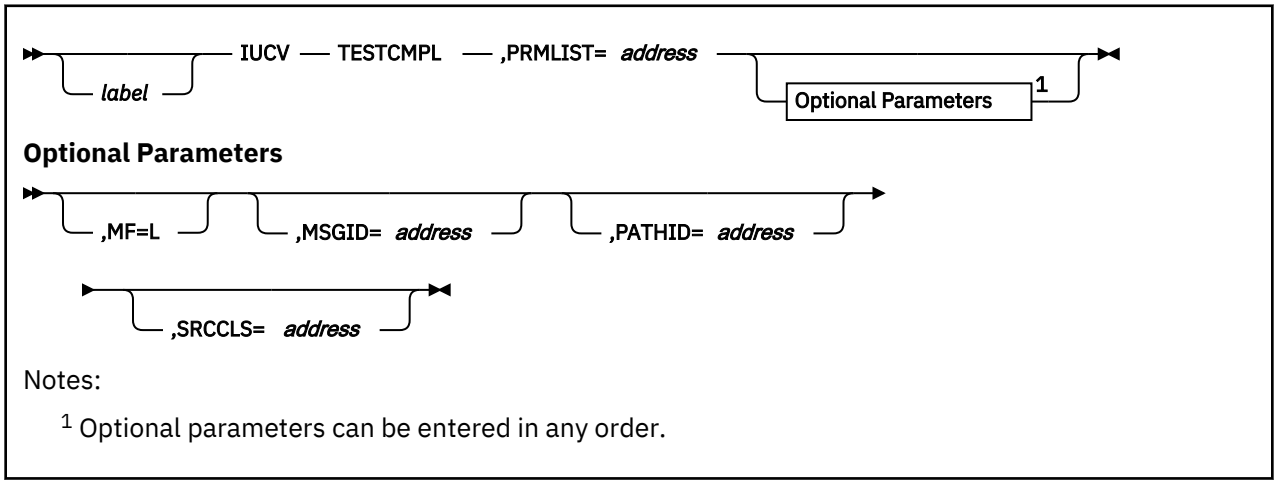
IPUSER
Contains the user data specified by the virtual machine that severed this path.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

TEST COMPLETION Function



Notes:

[1] Optional parameters can be entered in any order.

Purpose

The TEST COMPLETION function determines whether you have a message completion pending for your virtual machine. If a message completion is pending, information about the message is returned in the parameter list. Since you now have the message completion information, there is no need for IUCV to reflect an IUCV Message Completion external interrupt and you will not receive one for this message.

Since IUCV normally informs you of a message completion by reflecting a Message Completion external interrupt, you should not use the TEST COMPLETION function unless you have disabled for this type of interrupt. The IUCV SET MASK function can be used to disable your virtual machine for Message Completion external interrupts.

When invoking the TEST COMPLETION function, you can completely identify the message by specifying the message ID, path ID, and source message class. You can also identify the message by either the path ID or the source message class, or both. If you do not specify any identifiers when invoking the TEST COMPLETION function, any available message completion satisfies the function.

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Parameters

Required Parameters:

TESTCMPL
Requests that CP perform the IUCV TEST COMPLETION function.

PRMLIST=
Specifies the *address* of the TEST COMPLETION parameter list. The IUCV instruction is generated to reference the *address* specified. The address of the parameter list must be on a doubleword boundary.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
Lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

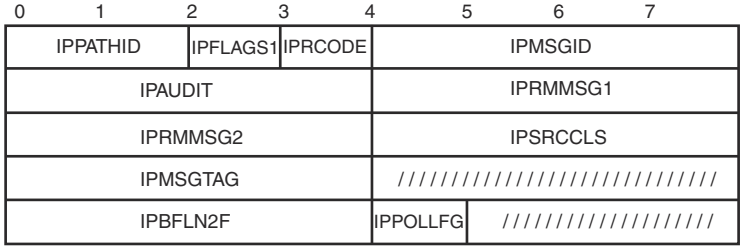
MSGID=
Specifies the message ID of the message. If the message ID is used to locate the message, the path ID and the source class must also be correctly specified in the parameter list.

PATHID=
Specifies the unique path identification number associated with a message.

SRCCLS=
Specifies the source message class associated with a message.

Parameter List Format:

IPARML DSECT



Parameter List Input Fields:

IPPATHID
Contains the path ID for the message completion.

IPFLAGS1
Contains options for the TEST COMPLETION function.

IPFGMID (X'04')
Indicates that you have specified a message ID (IPMSGID) to identify the message completion.

IPFGPID (X'02')
Indicates that you have specified a path ID (IPPATHID) to identify the message completion.

IPFGMCL (X'01')
Indicates that you have specified a source message class (IPSRCCL) to identify the message completion.

IPMSGID
Contains the message ID for the message completion.

IPSRCCLS
Contains the source message class for the message completion.

Condition Codes and Return Codes

CONDITION CODES

0 - Normal completion
1 - Nonzero value stored at IPRCODE
2 - No message found
3 - Nonzero audit trail stored.

Parameter List Output Fields:

IPPATHID
Contains the path ID on which the message was sent.

IPFLAGS1
Contains specific information about the message.

IPRMDATA (X'80')
Indicates that the 8-byte reply is in the parameter list.

IPPRTY (X'20')
Indicates that this is a priority message.

IPMSGID
Contains the message ID.

IPAUDIT
Contains information about possible asynchronous error conditions which may have affected the normal completion of this message. If this field is 0, the message has completed successfully.

The meanings of the bits in the audit trail are:

IPADRPLE (X'800000')
Reply too long for buffer

IPADSNPX (X'400000')
Protection exception on send buffer

IPADSNAX (X'200000')
Addressing exception on send buffer

IPADANPX (X'100000')
Protection exception on answer buffer

IPADANAX (X'080000')
Addressing exception on answer buffer

IPADRJCT (X'040000')
Message was rejected

IPADPRMD (X'020000')
Reply specified DATA=PRMMSG, but this path cannot handle data in the parameter list.

IPADPGNR (X'010000')
Message purged on send or receive queue.

IPADRCPX (X'008000')
Protection exception on receive buffer

IPADRCAX (X'004000')
Addressing exception on receive buffer

IPADRPPX (X'002000')
Protection exception on reply buffer

IPADRPAX (X'001000')
Addressing exception on reply buffer

IPADSVRD (X'000800')
Path was severed

IPADRLST (X'000400')
Invalid RECEIVE or REPLY address list

(X'000200')
Reserved

(X'000100')
Reserved

IPADBLEN (X'000080')
Bad length in SEND buffer list

IPADALEN (X'000040')
Bad length in SEND answer list

IPADBTOT (X'000020')
Invalid total SEND buffer length

IPADATOT (X'000010')
Invalid total SEND answer length

(X'000008')
Reserved

(X'000004')
Reserved

(X'000002')
Reserved

(X'000001')
Reserved.

IPRMMSG1/IPRMMSG2
Contains the message when it is stored in the parameter list (indicated by IPRMDATA in IPFLAGS1). The label IPRMMSG refers to the combined IPRMMSG1 and IPRMMSG2 fields.

IPSRCCLS
Contains the source message class.

IPMSGTAG
Contains the tag data of the message.

IPBFLN2F
Contains one of the following values:

If the buffer is exactly the correct length, this field contains 0.

If the buffer is too long, this field contains the number of bytes unused in the buffer.

If the buffer is too short, this field contains a residual count (that is, the number of bytes remaining of the reply that do not fit into the buffer). The IPADRPLE bit is set in the audit trail on this condition.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that another iteration of this function will probably not find a reply waiting at this time.

IPRCODE
Contains the return code describing how this function completed.

RETURN CODES in IPRCODE

0 - X'00' - Normal return
1 - X'01' - Path ID specified is not an established path
8 - X'08' - Message ID found but message class or path ID invalid.

Program Exceptions

The program exceptions for IUCV TEST COMPLETION are:

Specification Exception
The parameter list is not on a doubleword boundary, or the message ID was specified without the path ID and the message class.

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Addressing Exception
The parameter list address that you specified is outside the virtual machine's storage.

Protection Exception
The storage key of the specified parameter list address does not match the key of the user.

TEST MESSAGE Function

Purpose

The TEST MESSAGE function determines whether any IUCV Message Pending or IUCV Message Complete external interrupts are queued for your virtual machine.

This function, when used with the DESCRIBE and TEST COMPLETION functions, lets the virtual machine avoid the external interrupt handling associated with messages. In some applications, as when only one type of message is ever handled, the DESCRIBE may also be avoided and a RECEIVE issued directly.

When you receive message or message completions, IUCV informs you by reflecting a Message Pending or Message Complete external interrupt to your virtual machine. Therefore, unless you disable your virtual machine for Message Pending and Message Complete external interrupts, you should not use the TEST MESSAGE function.

If, when your virtual machine invokes the TEST MESSAGE function, it finds that there are no messages or message completions pending, your virtual machine enters a wait state. Your virtual machine remains enabled for all interrupts that were enabled when the TEST MESSAGE function was issued.

If, while your virtual machine is in a wait state, you receive an IUCV message or message completion, your virtual machine resumes execution by re-executing the TEST MESSAGE function (which returns a condition code).

This function has a different meaning in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for IUCV Applications” on page 308.

Condition Codes and Return Codes

The condition codes are:

1	Message pending
2	Message completion pending
3	Both message and message completion pending

Program Exceptions

The program exceptions for IUCV TEST MESSAGE are:

Operation Exception
The external interrupt buffer has not been declared using the DECLARE BUFFER function, your virtual machine is not in supervisor state, or a previous RETRIEVE BUFFER function is outstanding and has not completed yet.

Part 3. The Advanced Program-to-Program Communication/VM

This part contains the following chapters:

- Chapter 6, “Overview of the APPC/VM Assembler Interface,” on page 387
- Chapter 7, “APPCVM Macro Functions,” on page 411
- Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521
- Chapter 9, “Migrating Programs from IUCV to APPC/VM,” on page 567
- Chapter 10, “APPC Mapped with APPC/VM,” on page 573

© Copyright IBM Corp. 1991, 2020

Chapter 6. Overview of the APPC/VM Assembler Interface

In its simplest form, connectivity is the ability of one program to communicate with another program. In this document, we are concerned with communications between two application programs. Application programs are typically written to communicate with one another because a user needs access to some kind of data.

System Network Architecture (SNA) defines various sets of rules for data to be transmitted in a network. Application programs communicate with each other using a layer of SNA called **Advanced Program-to-Program Communication (APPC)**. APPC is also known as **LU 6.2**. z/VM implements the base set of APPC and several APPC option sets using **Advanced Program-to-Program Communication/VM (APPC/VM)**.

z/VM provides two programming interfaces to APPC/VM:

1. A low-level interface intended for programs written in assembler language. This low-level interface consists of CMS macros and CP macros. In this document we will discuss the APPC/VM macros that are part of the CP programming interface. The CMS macros that you can use with APPC/VM are documented in the *z/VM: CMS Macros and Functions Reference* and *z/VM: CMS Application Development Guide for Assembler*.
2. **Common Programming Interface (CPI) Communications**. This interface (also known as SAA communications interface) can be used with any programming language defined in the System Application Architecture (SAA). For z/VM, this interface can be used in CMS. This interface should be used for new application development.

   Note: For more information about CPI Communications routines, see *CPI Communications Reference*.

Overview of APPC/VM Assembler Interface

The assembler programming interface for APPC/VM allows communications between application programs that are written in assembler language. The APPC/VM assembler interface implements the base set of APPC (SNA LU 6.2) verbs and several APPC option sets.

The APPC/VM assembler interface provides macros and parameter lists that applications can use to set up and control the communications environment within one z/VM system, and among z/VM systems in a TSAF or Communication Services (CS) collection. (For programs communicating outside a TSAF or CS collection to an SNA network, AVS translates APPC/VM protocol into APPC/VTAM, which is the VTAM® implementation of APPC.)

In addition, z/VM has implemented some IUCV functions for an APPC/VM environment. IUCV functions are not part of the APPC architecture and are unique to z/VM. **Note: The remainder of this document refers to the APPC/VM assembler programming interface as simply *APPC/VM*.**

Basics of APPC/VM

The following sections describe some of the basics of APPC/VM communication: paths, states, and interrupts.

APPC/VM Paths

An APPC/VM path is a logical connection between one or more virtual machines. Information flows on APPC/VM paths. To establish an APPC/VM path between two virtual machines, at least one of the virtual machines must be authorized in the IUCV directory statement.

© Copyright IBM Corp. 1991, 2020

A path is created when the source virtual machine invokes the CONNECT function and the target virtual machine invokes the ACCEPT function. Once the path is created, communications can begin. Programs identify a path using the PATHID parameter of the pertinent APPC/VM function.

The target virtual machine can prevent the path from being established by invoking the SEVER function. Either virtual machine can destroy an established path with the SEVER function.

A single virtual machine can have up to 65,536 APPC/VM paths defined. Two virtual machines can have more than one path between them. Communication can occur over any and all paths at the same time.

APPC/VM States

The APPC/VM interface is a half-duplex communications protocol. This means that only one of the communications partners can send data at any given time. Because of this, APPC/VM uses states to define what functions a program can and cannot issue at any given time.

A program is always in a single state for a particular conversation. When your program or your communications partner issues an APPC/VM function, the state of the conversation may change. If your virtual machine is communicating with different virtual machines through various paths, it may be in different states on different paths at the same time. A program participating in multiple conversations could have multiple states, too.

The basic states for APPC/VM assembler programs are:

Reset
The state for each program before communications begin and after communications end.

Connect
The state for a source program after a connection has been started but before it has completed, or the state for a target program after it has received a connection pending interrupt but before it has accepted.

Send
The state in which a program is allowed to send data.

Receive
The state in which a program is ready to receive data.

Confirm
The state in which a program must respond to its communications partner.

Sever
The state a program is in when its partner stops communications.

These APPC/VM states are based on the states that APPC defines, but there are two differences:

1. The Connect state is unique to APPC/VM
2. The Sever state is analogous to the APPC Deallocate state.

In addition, APPC/VM defines several other states for special programs involving coordinated resource recovery (CRR). Refer to Table 72 on page 405 for details about these APPC/VM states.

APPC/VM Interrupts

In APPC/VM, your program may receive notification of pending functions through external ***interrupts***. Interrupts are caused by actions taken by the virtual machine on the other end of the local APPC/VM path. Interrupts indicate pending and completed functions. For example, your virtual machine receives an interrupt when another virtual machine sends you some kind of message that it wants you to receive.

At the start of your program, you must create a buffer to hold the interrupt information for any APPC/VM functions on any path. (You can create this buffer using the IUCV DCLBFR function.) This buffer is a 40-byte area called an external interrupt buffer. When your program is presented with an interrupt, information about the interrupt goes in this external interrupt buffer.

The possible APPC/VM interrupts you can get fall into two categories. The first type of interrupt signals that your communications partner has invoked some function, independent of your actions. These interrupts are:

- Connection pending
- Message pending
- Request-to-Send
- Sever.

The second type of interrupt signals the completion of a function that you initiated. These interrupts are:

- Connection complete
- Function complete.

The six basic types of APPC/VM interrupts are described in the following sections.

Connection Pending External Interrupt

You get a connection pending interrupt when a virtual machine invokes an APPCVM CONNECT function to connect to your virtual machine. The interrupt is placed in a control buffer if the virtual machine wants to connect to a private resource; otherwise, the interrupt is presented to your virtual machine's application buffer.

Refer to the diagram in APPCVM CONNECT, Connection Pending External Interrupt.

Message Pending External Interrupt

You get a message pending interrupt when your communications partner issues an APPC/VM function for which you should issue an APPCVM RECEIVE. Your communications partner issuing any of the following APPCVM macro functions can cause a message pending interrupt:

- RECEIVE
- SENDCNF
- SENDDATA
- SENDERR.

You only get a message pending interrupt if you are in Receive state on the corresponding path.

Refer to the diagram in APPCVM SENDDATA, Message Pending External Interrupt format.

Request-to-Send External Interrupt

You get a request-to-send interrupt when your communications partner issues the APPCVM SENDREQ function to request to send data.

Refer to the diagram in APPCVM SENDREQ, SENDREQ Interrupt.

Sever External Interrupt

You get a sever interrupt when the program to which you are connected or trying to connect to invokes an APPCVM SEVER or IUCV SEVER, invokes an HNDIUCV CLR (or IUCV RTRVBFR), or abends. You could also get a sever interrupt when the virtual machine to which you are connected or trying to connect to resets its virtual machine or logs off.

Note: If you get a sever interrupt after you have issued an APPC/VM function to your partner, do not assume that your function has terminated.

Refer to the diagram in APPCVM SEVER, SEVER External Interrupt format.

Connection Complete External Interrupt

You get a connection complete interrupt when you issue the APPCVM CONNECT WAIT=NO function and the virtual machine on the other end of the local APPC/VM path accepts the connection.

When you get a connection complete interrupt, do not assume that the target program performed any action to cause your function to complete.

For a diagram of the connection complete external interrupt in APPCVM CONNECT, see "Condition Codes and Return Codes" on page 424.

Function Complete External Interrupt

You get a function complete interrupt when the function that you issued completes. The completion of any of the following APPC/VM functions can cause a function complete interrupt:

- RECEIVE
- SENDCNF
- SENDDATA
- SENDERR
- SEVER.

The function complete interrupts are shown in the following sections:

- APPCVM RECEIVE Output Parameter List in APPCVM RECEIVE
- AFFCVM SENDCNF Output Parameter List in APPCVM SENDCNF
- APPCVM SENDDATA Output Parameter List in APPCVM SENDDATA
- APPCVM SENDERR Output Parameter List in APPCVM SENDERR
- SEVER Output Parameter List format in APPCVM SEVER

Whenever there are multiple APPC/VM interrupts queued for the virtual machine, control interrupts are always reflected to the virtual machine in first-in-first-out (FIFO) order before message interrupts. Message interrupts of the same subtype are reflected in first-in-first-out (FIFO), but message interrupts of different subtypes are reflected in the order shown in the previous list.

Interrupts are reflected to the virtual machine in this order regardless of the order in which the interrupts were queued for the virtual machine. There are many conditions which can cause more than one interrupt to be queued for a virtual machine, some of which are beyond the control of the application. For example, if the virtual machine disables for APPC/VM interrupts for a period of time or if the virtual machine is communicating with multiple partners, then often multiple APPC/VM interrupts will be on the virtual machine's queue. Also, the relative priorities and time slices given to the communicating virtual machines can affect the order in which APPC/VM interrupts are presented. For example, if virtual machine A sends a one-way message to virtual machine B, and B receives the message, a message complete interrupt (function complete interrupt for SENDDATA) is queued for A. If B then severs the APPC/VM path, then a sever interrupt is queued for A. If A is not dispatched, or doesn't enable for APPC/VM interrupts until after the sever interrupt is queued, then A would see the sever interrupt first. If A is dispatched and is enabled for APPC/VM interrupts before the sever interrupt is queued, then A would see the message complete interrupt first.

Invoking APPC/VM Communication Functions

z/VM programs at each end of an APPC/VM path use APPC/VM functions to communicate with each other. Most APPC/VM communications functions are provided as parameters of the APPCVM macro.

To use the APPCVM or IUCV macro, issue the CMS GLOBAL command for the HCPGPI MACLIB before assembling your program.

The APPC/VM communications functions used with the APPCVM macro are:

Function	Description	Page
CONNECT	Establishes and reserves a path to communicate with another program.	“APPCVM CONNECT” on page 412
QRYSTATE	Determines the current state of a path.	“APPCVM QRYSTATE (Query State)” on page 447
RECEIVE	Receives data and information sent to your program.	“APPCVM RECEIVE” on page 451
SENDCNF	Sends a confirmation request to your communications partner.	“APPCVM SENDCNF (Send Confirm)” on page 465
SENDCNFD	Sends a response to a confirmation request.	“APPCVM SENDCNFD (Send Confirmed)” on page 471
SENDDATA	Sends data to your communications partner.	“APPCVM SENDDATA” on page 475
SENDERR	Sends notice to your communications partner that your program has detected an error.	“APPCVM SENDERR (Send Error)” on page 490
SENDREQ	Requests permission to send data.	“APPCVM SENDREQ (Send Request)” on page 501
SETMODFY	Sets the state to Defer_Receive or Defer_Sever and sets the sync-point control modifier.	“APPCVM SETMODFY (Set Modify)” on page 505
SEVER	Ends communications with another program.	“APPCVM SEVER” on page 509

In addition, some APPC/VM functions are provided as parameters on the IUCV macro. The APPC/VM communications functions used with the IUCV macro are:

- IUCV ACCEPT
- IUCV QUERY
- IUCV SEVER.

The IUCV macro functions relate to both APPC/VM and IUCV paths. The IUCV functions are not defined by the SNA LU 6.2 (APPC architecture) verb interface, but they are a necessary complement for APPC programs executing in a z/VM processor. The IUCV macro functions that relate to APPC/VM are:

Function	Description	Page
ACCEPT	Accepts a connection from another virtual machine or from your own virtual machine.	“IUCV ACCEPT” on page 524
CONNECT	Establishes a path to the Identify system service (*IDENT).	“IUCV CONNECT” on page 529
DCLBFR*	Defines an interrupt buffer to prepare for APPC/VM communications.	“IUCV DCLBFR (Declare Buffer)” on page 533
DESCRIBE*	Gets a description of a pending APPC/VM or IUCV message.	“IUCV DESCRIBE” on page 538

Function	Description	Page
IPOLL*	Checks for pending replies or incoming messages.	“IUCV IPOLL (Interrupt Poll)” on page 540
QUERY	Determines the maximum number of communications paths that can be established for your virtual machine.	“IUCV QUERY” on page 543
RTRVBFR*	Undefines the external interrupt buffer, ending communications.	“IUCV RTRVBFR (Retrieve Buffer)” on page 548
SETCMASK*	Enables or disables external interrupts for certain APPC/VM and IUCV control functions.	“IUCV SETCMASK (Set Control Mask)” on page 550
SETMASK*	Enables or disables external interrupts for certain APPC/VM and IUCV functions.	“IUCV SETMASK” on page 553
SEVER	Terminates an APPC/VM or IUCV path or terminates a path established to the *IDENT system service.	“IUCV SEVER” on page 556
TESTCMPL*	Determines if any messages or functions have been completed.	“IUCV TESTCMPL (Test Completion)” on page 562
TESTMSG*	Allows programs to avoid using external interrupt handling.	“IUCV TESTMSG (Test Message)” on page 566

Note: Other IUCV macro functions can be used from APPC/VM, but are not recommended for use by APPC/VM programs running in CMS. For information about APPC/VM and IUCV macro functions that you can use in CMS, see the *z/VM: CMS Application Development Guide for Assembler*.

*These functions have different meanings in a virtual MP environment. For more information about a virtual MP environment, see “Virtual MP Considerations for APPC/VM Applications” on page 398.

Using Basic APPC/VM Functions

To write starter APPC/VM programs, you need to know the APPC/VM functions that do the basic steps of starting a conversation, communicating in a conversation, and ending a conversation.

Starting a Conversation

To start a conversation, your user program must issue an APPCVM CONNECT with a resource ID. Depending on the type of connection, a user program also might supply a ***connection parameter list extension***, which contains detailed information that is necessary to make a connection. (If the resource ID maps to an entry in a CMS communications directory file, the program generally does not have to build the extension itself, CMS does it.)

When your program issues an APPCVM CONNECT, your communications partner gets a connection pending interrupt. Your partner should examine the interrupt before accepting or rejecting the connection. The interrupt contains information such as the resource ID for which the connection is being made and the user ID of the requesting program.

In addition to the connection pending interrupt, your partner can also get other kinds of data before accepting the connection:

- *Allocate data*, which provides more details about the pending connection
- *Program Initialization Parameters (PIP data)*, which can serve many purposes. (See Specifying a PIP Variable for more information.)

After examining all this data, your communications partner can do either of the following in response to your connect request:

- Accept the connection if it wants to communicate with your program, making sure to specify the path ID that was on the connection pending interrupt.
- Accept the connection and then immediately sever the connection if it does not want to communicate with your program.

Sending and Receiving Data on the Conversation

When you issue the command to connect to a target, and your communications partner accepts the connection:

- Your program is in Send state for the conversation.
- Your communications partner's program is in Receive state.

You can now send data, using APPCVM SENDDATA. Your program must set up data in buffers, and the data must be in APPC logical record format. Remember that you can only send data when your program is in Send state and receive data when your program is in the Receive state.

As you send data, your communications partner is notified through one or more message pending interrupts. Your partner can then receive the data using APPCVM RECEIVE.

Ending a Conversation

When your program is finished communicating with your partner program, you should end the conversation by issuing APPCVM SEVER or IUCV SEVER.

APPCVM SEVER

You can issue an APPCVM SEVER anytime after you have established a path with your partner (that is, you must first issue a CONNECT and your partner must issue an ACCEPT). At this point your partner will receive a sever interrupt that contains information about the path and any errors that may have occurred during the sever.

You can also include log data on an APPCVM SEVER. This data can be accepted by your partner and used for debugging and error recovery.

After you issue an APPCVM SEVER, your partner can examine the sever interrupt information and:

- Issue an APPCVM SEVER to sever their side of the path.
- If log data is available, issue an APPCVM RECEIVE to obtain that data before severing using IUCV SEVER.
- Issue an IUCV SEVER to sever their side of the path.

IUCV SEVER

An IUCV SEVER can be issued at any time during a conversation. Usually your partner will receive a sever interrupt which contains information about the path and any errors that may have occurred during the sever.

After receiving an IUCV SEVER, your partner can issue an:

- APPCVM SEVER to sever their side of the path.
- IUCV SEVER to sever their side of the path.

In a CMS environment, you can also use the CMSIUCV macro. For more information on the CMSIUCV macro, see *z/VM: CMS Macros and Functions Reference*.

Managing a Resource

For a virtual machine to manage a local, global, or system resource, it must first be authorized to connect to the Identify System Service, *IDENT. Your system administrator is the person who can authorize your

virtual machine to manage a particular resource. To do this, the administrator must specify a special IUCV *IDENT statement in your virtual machine's directory entry. For more information on the IUCV directory control statement, see *z/VM: CP Planning and Administration*.

Your virtual machine must connect to *IDENT before using it to manage a resource or gateway. This connect should be done using an IUCV CONNECT. For more information on using IUCV CONNECT to connect to *IDENT, see "IUCV CONNECT" on page 529.

If your virtual machine becomes a local, global, or system resource manager by establishing a connection to *IDENT, APPC/VM lets other virtual machines connect to you. The connecting virtual machines must specify, on their connection request, the resource ID you identified through *IDENT.

*IDENT maintains a local system resource table. *IDENT adds an entry to this table each time it accepts a virtual machine connection and deletes the entry when it severs the associated connection. A virtual machine manages a resource only while connected to *IDENT. (For details on how *IDENT works, refer to Chapter 16, "Identify System Service (*IDENT)," on page 731.)

Revoking a Resource

To manage a resource, a program must identify the resource name by connecting to *IDENT. The virtual machine for the program must have proper directory authorization to connect to *IDENT. A program can also ***revoke***—stop management of—a resource name.

A program can revoke a resource it manages by issuing an IUCV SEVER to sever its path to *IDENT. *IDENT then deletes the resource from the system resource table and severs its half of the path. Your program then gets an IUCV sever interrupt from *IDENT. The SEVER does not affect existing APPC/VM paths to your virtual machine.

If another virtual machine connects to *IDENT to manage the resource that you revoked, requests to connect to the resource go to that virtual machine.

Note: If a virtual machine tries to connect to a resource that you manage before your revoke completes, the path may be established.

A program can revoke a resource that *another* program manages by issuing an IUCV CONNECT to *IDENT with the appropriate user data. The issuing program's virtual machine must have proper directory authorization to connect to *IDENT and to do this kind of revoke. Your system administrator can authorize your program's virtual machine to revoke a particular resource.

For more information on using IUCV CONNECT to connect to *IDENT, see "IUCV CONNECT" on page 529 and to Chapter 16, "Identify System Service (*IDENT)," on page 731.

You might have a case where two disjoint TSAF or CS collections merge, and the same resource name is identified (through connections to *IDENT) by resource managers in both collections. If this happens, the TSAF virtual machine issues a revoke to one of the competing resource managers while ISFC issues revokes to both resource managers. *IDENT severs the paths to the resource manager programs that own the revoked resources.

Understanding APPC/VM Parameter Lists

Note: For more information on IUCV macro functions that can be used on APPC/VM paths, see Chapter 8, "IUCV Macro Functions for Use in APPC/VM," on page 521.

Parameter lists for all APPC/VM functions are 40 bytes. Your program can specify a single 40-byte area in storage to use for all APPC/VM function parameter lists. The address of the 40-byte area must be a guest real address in the virtual machine's host-primary address space (guest=real).

The APPCVM and IUCV macros are both in the HCPGPI MACLIB, along with the IPARML DSECT. The APPCVM and IUCV macros use labels defined in IPARML DSECT to complete the parameter list. You need to provide a USING statement for the IPARML COPY file when you invoke the macro, and define proper storage for it.

To reference fields in parameter lists for APPCVM and IUCV functions, always use the name defined for that field in the IPARML DSECT, rather than using the displacement within IPARML DSECT. The parameter lists (IPARML DSECT mappings) are shown with each APPCVM and IUCV macro function.

All address fields in the APPCVM and IUCV parameter lists are 4-byte reserved fields.

Setting for Optional Parameters

If you do not specify a parameter shown as *optional* when you invoke an APPCVM or IUCV macro function, the macro assumes that you have stored a value in the parameter list before invoking the APPCVM macro. **Therefore, you should set all fields and flag bits in the parameter list that you are not defining for a particular APPCVM or IUCV macro function to 0**. This helps ensure that if these fields are defined in the future, applications will continue to work.

Note: There are **no explicit default values** for the optional parameters on the IUCV and APPCVM macros. As such, if a parameter is not specified or is specified with a null value, the value of the field it represents will not be altered by the macro invocation. This allows the actual parameter list to be filled in on multiple macro invocations using the MF=L option. This also allows an application to build a parameter list once using the macro and use it multiple times without having to recode all of the macro parameters. However, please take note of any values that may be altered by function completion.

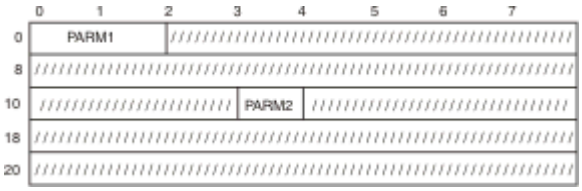
Parameters Reserved for IBM Use Only

You should set all fields and flag bits in the parameter list that are reserved for IBM use only to 0. Otherwise, unpredictable results may occur. All areas denoted by a series of slashes (////...) are reserved for IBM use.

Reading the Parameter Lists

For your reference, the format of input and output parameter lists is shown in this chapter. Parameter lists for APPC/VM functions are 40 (X'28') bytes. These 40 bytes are shown in rows of 8 bytes for easier reading. The numeric values for rows and columns are shown in hexadecimal.

For example, look at the following example parameter list:



PARM1 is in the first 2 bytes of the parameter list, and PARM2 is in byte 19. The slash characters (/) indicate bytes that are reserved for IBM use only and should contain X'00's.

Formatting the Parameter List with MF=L

If you specify MF=L on a macro, the macro generates the instructions necessary to format the parameter list by using the keyword values provided on the macro. However, the macro does **not** generate any instructions to execute the specified function.

When using CMS support of APPC/VM, it is especially useful for you to use MF=L. For example, to request a connection in CMS, you can issue functions in this sequence:

1. Issue APPCVM CONNECT with MF=L and any other appropriate macro keywords to fully prepare the parameter list.
2. Then issue CMSIUCV CONNECT with any other appropriate macro keywords to invoke the connection request.

Step 1 formats the parameter list, and step 2 invokes the function using the already-formatted parameter list. You can use a similar sequence for these other functions in CMS:

- IUCV ACCEPT followed by CMSIUCV ACCEPT

- APPCVM SEVER followed by CMSIUCV SEVER
- IUCV SEVER followed by CMSIUCV SEVER.

If you do not specify MF=L on an APPCVM macro, the macro generates the instructions necessary to:

1. Format the parameter list as specified by parameter values on the macro.
2. Execute the APPC/VM function.

Registers Altered by APPCVM and IUCV Macro Functions

If you specify MF=L on an APPCVM or IUCV macro function, the macro **may alter R0**. If you do not specify MF=L, **R0 is still altered** and for IUCV functions only **R1 may be altered**.

For more information on macro parameter lists, see *z/VM: CMS Macros and Functions Reference*.

Condition Codes and Return Codes

APPCVM macros generate four condition codes and four return codes. This section first gives a summary of the conditions under which each of the 4 return codes are generated followed by a description of each condition code and return codes that APPCVM macro functions generate. See the individual APPCVM macro details for the specific condition codes and return codes generated by each macro.

See Table 68 on page 396 for a summary of the conditions under which each of the 4 return code fields applies. In the figure, an *X* indicates that the field applies for the corresponding condition code.

Table 68. Applicable Codes Based on the Condition Code

	IPRCODE	IPAUDIT	IPCODE	IPWHATRC
CC=0		X	X	X
CC=1	X			
CC=2			X	X
CC=3		X		

Note: For CC=0; IPAUDIT, IPCODE, and IPWHATRC are meaningful in the **function complete interrupt**. (The interrupt signals the completion of the function that started with CC=0.)

For CC=1, 2, or 3; IPRCODE, IPAUDIT, IPCODE, and IPWHATRC are meaningful in the **output parameter list** for the function you issued.

Condition Codes

The condition code (CC) is stored in the program status word (PSW). There are four possible values for condition codes: 0, 1, 2, and 3.

CC=0
The function has started but has not yet completed. This condition code applies only when WAIT=NO (asynchronous processing). When the function does complete, a virtual machine gets a function complete interrupt.

Note: Throughout this section, function complete interrupt is meant to include the connection complete interrupt also.

CC=1
An error occurred when the function was initiated. In this case, the error code is stored in the IPRCODE field of the output parameter list (see a description of IPRCODE below) and no processing occurred.

CC=2
The function has successfully completed with no errors.

CC=3
The function has completed, but an error was detected. In this case, the error code is stored in the IPAUDIT field of the output parameter list. (See a description of IPAUDIT below.) This condition code applies only when WAIT=YES (synchronous processing).

See "Condition Codes and Return Codes for IUCV Macro Functions" on page 522 for completions of IUCV functions on APPC/VM paths.

Return Codes

Four types of return code fields are possible in an APPC/VM environment. The various return codes are stored in either the output parameter list or the function complete interrupt.

Here are the four types of return code fields:

IPRCODE
reports error conditions that CP detects when the function is initiated. IPRCODE is a 1-byte field in the output parameter list. A value is placed into IPRCODE when CC=1. Note that IPRCODEs are often given as a result of issuing a function from the wrong state. (For more information about states, see Table 71 on page 404.)

There is no corrective action for this type of error. You should generally sever the path when you get a nonzero value in this field.

IPAUDIT
reports error conditions that CP detects between the time that the function is initiated and the time that the function completes. These errors relate to data that is being sent between programs. IPAUDIT is a 4-byte field in the output parameter list and function complete interrupt. IPAUDIT is in the output parameter list (when WAIT=YES and CC=3) or in the function complete interrupt (when WAIT=NO and CC=0).

IPWHATRC
contains either a return code or what-received indication caused by your communications partner. IPWHATRC is a 1-byte field in the output parameter list and function complete interrupt.

When IPWHATRC is a what-received indication, IPCODE contains 0 and serves no purpose. IPWHATRC represents a what-received indication when it contains one of the following:

X'01'
Data was received

X'02'
Your partner switched the conversation around

X'04'
Your partner is requesting confirmation

X'05'
Your partner is requesting confirmation that it can issue a SEVER

X'06'
Your partner has confirmed your request

X'0B'
Allocate data was received

X'0C'
Your partner is requesting confirmation that it can enter Receive state

X'0D'
Log data was received

X'0E'
PIP variable was received.

IPWHATRC represents a return code (with IPCODE) when it contains one of the following:

X'03'
Your partner issued SENDERR

X'08'
Your partner issued SEVER TYPE=NORMAL

X'09'
Your partner issued SEVER TYPE=ABEND.

IPCODE
contains the sever or error code caused by your communications partner. IPCODE is a 2-byte field in the output parameter list and function complete interrupt. The value of the IPWHATRC field determines what type of code is in IPCODE. If IPWHATRC is:

- X'09', IPCODE contains a sever code (meaning your partner issued a SEVER TYPE=ABEND). For more information on sever codes, see "APPC/VM Sever, Error, and Sense Codes That You Can Get" on page 399.
- X'03', IPCODE contains an error code (meaning your partner issued a SENDERR). For more information on error codes, see "APPC/VM Sever, Error, and Sense Codes That You Can Get" on page 399.

Virtual MP Considerations for APPC/VM Applications

APPC/VM applications can be written to work in a virtual MP environment. The following list is intended to provide some guidance on using APPC/VM in a virtual MP environment.

- APPC/VM functions may be invoked by any virtual processor in the virtual configuration as long as one of the processors has issued an IUCV DECLARE BUFFER function.
- The IUCV DECLARE BUFFER function defines an interrupt buffer for the virtual processor that invokes it.
- In the virtual MP environment, APPC/VM interrupts are treated as "floating" interrupts. Any virtual processor that has one of the following conditions may receive an APPC/VM interrupt:
  - issued an IUCV DECLARE BUFFER
  - enabled to receive APPC/VM interrupts with the CR0 setting
  - enabled for APPC/VM interrupts with the SETMASK and SETCMASK functions
- The IUCV RETRIEVE BUFFER function will only retrieve the buffer for the currently running virtual processor. APPC/VM paths will not be SEVERed until the last virtual processor issues an IUCV RETRIEVE BUFFER.
- The IUCV SETMASK and SETCMASK functions will apply only to the virtual processor on which they are invoked. This will allow an application to force different types of APPC/VM interrupts to different virtual processors in the complex, if so desired.
- The following are associated with the virtual configuration:
  - APPC/VM directory specifications
  - APPC/VM paths
  - APPC/VM interrupts
  - APPC/VM messages.
- The following are associated with the virtual CPU:
  - the application buffer, the control interrupt buffer, and the interrupt buffer extension
  - interrupt enablement masks in the virtual PSW and virtual control register 0 (bit 30)
  - the interrupt enablement masks of SETMASK and SETCMASK
- The IUCV DESCRIBE, TEST COMPLETE, and IPOLL functions will complete on any processor in the virtual complex as long as one virtual processor has issued an IUCV DECLARE BUFFER (it does not have to be the virtual processor that issued the DESCRIBE, TEST COMPLETE, or IPOLL function).

- If multiple virtual processors in the complex issue the IUCV TEST MESSAGE function, it is unpredictable in which order the virtual processors will be taken out of their wait states.
- All addresses specified with APPC/VM parameter lists are guest absolute addresses.
- Without appropriate guest operating system support, it is difficult or impossible to use APPC/VM in a virtual MP environment. This support would allow your application to:
  - declare buffers on different processors
  - enable for APPC/VM interrupts on the needed processors
  - handle the interrupts and route them to the appropriate virtual processor.

  Note that CMS does not currently support APPC/VM virtual MP functions.

APPC/VM Sever, Error, and Sense Codes That You Can Get

After issuing an APPCVM macro function, your program could get a special problem code returned in bytes 4 and 5 (the IPCODE field) of the output parameter list/function complete interrupt. This 2-byte code is a sever, error, or sense code, originating from your communications partner, intermediate communications server, or VM system.

Sever, error, and sense codes can be reported on completion of the following APPCVM macro functions: CONNECT, RECEIVE, SENDCNF, SENDDATA, or SENDERR.

Currently-Defined APPC/VM Sever Codes

See Table 69 on page 399 for a list of all the APPC/VM sever codes that your application program can get at the current time. Sever codes can come from your communications partner, from your VM system, or from an intermediate communications server. Note that these sever codes can come from VTAM® or other SNA network components. (The sever codes that are generated by VM are listed in Table 70 on page 400 and are a subset of those listed in Table 69 on page 399.)

The corresponding APPC error condition is given for each APPC/VM sever code; refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* for a description of each APPC error condition.

Table 69. Possible APPC/VM Sever Codes

APPC/VM Sever Code	APPC Error Condition
	ALLOCATION_ERROR
X'0110'	ALLOCATION_FAILURE_NO_RETRY
X'0111'	ALLOCATION_FAILURE_RETRY
X'0112'	UNSUCCESSFUL
X'0120'	CONVERSATION_TYPE_MISMATCH
X'0130'	SYNC_LEVEL_NOT_SUPPORTED_BY_PGM
X'0131'	SYNC_LEVEL_NOT_SUPPORTED_BY_LU
X'0140'	TRANS_PGM_NOT_AVAIL_NO_RETRY
X'0141'	TRANS_PGM_NOT_AVAIL_RETRY
X'0142'	TPN_NOT_RECOGNIZED
X'0150'	PIP_NOT_SPECIFIED_CORRECTLY
X'0151'	PIP_NOT_ALLOWED
X'0160'	SECURITY_NOT_VALID
X'0210'	DEALLOCATE_ABEND_PROG
X'0220', X'0221', X'0222'	DEALLOCATE_ABEND_SVC

Table 69. Possible APPC/VM Sever Codes (continued)

APPC/VM Sever Code	APPC Error Condition
X'0230'	DEALLOCATE_ABEND_TIMER
X'0301' X'0302'	PARAMETER_ERROR Invalid LU name Invalid mode name
X'0610' (Note 1)	RESOURCE_FAILURE_NO_RETRY or ALLOCATION_FAILURE_NO_RETRY
X'0620' (Note 2)	RESOURCE_FAILURE_RETRY or ALLOCATION_FAILURE_RETRY

Notes:

1. X'0610' resulting from APPCVM CONNECT corresponds to ALLOCATION_FAILURE_NO_RETRY; from all other functions, X'0610' corresponds to RESOURCE_FAILURE_NO_RETRY.
2. X'0620' resulting from APPCVM CONNECT corresponds to ALLOCATION_FAILURE_RETRY; from all other functions, X'0620' corresponds to RESOURCE_FAILURE_RETRY.

Sever Codes Generated by VM

Your program can get sever codes generated from the CP, CMS, TSAF, or AVS components on VM. See Table 70 on page 400 for a list of these sever codes and some possible causes for each sever code. See Table 69 on page 399 for the APPC error condition that corresponds to each of these APPC/VM sever codes.

Your program can also get sever codes from AVS that AVS is just passing along from VTAM or other SNA network component. See Table 69 on page 399 for a list of sever codes you can receive.

Note: It is important to note that the possible causes shown for each code may not be an exhaustive list.

Table 70. Sever Codes Generated by VM

APPC/VM Sever Code	Possible Causes
X'0110'	• The local AVS had problems receiving PIP data on an incoming APPC/VM connection. • The local AVS received a storage error from VTAM or some other VTAM problem occurred. • The communications server unsuccessfully issued an APPCVM CONNECT with PIP data resulting in an IPRCODE. • The communications server issued an APPCVM CONNECT with PIP data but a nonzero return code was reflected in IPAUDIT.
X'0111'	• A completion time-out.
X'0131'	• The remote LU does not support connections with the specified synchronization level.
X'0140'	• Your program is not authorized to make the connection. • Your program tried connecting to a private resource manager program, but the private server virtual machine either had SET SERVER OFF or SET FULLSCREEN ON. • TSAF did not have authorization to make a connection on behalf of your program.

Table 70. Sever Codes Generated by VM (continued)

APPC/VM Sever Code	Possible Causes
X'0141'	• The target program has not issued HNDIUCV SET (or IUCV DCLBFR). • The target server virtual machine has exceeded its maximum number of connections. • The TSAF virtual machine exceeded its maximum number of connections.
X'0142'	• The TSAF virtual machine or target server virtual machine does not exist, or TSAF cannot find the resource owner. • The target local or global server virtual machine is not logged on. • The target private servers virtual machine cannot be autologged. • The target resource manager did not issue an HNDIUCV SET for the resource ID (program name). • For private resource manager programs, the resource ID was not registered in the private server's $SERVER$ NAMES file. • The CMS-invoked routine in a private server (specified on the :module. tag in $SERVER$ NAMES) is unknown.
X'0151'	• You tried to connect to your partner with PIP data, but, the communications server on the target system is running on a back level CP that does not support PIP data.
X'0160'	• The security information specified on your connection is invalid. • The user ID requesting to connect to a private resource (IPVMID in the target's connection pending interrupt) is not authorized in the private server's $SERVER$ NAMES file.
X'0220'	• IUCV SEVER was issued on a conversation established with SYNCLVL=SYNCPT, and the sync-point-in-progress flag was off. • A SEND or SENDCNF TYPE=PREPRECV was not accepted by VTAM and the conversation is not in a sync-point.
X'0221'	• CP detected an inbound protocol violation during a sync-point. CP reports the error to AVS using CC=1.
X'0222'	• AVS issues this code on an inbound protocol violation during a sync-point when there is a problem receiving log data from VTAM.

Table 70. Sever Codes Generated by VM (continued)

APPC/VM Sever Code	Possible Causes
X'0610'	• ALLOCATION_FAILURE_NO_RETRY – The AVS had problems receiving PIP data from VM or VTAM. – The AVS could not obtain storage for the PIP data. – The remote AVS unsuccessfully issued APPCVM CONNECT with PIP data resulting in an nonzero IPRCODE. – The HNDIUCV SET for CMS failed during IPL. – There was a problem with your program's call to the DMSNAM module (for communications directory resolution). – There was insufficient storage during the connection pending processing. – Your partner program issued an HNDIUCV CLR for a resource ID (program name), but there are still active paths associated with that name. – A CMS abend occurred. – CMS had problems receiving PIP data on an incoming APPC/VM connection. As a result, CMS issued an IUCV SEVER. The problem could be: - CMS had problems receiving PIP data into CMS storage. - CMS could not obtain storage for the PIP data. – The communications server issued APPCVM CONNECT with PIP data but a nonzero return code was reflected in IPAUDIT2. – An error occurred that caused CMS to issue an IUCV SEVER. • RESOURCE_FAILURE_NO_RETRY – IUCV RTRVBFR or HNDIUCV CLR for the resource ID (program name). – IUCV SEVER was issued on a conversation that had SYNCLVL=NONE or SYNCLVL=CONFIRM. – IUCV SEVER issued immediately after issuing IUCV ACCEPT. – A command (a re-IPL, for instance) that caused CP to issue an IUCV SEVER, IUCV RTRVBFR, or HNDIUCV CLR. – The remote AVS virtual machine issued a VTAM function that failed, causing AVS to reject the VTAM conversation.
X'0620'	• RESOURCE_FAILURE_RETRY – The TSAF virtual machine encountered a problem during its processing, or the TSAF link went down. – IUCV SEVER was issued on a conversation with SYNCLVL=SYNCPT, and the sync-point-in-progress flag was on. • ALLOCATION_FAILURE_RETRY – APPCVM CONNECT with PIP data was issued and TSAF on the remote system specified PIP incorrectly on its connection to the target application. (The remote TSAF will abend in this situation.)

Currently-Defined Error Codes

The following table summarizes the error codes that you can get in an APPC/VM program.

For each APPC/VM-defined error code shown in this section, the APPC error condition is given. For more information of each APPC error condition, see *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

APPC/VM Code	APPC Error Condition
X'0410'	PROG_ERROR_NO_TRUNC
X'0420'	PROG_ERROR_TRUNC
X'0430'	PROG_ERROR_PURGING
X'0510'	SVC_ERROR_NO_TRUNC
X'0520'	SVC_ERROR_TRUNC
X'0530'	SVC_ERROR_PURGING

Currently-Defined Sense Code

The following table shows the sense code that you can get in an APPC/VM program. Refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, for a description of the APPC BACKOUT condition.

APPC/VM Code	APPC Error Condition
X'0824'	BACKOUT

The *z/VM: CMS Application Development Guide for Assembler* contains programming information, scenarios, and sample programs that illustrate how to write APPC/VM programs using these APPCVM macro functions.

State Table for APPC/VM Functions

The basic states for APPC/VM assembler programs are:

Reset
The state for each program before communications begin and after communications end

Connect
The state for a source program after a connection has been started but before it has completed, or the state for a target program after it has received a connection pending interrupt but before it has accepted.

Send
The state in which a program is allowed to send data

Receive
The state in which a program is ready to receive data

Confirm
The state in which a program must respond to its communications partner

Sever
The state a program is in when its partner stops communications.

These APPC/VM states are based on the states that APPC defines, but there are two differences:

- The Connect state is unique to APPC/VM
- The Sever state is analogous to the APPC Deallocate state.

The following table summarizes general APPC/VM functions that can be issued from the basic APPC/VM states.

Table 71. APPC/VM States

State	When the State Occurs	Functions You Can Issue
Reset	• Before the program sets up a path. • Before the connection pending interrupt is accepted. • After the program issues a SEVER.	APPCVM CONNECT IUCV SEVER KEEP=NO APPCVM QRYSTATE
Connect	• After the program issues CONNECT, but before the CONNECT completes. • After the program receives a connection pending interrupt, but before it invokes an ACCEPT for the connection.	IUCV SEVER APPCVM RECEIVE IUCV ACCEPT APPCVM QRYSTATE
Send	• After the CONNECT completes. • After you receive notice that your communications partner issued RECEIVE or SENDDATA RECEIVE=YES. • When your SENDCNF TYPE=NORMAL is completed by your partner's SENDCNFD. • After you issue SENDCNFD in response to your partner's SENDCNF TYPE=PREPRECV. • After SENDERR completes normally.	• APPCVM SENDDATA • APPCVM SENDCNF • APPCVM RECEIVE • APPCVM SENDERR • APPCVM SENDREQ (if target is not accessed through AVS) • APPCVM SEVER TYPE=NORMAL, or APPCVM SEVER TYPE=ABEND • IUCV SEVER • APPCVM QRYSTATE • APPCVM SETMODFY
Receive	• After the program issues ACCEPT for a connection. • After RECEIVE completes. • After any function completes, and you receive notice that your partner has issued a SENDERR. • After you issue SENDCNFD, in response to your partner's SENDCNF TYPE=NORMAL. • When your SENDCNF TYPE=PREPRECV is completed by your partner's SENDCNFD.	APPCVM RECEIVE APPCVM SENDERR APPCVM SENDREQ APPCVM SEVER TYPE=ABEND IUCV SEVER APPCVM QRYSTATE
Confirm	• After the program receives a confirmation request (APPCVM SENDCNF function) from its communications partner.	APPCVM SENDCNFD APPCVM SENDERR APPCVM SENDREQ APPCVM SEVER TYPE=ABEND IUCV SEVER APPCVM QRYSTATE

Table 71. APPC/VM States (continued)

State	When the State Occurs	Functions You Can Issue
Sever	• After a SEND or RECEIVE completes with an indication that your communications partner issued a SEVER. • After your SENDCNF TYPE=SEVER is completed by your partner's SENDCNFD. • After you issue SENDCNFD, in response to your partner's SENDCNF TYPE=SEVER.	APPCVM SEVER TYPE=NORMAL IUCV SEVER APPCVM RECEIVE APPCVM QRYSTATE

In addition, APPC/VM defines several other states for special programs involved with coordinated resource recovery. The additional states for APPC/VM assembler programs are:

Defer_Receive
The state a program is in after issuing SETMODFY TYPE=RECEIVE.

Defer_Sever
The state a program is in after issuing SETMODFY TYPE=SEVER.

Prepare_Receive
The state a program is in when a RECEIVE or SENDDATA RECEIVE=YES function completes with IPWHATRC=IPPREPAR, an indication that the partner initiated a commit sync-point.

Unsolicited_Request_Commit_Received
The state a program is in when a RECEIVE or SENDDATA RECEIVE=YES function completes with IPWHATRC=IPREQCOM, an indication that the partner initiated a commit sync-point.

Backout_Received
The state a program is in when a function completes with IPWHATRC=IPBACK, an indication of a backout sync-point.

Backout_Required
The state a program is in when CMS is backing out the CMS work unit.

The following table summarizes these states:

Table 72. APPC/VM States for Coordinated Resource Recovery

State	When the State Occurs	Functions You Can Issue
Defer_Receive	• When SETMODFY TYPE=RECEIVE completes.	APPCVM QRYSTATE APPCVM SENDCNF TYPE=PREPRECV APPCVM SENDDATA RECEIVE=NO, FLUSH=YES APPCVM SEVER TYPE=ABEND IUCV SEVER
Defer_Sever	• When SETMODFY TYPE=SEVER completes.	APPCVM QRYSTATE APPCVM SEVER TYPE=ABEND IUCV SEVER
Prepare_Received	• When a function completes with IPWHATRC=IPPREPAR, which is an indication that your partner initiated a CRR commit sync-point.	APPCVM QRYSTATE APPCVM SENDERR APPCVM SENDREQ APPCVM SEVER TYPE=ABEND IUCV SEVER

Table 72. APPC/VM States for Coordinated Resource Recovery (continued)

State	When the State Occurs	Functions You Can Issue
Unsolicited_ Request_ Commit_Received	• When a RECEIVE or SENDDATA RECEIVE=YES function completes with IPWHATRC=IPREQCOM, which is an indication that your partner initiated a CRR commit sync-point.	APPCVM QRYSTATE IUCV SEVER APPCVM SENDERR APPCVM SENDREQ APPCVM SEVER TYPE=ABEND
Backout_Received	• When a function completes with IPWHATRC=IPBACK, which is an indication that your partner initiated a CRR backout sync-point.	APPCVM QRYSTATE APPCVM SENDCNFD IUCV SEVER APPCVM SEVER TYPE=ABEND
Backout_Required	• When CMS is backing out the CMS work unit.	APPCVM QRYSTATE IUCV SEVER APPCVM SEVER TYPE=ABEND

Examples of Basic States

The following examples show one way that you can get into each of the APPC/VM basic states.

Reset State: After the path is established,

USER1:	USER2:
• issues APPCVM SEVER and receives CC=2	

At this point, USER1 is in Reset state.

Connect State: After IUCV DCLBFR and an identification of a resource by USER2 has already been performed,

USER1:	USER2:
• issues APPCVM CONNECT and receives CC=0	
	checks the connection pending interrupt

At this point, USER1 and USER2 are in Connect state.

Send State: After IUCV DCLBFR and an identification of a resource by USER2 has already been performed,

USER1:	USER2:
• issues APPCVM CONNECT and receives CC=0	
	• checks the connection pending interrupt
	• issues ACCEPT and receives CC=0

- checks the connection complete interrupt

At this point, USER1 is in Send state.

Receive State: After IUCV DCLBFR and an identification of a resource by USER2 has already been performed,

USER1:	USER2:
• issues APPCVM CONNECT and receives CC=0	
	• checks the connection pending interrupt • issues ACCEPT and receives CC=0
• checks the connection complete interrupt	

At this point, USER2 is in Receive state.

Confirm State: After a path is established with SYNCLVL=CONFIRM,

USER1:	USER2:
• issues APPCVM SENDCNF and receive CC=0	
	• checks the message pending interrupt • issues RECEIVE and receives CC=2

At this point, USER2 is in Confirm state.

Sever State: After a path is established,

USER1:	USER2:
• issues APPCVM SEVER and receives CC=2	
	• checks the sever interrupt • issues RECEIVE and receives CC=2

At this point, USER2 is in Sever state.

State Table for Error Conditions

If you issue a function from the wrong state, you receive an IPRCODE. The IPRCODE tells you the error condition that CP detected when the function was initiated. The IPRCODE you receive depends on the function you issued and the state from which you issued it.

The following table lists the basic APPC/VM functions and the IUCV functions that you can use in APPC/VM. The numbers across the top of Table 73 on page 408 correspond to the following states or conditions:

Number
State or Condition

1
Reset

2
Connect

3
Send

4
Receive

5
Confirm

6
Sever

7
Defer_Receive

8
Defer_Sever

9
Prepare_Received

10
(For IBM use only)

11
Unsolicited_Request_Commit_Received

12
(For IBM use only)

13
Backout_Received

14
Backout_Required

15
(For IBM use only)

16
(For IBM use only)

17
Synchronization level is not SYNCPT.

The following matrix shows which APPC/VM functions can be invoked from each defined state. The number in the box indicates the IPRCODE (in decimal) given when a function is issued from a particular state. An *X* in the box indicates that no IPRCODE is issued because of the function issued from that particular state. If a number and an *X* appear in a box then the IPRCODE is conditional.

Table 73. Error Conditions

Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**11**	**13**	**14**	**15**	**16**	**17**
APPCVM CONNECT	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
APPCVM RECEIVE	68	32X	X	X	35	36X	69	70	71	73	75	76	81X	X	X
APPCVM SENDCNF	68	32	X	34	35	36	69X	70	71	73	75	76	81	79X	X
APPCVM SENDCNFD	68	32	33	34	X	36	69	70	71	73	X	76	X	83X	X
APPCVM SENDDATA	68	32	X	34	35	36	69X	70	71	73	75	76	81	X	X
APPCVM SENDERR	68	32	X	X	X	36	69	70	X	X	75X	76	81	X	X
APPCVM SENDREQ	68	32	X	X	X	36	69	70	X	X	75	76	81	X	X

Table 73. Error Conditions (continued)

Function	1	2	3	4	5	6	7	8	9	11	13	14	15	16	17
APPCVM SEVER TYPE=ABEND	68	32	X	X	X	36X	X	X	X	X	X	X	X	X	X
APPCVM SEVER TYPE=NORMAL	68	32	X	34	35	X	69	70	71	73	75	76	81	X	X
IUCV ACCEPT	68	X	01	01	01	01	01	01	01	01	01	01	X	X	X
IUCV CONNECT	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV DCLBFR	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19
IUCV DESCRIBE	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV IPOLL	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV QUERY	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV RTRVBFR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV SETCMASK	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV SETMASK	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV SEVER	68X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV TESTCMPL	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
IUCV TESTMSG	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Chapter 7. APPCVM Macro Functions

This chapter describes in detail the following APPCVM macro functions:

- CONNECT
- QRYSTATE (Query State)
- RECEIVE
- SENDCNF (Send Confirm)
- SENDCNFD (Send Confirmed)
- SENDDATA
- SENDERR (Send Error)
- SENDREQ (Send Request)
- SETMODFY
- SEVER.

If you are unfamiliar with reading syntax diagrams, see "Syntax, Message, and Response Conventions" on page xxxv.

Using the Online HELP Facility for APPCVM Functions

You can receive online information about the APPCVM macro functions by using the z/VM HELP Facility. For example, to display a menu of the APPCVM macro functions, enter:

```
help appcvm menu
```

To display information about a specific APPCVM macro function (CONNECT in this example), enter:

```
help appcvm connect
```

For more information about using the HELP Facility, see *z/VM: CMS User's Guide*. To display the main HELP task panel, enter:

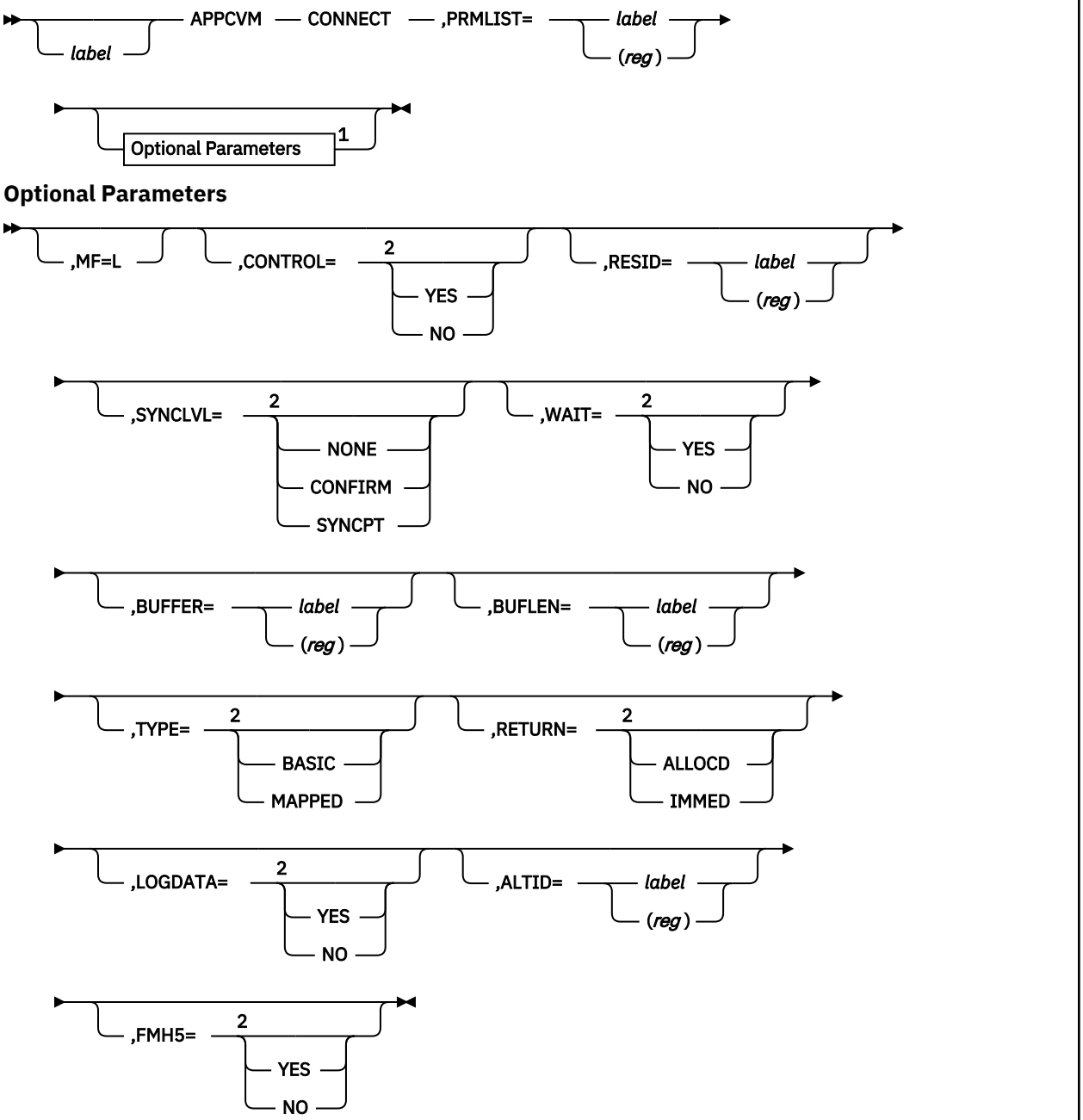
```
help
```

For more information about the HELP command, see *z/VM: CMS Commands and Utilities Reference* or enter:

```
help cms help
```

© Copyright IBM Corp. 1991, 2020

APPCVM CONNECT



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

Use the CONNECT function to establish a communication path with a program residing in either your own virtual machine, another virtual machine in your TSAF or CS collection, or in an SNA network.

Note: If an external security manager is installed on your system, you may not be authorized to use this function. For additional information, contact your security administrator.

Parameters

Required Parameter:

PRMLIST=
Lets you specify the address of the APPC/VM parameter list. The address must be a guest real address; that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
Is the relocatable label of the parameter list.

(*reg*)
Is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
Generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM CONNECT.

CONTROL=
Lets you specify whether a control path is being established. Control paths allow interrupt information for your half of the path to be placed in the control buffer.

YES
Sends APPC/VM interrupt information on this path to the control buffer.

Note: Do not specify CONTROL=YES in application programs running in CMS; CMS uses control paths.

NO
Sends APPC/VM interrupt information on this path to the application buffer.

RESID=
Lets you specify a 1- to 8-character resource identifier. Your program will be connected to the program that manages the resource. If the resource ID you specify is less than eight bytes, left-justify the value in this field and pad the right with blanks.

label
Is the relocatable label of the storage area that contains the resource ID.

(*reg*)
Is the register number that contains the address of the storage area. This storage area contains the resource ID.

The RESID value you specify here either specifies the target transaction program name (TPN), or it is resolved into a transaction program name and additional allocation data using a CMS communication directory file. When CMS resolves the TPN, CMS replaces the RESID value specified in IPRESID with the first 8 bytes of the TPN.

Notes:

1. The RESID (or the RESID resolved by CMS) must match the TPN. If the TPN is longer than 8 bytes, then just the first 8 bytes must match. If the TPN is less than 8 bytes, then the TPN is padded to the right with blanks for the compare.
2. The RESID value (or resolved TPN) should be the same as the resource manager program name required by the CMS interface to assembler APPC/VM (the NAME parameter on the HNDIUCV and CMSIUCV macros).
3. RESIDs beginning with a period (.). The . character must be reserved for recovery servers.

SYNCLVL=
Lets you specify the synchronization level for the path being established.

NONE
Does not let either communication partner request confirmation (issue SENDCNF or SENDCNFD) on the path this connection is establishing.

CONFIRM
Lets either communication partner request confirmation (issue SENDCNF or SENDCNFD) on the path this connection is establishing.

SYNCPT
Specifies that this path can have SYNCPT synchronization level.

Notes:

1. SYNCPT also allows either communication partner to issue confirmation functions (SENDCNF, SENDCNFD).
2. CP rejects connections attempted with SYNCLVL=SYNCPT if the resource manager is located in a TSAF or CS collection.
3. CMS does not allow SYNCLVL=SYNCPT specified on control paths.

WAIT=
Lets you specify when control is returned to your virtual machine.

YES
Returns control to your virtual machine after the CONNECT completes.

NO
Returns control to your virtual machine as soon as the CONNECT request is initiated. When the CONNECT completes, you are notified by a connection complete interrupt. You can issue any APPC/VM function on any path, except the path that you are trying to establish with the CONNECT; the only function that you can issue on the path you are trying to establish is IUCV SEVER.

BUFFER=
Lets you specify the starting address of the connection parameter list extension. This extension either contains actual allocate data (VM area, FMH5, and VM-defined variable-length section) or information that CP uses to build the allocate data. This buffer address must be a *guest* real address (real to the virtual machine).

label
Is the relocatable label of the storage area that contains the connection parameter list extension.

(*reg*)
Is the register number that contains the address of the storage area. This storage area contains the connection parameter list extension.

If you wish to invoke a connection using communication directory resolution (COMDIR=YES on CMSIUCV CONNECT), you do not need to specify this BUFFER address. However, if you want communication directory resolution without invoking a connection (CMSIUCV RESOLVE), you must specify this BUFFER address.

BUFLEN=
Is a 4-byte field that specifies the length of the area containing the connection parameter list extension.

label
Is the relocatable label of the storage area that contains the length.

(*reg*)
Is the register number that contains the length of the storage area.

If you specify FMH5=NO, you are passing CP information it needs to build the allocate data needed by your communication partner. The value you specify for BUFLEN depends on how you want the connection parameter list extension created.

1. If you are explicitly creating the extension, the valid lengths for BUFLEN are 0, 16, 32, 56, 120, 128, or 160 bytes. If you supply fewer bytes than any of these values, the remaining bytes are

considered to be omitted. When SYNCLVL=SYNCPT and FMH5=NO, or when you use PIP data, BUFLEN must be 160.

2. If you wish to invoke a connection using CMS communication directory resolution (COMDIR=YES on CMSIUCV CONNECT), you do not need to specify BUFLEN.
3. If you want communication directory resolution without invoking a connection (CMSIUCV RESOLVE), BUFLEN must be set to at least 120 bytes.

If you specify FMH5=YES (allowed only for communication servers), this means you are directly passing the allocate data needed by your communication partner. The allocate data consists of a VM area, an FMH5, and a VM-defined variable-length section. In addition, if a communication server is passing a PIP variable, it needs to include the length of the VM communication server area (8-bytes) in the BUFLEN total. As a result, BUFLEN can vary from 43 to 911 bytes for a communication server.

See Considerations for Communications Servers for more details.

TYPE=
lets you specify the conversation type being established by the invoker.

BASIC
indicates that a basic conversation is being allocated.

MAPPED
Indicates that a mapped conversation is being allocated.

Note: It is the programmer's responsibility to format and interpret the data according to the connection type specified here. Refer to “APPCVM SENDDATA” on page 475.

RETURN=
lets you specify whether the SNA communication server should wait for a session to become available or should return immediately if no suitable session is available. This operand applies only to connections outside of a TSAF collection.

ALLOCD
indicates that an SNA session should be allocated for the conversation before control is returned to the invoker.

IMMED
indicates that an SNA session should be allocated for the conversation only if a suitable session is immediately available.

LOGDATA=
lets you specify whether your connecting program receives log data on the path being established.

YES
Indicates that the connecting program receives log data.

NO
Indicates that the connecting program does not receive log data. In this case, CP does not log the data for the program.

Communication Server Parameters:

ALTID=
is the 8-byte user ID of the virtual machine that made the original connection for which the communication server is establishing the path. If the user ID that you specify is less than eight bytes, left-justify the value in this field and pad the right with blanks.

label
Is the relocatable label of the storage area that contains the user ID.

(*reg*)
Is the register number that contains the address of the storage area. This storage area contains the user ID.

Note: Only virtual machines authorized as communication servers can specify ALTID. (Refer to Considerations for Communications Servers.)

FMH5=
indicates whether your program's connection parameter list extension (specified by the BUFFER=keyword) contains actual allocate data or information that CP uses to build the allocate data. Actual allocate data consists of a VM area, an FMH5 (Function Management Header Type 5) area, and a VM-defined variable-length section.

NO
indicates that information in the connection parameter list extension is being supplied so that CP can build the allocate data for you.

You should specify FMH5=NO if you want to invoke the connection using the CMS communication directory (COMDIR=YES on the CMSIUCV CONNECT), or if you just want communication directory resolution without invoking a connection (CMSIUCV RESOLVE). In these two cases, CMS fills in the extension from the information in the communication directory file.

If your program is explicitly supplying the connection extension so that CP can build the allocate data, refer to CONNECT Input Parameter List Extension for a description on how you must set up this extension.

YES
indicates that your program is a communication server, and it is supplying allocate data (a VM area, an FMH5, and a VM-defined variable-length section) and possibly a VM communication server area in the connection parameter list extension. See Considerations for Communications Servers for more information.

Note: If you specify FMH5=YES, CMS communication directory resolution is disabled on any subsequent CMSIUCV CONNECT or CMSIUCV RESOLVE functions.

Input Parameter List: The APPCVM CONNECT parameter list has the input format shown in the following figure when establishing APPC paths:

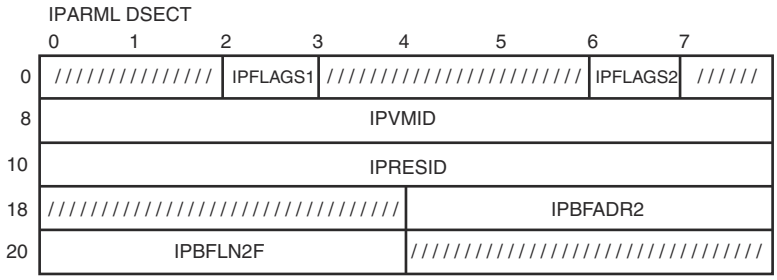


Figure 22. APPCVM CONNECT Input Parameter List

IPFLAGS1
may contain one or more of the following input bit flags:

IPAPPC (X'08')
APPC protocol is used on the path.

IPCNTRL (X'04')
a control path is being established.

X'80'
this value is reserved for IBM use only.

IPFLAGS2
may contain one or more of the following input bit flags:

IPWAIT (X'80')
the connecting program specified WAIT=YES (meaning a synchronous connection).

IPLVLCF (X'40')
a synchronization level of CONFIRM is permitted.

Note: If this bit is on, IPSYNCPT must be off.

IPCOMSRV (X'20')
This connection is being made for another user. This flag is set when the ALTID keyword is specified. See Considerations for Communications Servers.

IPMAPPED (X'10')
A mapped conversation is being created.

IPFMH5 (X'08')
A connection parameter list extension contains the VM area, the FMH5 area, and a VM-defined variable-length section.

IPLOGDOK (X'04')
The connecting program specified LOGDATA=YES.

IPIMMED (X'02')
The connecting program specified RETURN=IMMED.

IPSYNCPT (X'01')
SYNCLVL=SYNCPT is specified for the conversation.

Note:

1. A synchronization level of CONFIRM is also permitted on this conversation.
2. If this bit is on, IPLVLCF must be off.

IPVMID
is the user ID that this connection is made for. Only communication servers can supply this parameter. This field is set from the user ID value specified with the ALTID parameter. See Considerations for Communications Servers.

IPRESID
is the name of the 1- to 8-character resource identifier. Your program is connected to the program that manages the resource. If IPRESID is less than eight bytes, left-justify the value in this field and pad the right with blanks.

IPRESID either specifies the target transaction program name (TPN), or it is resolved into a transaction program name and additional allocation data using a CMS communication directory file. When CMS resolves the TPN, CMS replaces the IPRESID value specified in IPRESID with the first 8 bytes of the TPN.

Notes:

1. The IPRESID (or the IPRESID resolved by CMS) must match the TPN. If the TPN is longer than 8 bytes, then just the first 8 bytes must match. If the TPN is less than 8 bytes, then the TPN is padded to the right with blanks for the compare.
2. IPRESID (or resolved TPN) should be the same as the resource manager program name required by the CMS interface to assembler APPC/VM (the NAME parameter on the HNDIUCV and CMSIUCV macros).

IPBFADR2
is the address of the connection parameter list extension.

IPBFLN2F
is the length of the connection parameter list extension.

For a communication server (where FMH5=YES), this must include the allocate data length and, if applicable, the length of the VM communication server area. (See Considerations for Communications Servers.)

Input Parameter List Extension: If your program specifies FMH5=NO on APPCVM CONNECT, it wants CP to build the allocate data (the VM area, the FMH5 area, and the VM-defined variable-length section) that your communication partner needs. In this case, your program can supply CP with the connection parameter list extension data shown in this section.

Does Your Program Need to Make This Extension?

Your program **does not** have to actually build a connection parameter extension as shown in this section if any of the following are true:

1. Your program is an intermediate communication server that specifies FMH5=YES on the connection. (Refer to Considerations for Communications Servers.)
2. Your program does not specify a PIP variable and requests a connection using CMS communication directory resolution (COMDIR=YES on CMSIUCV CONNECT) to resolve the RESID on APPCVM CONNECT. In this case, CMS provides an extension for you; your program's APPCVM CONNECT does not have to specify BUFFER or BUFLEN.
3. Your program just requests to resolve RESID using the CMS communication directory, without invoking a connection (CMSIUCV RESOLVE). In this case, CMS fills in the extension for you; your program's APPCVM CONNECT must specify a BUFFER and a BUFLEN of at least 120 bytes.

If your program is building its own connection parameter list extension, it should include a USING statement for the IPARMLX COPY file and define proper storage for this file. (The IPARMLX COPY file is contained in the HCPGPI MACLIB.) IPARMLX contains labels that your program can refer to in the connection parameter list extension.

Certain fields of the connection parameter list extension can be omitted. There are two ways you can indicate this, you can specify a:

- Field as binary zeros.
- Connection parameter list extension length (BUFLEN), which is less than the displacement of that field into the extension. For example, if you specify a BUFLEN of 0, all fields are considered to be omitted. If you specify a BUFLEN of 16, the first 16 bytes of the extension are used, and the fields in the remaining bytes are considered to be omitted. Note that the length must still be a valid lengths for the BUFLEN parameter. See the description of BUFLEN.

The format of the connection parameter list extension is shown in the following figure.

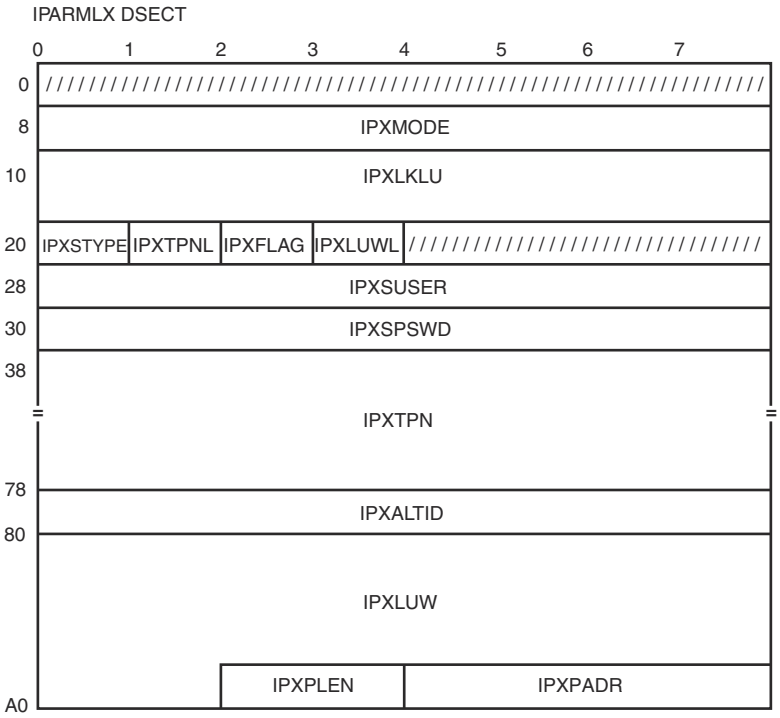


Figure 23. Connection Parameter List Extension

IPXMODE
contains the **mode name**. This is an 8-byte field, left-justified and padded with blanks as necessary.

When the target LU name, IPXTRGLU of IPXLKLU, is omitted (a connection within a TSAF or CS collection), the mode name identifies the type of communication server being used:

- VMINT or binary zeros (the default) identifies an interactive communication server.
- VMBAT identifies a batch-oriented communication server. A path with a mode name of VMBAT gives messages or files a lower priority than messages or files over VMINT or regular TSAF virtual machine session paths.

If an outbound connection through AVS is being requested and the LU name qualifier (gateway name) is specified in the connection parameter list extension, the mode name can be any mode name that is valid for the locally known LU name specified in IPXLKLU. Information on mode name is in *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

IPXLKLU
contains the **locally known LU name**. The locally known LU name is a 16-byte field that is made up of these two pieces:

- IPXLQUAL—an 8-byte **LU name qualifier**
- IPXTRGLU—an 8-byte **target LU name**.

Both these 8-byte fields are left-justified and padded with blanks as necessary.

The meaning of the target LU name depends on the LU name qualifier and the type of connection:

- *Connection to a local, system, or global resource:* If the LU name qualifier is *IDENT, the resource manager program (IPRESID) has already identified itself to the TSAF or CS collection as a local, global, or system resource using the Identify system service (*IDENT). The connection is routed to the local, global, or system resource manager program. If the LU name qualifier is omitted, then it is assumed to be *IDENT. The target LU name must be omitted if the LU name qualifier is *IDENT or is omitted.
- *Connection to a private resource:* If the LU name qualifier is *USERID, then the target LU name is the user ID of the private server virtual machine. The connection is routed to the private resource manager program (IPRESID), which is located in the virtual machine named in the target LU name field.

  Note: Private resource connections to recovery servers (IPRESID is X'06F2', or whose first character is a period .) are treated differently.
- *Connection to a specific system:* If the LU name qualifier is a system gateway name of a system in the TSAF or CS collection, then the connection is routed to this specific system. If the target LU name is binary zeros, then the target system routes the connection to either the system or global resource manager that identified the resource specified in the IPRESID field. If the target LU name is not binary zeros, then the target system routes the connection to the private resource manager whose user ID is the target LU name. The connection is only completed if the resource manager resides on the specified system.
- *Connection to a resource in the SNA network:* The LU name qualifier is a gateway name. The target LU name is the name of an LU in the SNA network, and resource name (IPRESID) is the name of a transaction program at the remote LU. If the gateway name has already been identified to the Identify system service (*IDENT), then the connection is routed to the specified gateway, and the target LU name is presented to the AVS virtual machine that defined the gateway. The gateway uses the target LU name to allocate the conversation for the invoker of the connect.

The connection is rejected if the LU name qualifier is one of the following:

- A gateway name, and that gateway is not owned by a virtual machine
- *IDENT, and the resource is not owned by a virtual machine.

For a summary of the connections, given a LU name qualifier and target LU name, see Table 74 on page 420.

Table 74. Summary of Locally Known LU Names

Locally Known LU Name		IPRESID Value	Connection Made To:
LU Name Qualifier	**Target LU Name**		
*IDENT	Omitted	*X*	Local, global, or system resource *X*
*USERID	*userid*	*Y*	Private resource *Y*
G	*L*	*Z*	Transaction program *Z* residing at LU *L* accessed through gateway *G*
SYSGATE	Omitted	*X*	A global or system resource *X* on the system in the TSAF or CS collection that owns the system gateway, SYSGATE.
SYSGATE	*userid*	*Y*	A private resource *Y* on the system in the TSAF or CS collection that owns the system gateway, SYSGATE.

IPXSTYPE
indicates the **access security type** of the connection.

IPXSSAME (X'00')
SECURITY(SAME) is specified.

The target of the connection receives a connection pending interrupt with the source user ID in the IPVMID field. The source user ID is one of the following:

- The logon user ID of the virtual machine issuing the CONNECT
- An alternate user ID specified by a virtual machine issuing DIAGNOSE code X'D4'
- The user ID specified on the ALTID keyword (from intermediate servers)

If the target LU is a TSAF collection, the source user ID is sent to the target program in the connection pending interrupt and the already verified flag set in the FMH5. If the target LU is in the SNA network, the source user ID is sent to the target LU with the already verified flag set in the FMH5. If the target LU does not support the already verified option, the local LU downgrades the security information to the equivalent of SECURITY(NONE).

IPXSNONE (X'01')
SECURITY(NONE) is specified.

The target of the connection receives a connection pending interrupt with binary zeros in the IPVMID field and an FMH5 with no security fields.

1. If the LU name qualifier is *IDENT, the connection is rejected, unless the resource specified that it would receive connections with SECURITY(NONE) when it connected to *IDENT.
2. If the LU name qualifier is *USERID, the connection is made to the private resource manager, who may or may not reject the connection depending on the CMS authorization list (in the $SERVER$ NAMES file) for that private resource.
3. If the LU name qualifier is not *IDENT or *USERID (meaning a connection to a program in the SNA network), the connection is routed to the specified gateway to be sent to the remote LU.

Note: The following two paragraphs apply to IPXSPGM, IPXSPGLU, and IPXSPGU.

If the LU name qualifier is *IDENT or *USERID, the security fields are validated in the TSAF or CS collection. If the access user ID and password are valid, the access user ID (not the user ID of the virtual machine issuing the CONNECT) is specified as the user ID in the IPVMID field of the connection pending interrupt and in the FMH5. The virtual machine issuing the CONNECT must have directory authorization to connect to the target virtual machine; however, when the

connection is to a private resource and the user ID of the target private server is the same as the access user ID, no authorization is needed.

If the LU name qualifier is not *IDENT or *USERID, the remote LU validates fields in the security area. If the remote LU is a TSAF or CS collection, the collection validates the access user ID and password, the target program gets an FMH5 containing the user ID and the already verified flag set.

IPXSPGM (X'02')
SECURITY(PGM) is specified; the access user ID and password are both supplied in this extension. (The user ID is in field IPXSUSER; the password is in IPXSPSWD.)

IPXSPGLU (X'03')
SECURITY(PGM) is specified; and the access user ID and password are taken from the invoker's APPCPASS directory statement.

The APPCPASS directory statement used must have a locally known LU name that matches the locally known LU name supplied in the connection parameter list extension (field IPXLKLU). If there are multiple APPCPASS directory statements in the invoker's directory entry with matching LU names, the user ID and password in the first match are used.

IPXSPGUS (X'04')
SECURITY(PGM) is specified; the access user ID is supplied in field IPXSUSER of this extension, and the access password is taken from the invoker's APPCPASS directory statement.

The APPCPASS directory statement used must have a locally known LU name that matches the locally known LU name supplied in field IPXLKLU and a user ID that matches the user ID supplied in field IPXSUSER. If there are multiple APPCPASS directory statements in the invoker's directory entry with a matching locally known LU name and access user ID, then the password in the first match is used.

Note: This lets a single user have multiple user IDs at a given LU, each with a different password.

IPXTPNL
is the **length of the transaction program name** (which is contained in field IPXTPN). The length can be 0 to 64 bytes.

IPXFLAG
may contain the following flag:

IPXPBUFL (X'40')
specifies that the PIP address field (IPXPADR) contains the address of a list of buffer addresses and lengths for the PIP variable. (Refer to Specifying a PIP Variable for more information.)

IPXLUWL
is the length (in bytes) of the logical-unit-of-work identifier fields defined in the IPXLUW field. Values 0 and 10 through 26 are valid. CMS fills in this field for APPC/VM programs that use SYNCLVL=SYNCPT paths.

IPXSUSER
is the **access security user ID**.

This area is valid only when IPXSTYPE is IPXSPGM, IPXSPGLU, or IPXSPGUS (indicating a security type of PGM).

IPXSPSWD
is the **access security password**.

This area is valid only when IPXSTYPE is IPXSPGM, IPXSPGLU, or IPXSPGUS (indicating a security type of PGM).

Note: The target virtual machine never receives the password. When the password is supplied, CP verifies the password and sends the verified user ID to the resource manager program. If the password being verified by CP is LBYONLY then the password validation, and subsequently the CONNECT, will fail. Refer to the USER or IDENTITY directory statement in *z/VM: CP Planning and Administration* for more information about the LBYONLY operand.

IPXTPN
is the **transaction program name** (TPN). This must be a 64-byte field. (IPXTPNL describes the actual length of this name.)

If the TPN length (IPXTPNL) is 0 or omitted, CP places:

- 8 into the transaction program name length field (CPEFTPNL) of the FMH5
- The RESID value from the APPCVM CONNECT into the transaction program name field (CPEFTPN) of the FMH5.

Any value specified in IPXTPN is not used.

If the length is not 0, CP places:

- The value from IPXTPNL into the transaction program name length field (CPEFTPNL) of the FMH5
- The value from IPXTPN into the transaction program name field (CPEFTPN) of the FMH5. Even though CP places IPXTPN into the FMH5, the RESID value on APPCVM CONNECT still determines the target resource.

If IPXTPNL is less than 64 bytes, the user must still supply the full 64-byte transaction program name area. The value of IPXTPNL must equal the value of IPRESID. See "IPRESID" on page 417 for details.

For SYNCLVL=SYNCPT conversations, only the first 24 bytes of the TPN length register the path to the sync-point manager.

IPXALTID
is an alternate user ID that is supplied to the target virtual machine. To specify a nonzero value in this field, a virtual machine must be authorized in the CP directory for DIAGNOSE code X'D4'.

IPXLUW
is the *logical-unit-of-work identifier*. This is a 26-byte field. (IPXLUWL describes the actual length.) CMS fills in this field for APPC/VM programs in CMS for SYNCLVL=SYNCPT conversations only.

IPXPLEN
is the length, in binary, of the number of bytes for the *entire* PIP variable (PIP data and all header information). Valid values are 0, 4, or any value from 8 to 32,767.

IPXPADR
specifies either the address of a PIP variable buffer, or the address of a list that contains addresses and lengths of multiple PIP variable buffers.

Usage Notes

Specifying a PIP Variable: Your program can supply a PIP variable that the target program can receive before accepting your connection. When specifying a PIP variable, your program must supply the following information in the APPCVM CONNECT parameter list extension:

- The length of the PIP variable (field IPXPLEN)
- The address of the PIP variable (field IPXPADR)
- An indication of whether you are specifying the PIP variable with a single buffer using one address and one length, or specifying multiple buffers using a list of addresses and lengths (flag IPXPBUFL in field IPXFLAG).

When you specify a single buffer using one address and one length:

- IPXPADR specifies the address.
- IPXPLEN specifies the length.
- The flag IPXPBUFL in the field IPXFLAG must be turned off.

When you specify multiple buffers with a list of addresses and lengths,

- IPXPADR specifies the address of the list.

- IPXPLEN specifies the sum of the lengths of the buffers in the list.
- The flag IPXPBUFL in the field IPXFLAG must be turned on.

You must follow these conventions when you use address lists:

- The list must begin on a doubleword boundary.
- Each list entry must be two fullwords:
  - The first fullword is the address of that portion of the list.
  - The second fullword is the length of that portion of the list.

When you use an address list, the addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete. Also, APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified (by IPXPLEN).

Note: The data in a PIP variable buffer must not be reused until the connection is completed by the partner's IUCV ACCEPT or rejected by the partner's IUCV or APPCVM SEVER. This is because the data in a PIP variable buffer (specified on APPCVM CONNECT) does not move to the receive area (specified on APPCVM RECEIVE) until APPCVM RECEIVE PIP=YES is issued by the target.

The PIP variable you are sending in buffers must be coded into a PIP general data stream (GDS) variable. A PIP GDS variable has the format shown in Figure 24 on page 423 as defined by SNA LU 6.2:

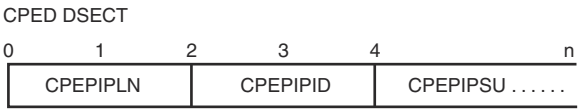


Figure 24. Format of the PIP Variable

CPEPIPLN
the total length in binary, of the PIP variable (including this length field). CPEPIPLN should be equal to the IPXPLEN field specified in the connection parameter list extension.

CPEPIPID
the GDS identifier for the PIP variable, X'12F5'.

CPEPIPSU
zero or more PIP subfields, each of which has the format shown in Figure 25 on page 423.

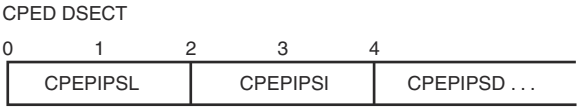


Figure 25. Format of a PIP Subfield

CPEPIPSL
is the length, in binary, of the PIP subfield (including this length field).

CPEPIPSI
is the GDS identifier for a PIP subfield, X'12E2'.

CPEPIPSD
is the actual PIP data.

Considerations for single system image (SSI): When APPCVM CONNECT is used from within an SSI cluster to connect to a private resource, the initial search for the associated userid is performed across the SSI cluster before extending the search across the ISFC Collection.

Examples

Example of Specifying a PIP Variable: A simple example of specifying a PIP variable would be if you wanted to send the letter *Z* to a program with which you are trying to connect. To do this, you could specify the PIP variable as:

```
X'000912F5000512E2E9'
```

- The X'0009' gives the length of the entire PIP variable. This corresponds to the CPEPIPLN field in Figure 24 on page 423.
- The X'12F5' is the GDS identifier for the PIP variable. This corresponds to the CPEPIPID field in Figure 24 on page 423. If you are sending PIP variable, CPEPIPID must always have a value of X'12F5'.
- The X'0005' is the number of bytes in the PIP subfield. This corresponds to CPEPIPSL field in Figure 25 on page 423.
- The X'12E2' is the GDS identifier for the PIP subfield. This corresponds to the CPEPIPSI field in Figure 25 on page 423.

  If you are sending PIP variables with subfields, each subfield must specify a length in the first 2 bytes and the value X'12E2' in the next 2 bytes.
- The X'E9' (Z) is the actual PIP data that we want to send.

The format of this PIP variable would look like what is shown in the following figure.

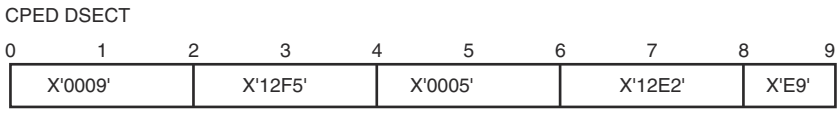


Figure 26. Example Format for a PIP Variable

For more information about the PIP variable, see *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*.

Condition Codes and Return Codes

CC=0
the CONNECT started successfully, but has not completed. IPPATHID is placed in the output parameter list, identifying the path being started by this connection. When the function does complete and your virtual machine is properly enabled for interrupts, you get a connection complete or Sever interrupt. Both the connection complete and the Sever interrupt have the same format as the APPCVM CONNECT output parameter list (see CC=2).

Note: When you specify WAIT=YES, CC=0 is not possible.

CC=1
an error occurred before the CONNECT was initiated. The output parameter list is the same as the input shown in CONNECT Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'0A'	10	The buffer length for the PIP variable is negative.
X'0B'	11	CP could not find the resource or gateway, or, the resource or gateway is not available for connections, and no TSAF virtual machine is currently operating on your system.

Hex Code	Decimal Code	Why the Error Occurred
X'0C'	12	Your communication partner has not invoked the HNDIUCV SET (or IUCV DCLBFR) function.
X'0D'	13	Your virtual machine already has the maximum number of connections.
X'0E'	14	Your communication partner already has the maximum number of connections.
X'0F'	15	• Your virtual machine is not authorized to connect to the resource, or • The request was from a communication server and security label checking is enabled on the system, or • A problem was detected in internal CP control structures. In this case, the error is accompanied by a CP soft abend.
X'1A'	26	The buffer list for the PIP variable is not on a doubleword boundary.
X'1D'	29	You are not authorized to act for another user.
X'27'	39	Your program specified an invalid connection parameter list extension length.
X'28'	40	Your program specified an invalid locally known LU name.
X'29'	41	Your program specified an invalid mode name.
X'2F'	47	Your program specified invalid security fields in the FMH5.
X'31'	49	Your communication partner does not allow connections with SECURITY(NONE).
X'32'	50	Your program specified invalid allocation data. Or, a communication server did not include the length of the VM communication server area in the allocate data length. Refer to Considerations for Communications Servers for more information.
X'34'	52	There is no APPCPASS directory statement.
X'35'	53	Your program specified an invalid transaction program name (TPN) length.
X'36'	54	Your program specified an invalid transaction program name (TPN).
X'38'	56	WAIT=YES was specified on a function issued to this same virtual machine.
X'39'	57	Your program specified an invalid length for the PIP variable. Valid values are 0, 4, or any value from 8 to 32,767.

Hex Code	Decimal Code	Why the Error Occurred
X'3A'	58	Your program specified an invalid length for the VM communication server area. Valid values are 0 and 8.
X'3D'	61	Your virtual machine is not authorized to specify an alternate user ID.
X'3F'	63	Your program specified an invalid logical-unit-of-work identifier length. Valid values are 0 and 10 to 26.
X'40'	64	Your program specified an invalid fully-qualified LU network name in the logical-unit-of-work identifier.
X'41'	65	Your program specified SYNCLVL=SYNCPT, but the logical-unit-of-work identifier length field is zeros.
X'43'	67	Both SYNCLVL=CONFIRM and SYNCLVL=SYNCPT were specified in IPFLAGS2 in the APPCVM CONNECT input parameter list.
X'56'	86	A control buffer must be declared to issue APPCVM CONNECT with SYNCLVL=SYNCPT.
X'59'	89	For your SYNCLVL=SYNCPT connection, CP could not find the resource, or the resource is not available for connections on the local system. Therefore, your SYNCLVL=SYNCPT would have to be routed through a TSAF virtual machine or ISFC, but TSAF and ISFC do not support SYNCLVL=SYNCPT conversations.
X'5C'	92	A paging or storage error was detected.

CC=2 or CC=3
the connect completed. (See CONNECT Completion for more information.) When CC=2, the connect completed with no errors; when CC=3, there is some error information in the IPAUDIT field.

Note: When WAIT=NO, CC=2 or 3 is not possible.

The output parameter list when CC=2 or 3 is shown in Figure 27 on page 426.

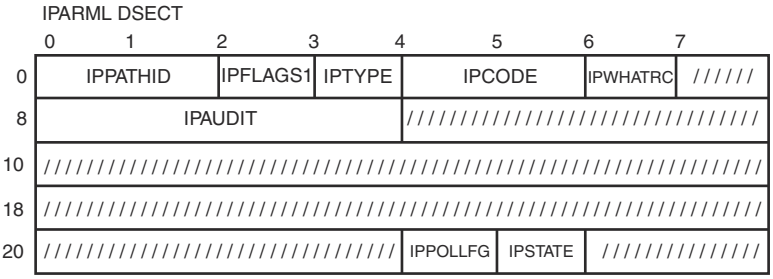


Figure 27. APPCVM CONNECT Output Parameter List (Connection Complete Interrupt)

IPPATHID
contains the path ID on which the connection was completed or severed.

IPFLAGS1
contains one of the following bit flags:

IPCNTRL (X'04')
a connection complete is on a control path. This flag is not set if the external interrupt was caused by a SEVER (IPTYPE=X'83').

IPREMOTE (X'02')
the connection was accepted by a communication server.

IPTYPE
contains one of the following codes:

IPTYPCCA (X'82')
the connection complete interrupt code, indicating your partner or an intermediate communication server accepted the connection.

IPTYPSVA (X'83')
the sever interrupt code, indicating your partner or an intermediate communication server rejected the connection with the SEVER function.

IPCODE
contains the sever code from the partner's SEVER. IPCODE is only valid when the external interrupt was caused by a SEVER (IPTYPE=X'83'). See "APPC/VM Sever, Error, and Sense Codes That You Can Get" on page 399.

IPWHATRC
contains the following what-received code:

IPSABEND (X'09')
your partner issued SEVER TYPE=ABEND.

IPWHATRC is only valid when the external interrupt was caused by a SEVER (IPTYPE=X'83').

IPAUDIT
has four fields that may contain error information caused by a PIP data problem.

Notes: In the following descriptions:

- *PIP area* refers to either a PIP buffer specified by IPXPADR or a PIP buffer that is part of a buffer list.
- *Receive area* refers to either a receive buffer specified directly on APPCVM RECEIVE, BUFFER= or a receive buffer that is part of a buffer list.

IPAUDIT1 (first byte of IPAUDIT)
may contain one of the following bit flags:

IPADSNPX (X'40')
a protection exception occurred on your PIP area.

IPADSNAX (X'20')
an addressing exception occurred on your PIP area.

IPAUDIT2 (second byte of IPAUDIT)
may contain one of the following bit flags:

IPADRCPX (X'80')
a protection exception occurred on your communication partner's receive area for the PIP variable.

IPADRCAX (X'40')
an addressing exception occurred on your communication partner's receive area for the PIP variable.

IPADRLST (X'04')
your communication partner specified an invalid receive buffer list.

IPAUDIT3 (third byte of IPAUDIT)
may contain the following bit flag:

IPADBLEN (X'80')
an invalid length is in your PIP variable buffer list.

IPADBTOT (X'20')
your total PIP data buffer length (CPEPIPLN) is invalid.

IPASYRC (fourth byte of IPAUDIT)
may contain one of the following error codes (for which an appropriate IPRCODE was given to your communication partner):

Hex Code	Decimal Code	Meaning
X'3B'	59	An invalid general data stream ID was specified within the PIP variable.
X'3C'	60	An invalid length was specified for a PIP subfield, or the total length in bytes 0 through 1 of the PIP variable is not the sum of the lengths of the PIP subfields.
X'5B'	91	A paging or storage error occurred on your PIP area.
X'5C'	92	A paging or storage error occurred on your communication partner's receive area for the PIP variable.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
contains the current state for this path, which has one of the following values:

IPSENDST (X'03')
the path is in the Send state.

IPBKREQ (X'0E')
the path is in the Backout_Required state.

IPSTATE is not valid when the connection completes with a sever interrupt (IPTYPE=X'83').

Program Exceptions

The program exceptions for CONNECT are:

Type	Description
Addressing	The parameter list address or connection extension address is outside of the virtual machine.
Operation	Either an external interrupt buffer has not been declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user. This exception also occurs if the connection extension address is fetch-protected.
Specification	The parameter list is not on a doubleword boundary.

Note: APPC/VM does not reflect addressing exceptions or protection exceptions on PIP variable buffers, because your partner cannot have a predefined receive area for the PIP variable. Instead, the errors are reported to your program in the corresponding IPAUDIT flags in the connection complete data.

State Checks and State Changes

There are no state checks for APPCVM CONNECT.

Your virtual machine path is in one of the following states:

- The Connect state for either of the following situations:
  - After you issue the CONNECT, but before your communication partner or an intermediate communication server accepts the connection (CC=0).
  - If your communication partner or an intermediate communication server issues a SEVER when you issue the CONNECT, you get a sever interrupt and no path is established. You remain in the Connect state, and you must issue IUCV SEVER to delete your side of the path.
- The Send state after you receive the connection complete indication.

Completion Conditions

Connection complete data can be in different forms, depending on whether you specify WAIT=YES or WAIT=NO on the APPCVM CONNECT.

If you specify WAIT=YES, and your communication partner or intermediate communication server accepts the connection, you get a connection complete indication (with CC=2 or CC=3). If it severs the connection, you get a sever indication in the connection complete data.

The connection complete data goes to the parameter list that you specified on the APPCVM CONNECT macro. The format of the connection complete data is the same as the output parameter list described under CC=2 in Figure 27 on page 426.

If you specify WAIT=NO, and your communication partner accepts the connection, you get a connection complete interrupt. If your partner severs the connection, you get a sever interrupt.

When you specify WAIT=NO and your virtual machine is enabled for external interrupts, the connection complete data goes to your external interrupt buffer.

The format of the connection complete interrupt data is the same as described under CC=2 in Figure 27 on page 426. All subsequent interrupts for the established path are presented in the same buffer as the connection complete interrupt.

Note: The connection complete interrupt indicates only that your CONNECT has completed and that you are now in the Send state on the path. A connection complete interrupt does not necessarily indicate that the actual target of your CONNECT has issued an ACCEPT for the connection, or even that the target of the CONNECT has been invoked. For example, if your connection goes through AVS, AVS accepts the conversation when it communicates with VTAM.

Connection Complete Extended Data: When an APPC/VM connection complete interrupt is reflected to a virtual machine, the Connection Complete Extended Data (CCED) is reflected in the interrupt buffer extension, if it exists. The CCED DSECT and CCED COPY files are located in the HCPGPI macro library. Note that when running in CMS the interrupt buffer extension is defined by the virtual machine, not by the application.

A program running in CMS can obtain connection complete extended data in two ways:

- After CMSIUCV CONNECT completes with return code 0 (signifying CC=0 from APPCVM CONNECT), and while the user exit (interrupt handler) is being driven for the connection complete interrupt, Register 3 points to the CCED.
- When CMSIUCV CONNECT completes with return code 2 or 3 (signifying CC=2 or CC=3 from APPCVM CONNECT), the connection complete extended data is in the address specified on the CMSIUCV CONNECT BUFFER parameter.

A program running in GCS can obtain connection complete extended data when IUCVCOM CONNECT completes with return code 0 (signifying CC=0 from APPCVM CONNECT), and while the user exit (interrupt handler) is being driven for the connection complete interrupt, Register 3 points to the CCED.

The virtual machine has no way of getting any CCED data that does not fit in the interrupt buffer extension.

Notes:

1. CCED data is returned for all synchronization levels, but it is most useful for SYNCLVL=SYNCPT conversations.
2. You still get CCED data, even if a PIP data problem caused an IPAUDIT error.

The following shows the format of the connection complete extended data.

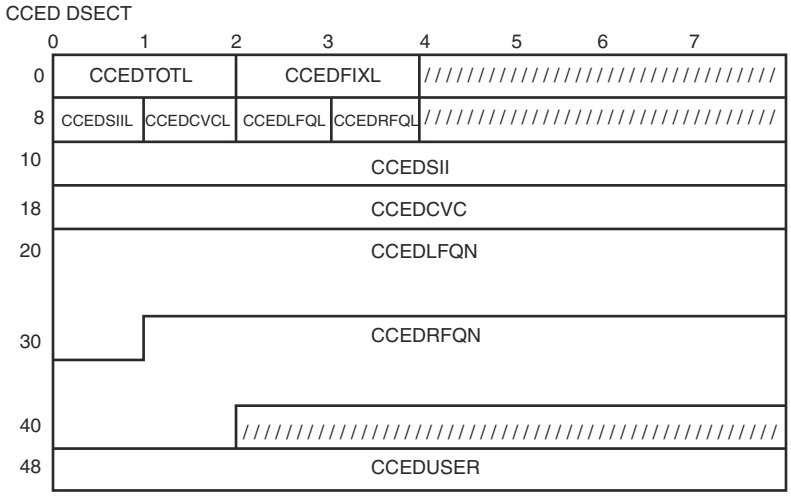


Figure 28. Connection Complete Extended Data

CCEDTOTL
contains the total length of the CCED.

CCEDFIXL
contains the length of the fixed length fields of the CCED (CCEDSIIL through CCEDUSER).

CCEDSIIL
contains the length of the session instance identifier (field CCEDSII). Valid values for the length are 0 and 2 through 8. If the session instance identifier is not known because neither the issuer of CONNECT or ACCEPT specified it, this length field is 0, and CCEDSII is undefined.

CCEDCVCL
contains the length of the conversation correlator (field CCEDCVC). Valid values for the length are 0 through 8. If the conversation correlator is not known because the issuer of CONNECT did not specify it and CP could not generate it, this length field is zero, and CCEDCVC is undefined.

Note: CP does not generate a conversation correlator if the CONNECT was specified with FMH5=YES. This means that if the length of the conversation correlator specified in the FMH5 is zero, the corresponding length in field CCEDCVCL is zero. CCEDCVCL is nonzero in all other cases.

CCEDLFQL
contains the length of the local network fully qualified LU name (field CCEDLFQN). Valid values for the length are 0 through 17. If the local fully qualified LU name is not known because neither the issuer of CONNECT or ACCEPT specified it, this length field is 0, and CCEDLFQN is undefined. (This happens when the target program is on the same system as the source program and AVS is not used.)

CCEDRFQL
contains the length of the remote network fully qualified LU name (CCEDRFQN). Valid values for the length are 0 through 17. If the remote fully qualified LU name is not known because neither the issuer

of CONNECT or ACCEPT specified it, this length field is zero, and CCEDRFQN is undefined. (This happens when the target program is on the same system as the source program and AVS is not used.)

Note: The following applies to all fields that have a corresponding length field in the CCED. If the actual length of the field is less than the maximum length, the data is left-justified within the field and padded on the right with blank characters (X'40'):

CCEDSII
contains the session instance identifier.

If the length CCEDSIIL is zero, this field is undefined and contains binary zeros. This is the case if neither the virtual machine which issued the CONNECT nor the ACCEPT is the SNA communication server.

CCEDCVC
contains the conversation correlator.

If the length CCEDCVCL is zero, the conversation correlator is undefined and contains binary zeros.

CCEDLFQN
contains the local network fully qualified LU name.

If the length CCEDLFQL is zero, this field is undefined and contains binary zeros. This is the case if neither the virtual machine which issued the CONNECT nor the ACCEPT is the SNA communication server.

CCEDRFQN
contains the remote network fully qualified LU name.

If the length CCEDRFQL is zero, this field is undefined and contains binary zeros. This is the case if neither the virtual machine which issued the CONNECT nor the ACCEPT is the SNA communication server.

CCEDUSER
is the access user ID provided for this conversation.

Because the source application does not always specify this user ID, this field allows the source application to know what access security user ID was used when establishing this connection. The value of this field depends on the SECURITY level specified by the invoker of CONNECT.

- When SECURITY NONE was specified, CCEDUSER contains binary zeros. This is the same value that is passed to the target program, assuming that the target program is on a VM system. SECURITY NONE might be presented to the target program differently on non-VM systems. However, the target program will be able to determine that SECURITY NONE was specified.
- When SECURITY PGM was specified, CCEDUSER contains the user ID specified by the invoker. This is the same value passed to the target program.
- When SECURITY SAME was specified, then CCEDUSER generally contains the logon user ID of the invoker. However, there are two exceptions to this:
  1. If the CONNECT was issued by a communication server on behalf of another user (IPCOMSRV is on), CCEDUSER contains the user ID specified on the ALTID parameter of CONNECT.
  2. If the CONNECT was not issued by a communication server on behalf of another user (IPCOMSRV is off), but the invoker has an alternate user ID defined using DIAGNOSE code X'D4', CCEDUSER contains this alternate user ID.

  This is the same value that is passed to the target program if the target program is within a TSAF collection. If the target program is on a VM system outside of the source TSAF collection, this user ID may be mapped to another user ID on the target system.

What Happens to Your VM Communication Partner

This section describes the information presented to your partner program after your program issues APPCVM CONNECT. It includes information on:

- Connection pending interrupt
- Connection pending extended data (allocate data).

Connection Pending Interrupt: When you invoke a CONNECT, your communication partner gets a connection pending external interrupt (assuming it is enabled for interrupts). If you are connecting to a private resource, the connection pending interrupt data goes to the private resource manager's virtual machine control buffer; for all connections to local, global, or system resources, the connection pending external interrupt data always goes to your communication partner's application buffer.

The connection pending external interrupt format is shown in the following figure.

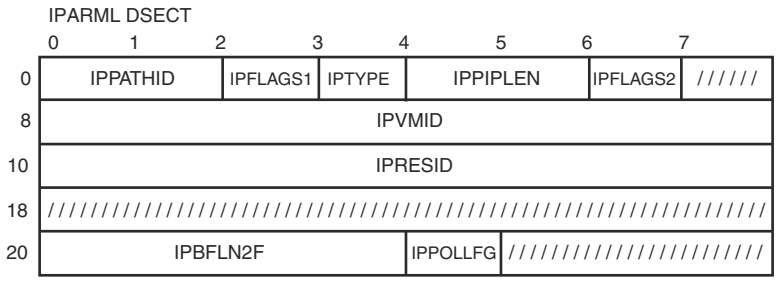


Figure 29. Connection Pending External Interrupt

IPPATHID
contains the path ID on which a connection is pending.

IPFLAGS1
may contain the following bit flag:

IPREMOTE (X'02')
the connection is through ISFC, the TSAF virtual machine, AVS virtual machine, or another communication server that specified a connection using ALTID.

IPTYPE
contains the interrupt type for a connection pending (IPTYPPCA, X'81').

IPPIPLEN
contains the number of bytes (in binary) for the PIP variable. If a PIP variable was not specified on the APPCVM CONNECT, this field is set to zero. If IPPIPLEN is greater than zero, your partner can receive a PIP variable in addition to the allocate data.

Your partner can obtain a PIP variable using APPCVM RECEIVE PIP=YES before accepting the connection with IUCV ACCEPT. See "APPCVM RECEIVE" on page 451 for information about receiving PIP variables. Programs running on CMS and using CMS support for communication do not need to issue a receive to get a PIP variable. CMS receives the PIP variable and places its address in register 4.

IPFLAGS2
may contain one or more of the following bit flags:

IPINVOKE (X'80')
the IPRESID specified is a private resource. The private resource manager program should be invoked if it is not currently active and if the connecting program is authorized in the $SERVER$ NAMES file.

Notes:

1. CMS examines this flag, and when necessary, invokes the appropriate application. Applications not running in the CMS environment can provide this program invocation function if desired.
2. This flag is set when the incoming connection has an LU name qualifier (gateway name) of *USERID. However, do not assume that the connecting application specified *USERID as the gateway name; the SNA communication server (AVS) could have transformed the gateway name originally specified by the connecting application.

IPLVLCF (X'40')
confirmation requests (SENDCNF and SENDCNFD functions) are permitted on this path.

IPMAPPED (X'10')
a mapped conversation is being established.

IPIMMED (X'02')
the connecting program specified RETURN=IMMED.

IPSYNCPT (X'01')
the conversation was established with SYNCLVL=SYNCPT.

IPVMID
contains the source user ID of the virtual machine that wants to connect which may be one of the following:

- The user ID specified with the ALTID parameter of APPCVM CONNECT
- An alternate user ID set by a DIAGNOSE code X'D4'
- The logon user ID of the virtual machine issuing the APPCVM CONNECT
- The access security user ID when a security type of PGM was specified on the APPCVM CONNECT.

The field may be zero, which indicates that the identity of the connecting virtual machine is unknown. This is an 8-byte field, left-justified and padded with blanks as necessary.

IPRESID
for a noncommunication server, this field contains the name of the resource that is the target of the connection (the transaction program name, or TPN).

For a communication server (including ISFC, and the TSAF and AVS virtual machines), this field contains one of the following:

- The name of the target resource, if the connection is to a VM local, global, or system resource. (The LU name qualifier on the connection was *IDENT.)
- The value **USERID*, if the connection is to a VM private resource. (The LU name qualifier on the connection was *USERID.)
- The value **GATEWAY*, if the connection is outbound to a resource in the SNA network. (The LU name qualifier on the connection was the name of an established gateway.)

IPBFLN2F
contains the length of pending allocate data. Refer to Connection Pending Extended Data for a further explanation of pending allocate data.

Note: This allocate data consists of the VM area, the FMH5, and the VM-defined variable-length section only. The length of the PIP variable is contained in the IPPIPLEN field of this interrupt.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

Connection Pending Extended Data (Allocate Data): When your partner is presented with a connection pending interrupt, it is also presented with connection pending extended data (CPED), if the interrupt buffer extension exists. The CPED is referred to as allocate data, which consists of three parts:

- VM architected area
- Attach FMH5 (Function Management Header Type 5)
- VM-defined variable-length section.

The allocate data is created from information in the entire connection parameter list, including the connection parameter list extension.

The CPED (allocate data) currently has a maximum of 919 bytes[3]: 164 bytes maximum length for the VM area, 255 bytes maximum length for the FMH5, and 500 bytes maximum length (NGROUPS_MAX * 4) for the VM-defined variable-length section.

Your partner can obtain the allocate data in two ways:

1. Use APPCVM RECEIVE to receive the data into buffers. If some of the allocate data does not fit in the interrupt buffer extension, the virtual machine must still receive all the allocate data (as indicated by the IPBFLN2F field in the connection pending interrupt), not just the portion of allocate data that fits in the interrupt buffer extension.
2. When CMS (or GCS) drives the user exit with the connection pending interrupt, Register 3 contains the address of the connection pending extended data.

Once a program accepts the connection, the allocate data is purged by CP. If the target of your connection wants to use the allocate data, it must do so before accepting the connection—once the connection is accepted, CP purges allocate data.

The three pieces of allocation data are described in the following sections. Diagrams of these areas are shown in Figure 30 on page 435 and in Figure 31 on page 438.

CPED Part One-VM Area:

[3] There are two exceptions to this:

- When the TSAF virtual machine is handling a connection to a private resource, the VM area is preceded with an 8-byte node ID. Or when a communications server is making a connection with PIP data on behalf of another virtual machine, the FMH5 is followed by the 8-byte VM communications area. As a result, the maximum CPED length is 927.
- When the TSAF virtual machine is handling a connection to a private resource with PIP data, the maximum CPED length will be 935.

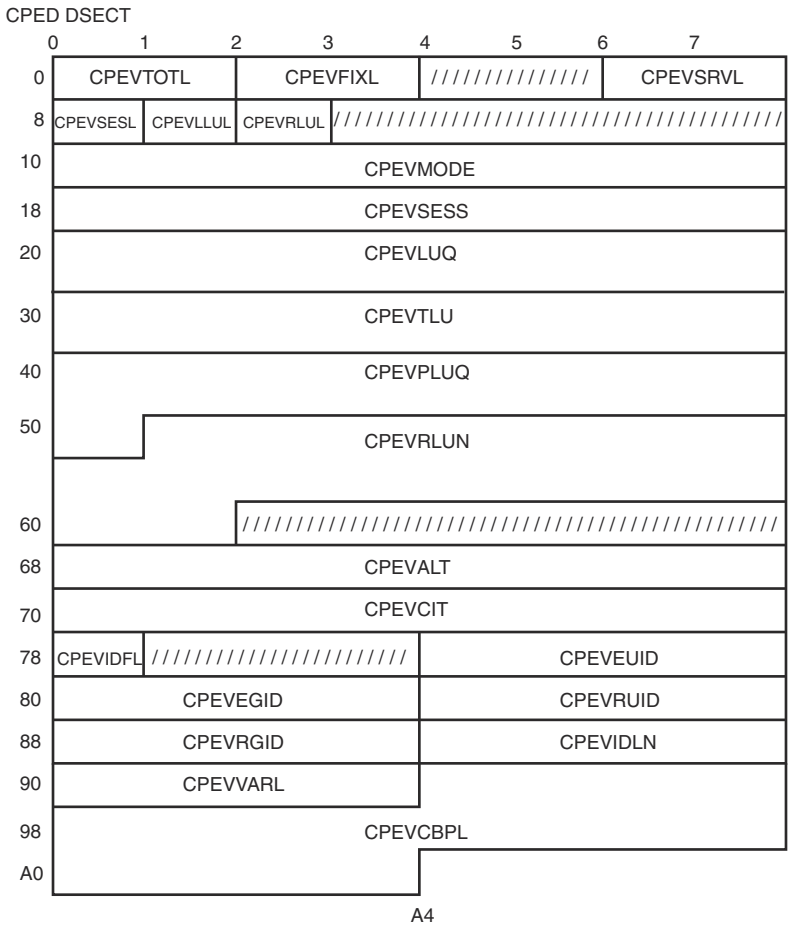


Figure 30. Connection Pending Extended Data, Part One: VM Area

CPEVTOTL
is the total length of the VM architected area, including this length field. CPEVTOTL can have one of the following values: 32, 64, 120, 148, or 164.

CPEVFIXL
is the length of the **fixed length fields** area (the total length of the fields CPEVSESL through CPEVVARL).

CPEVSRVL
contains zeros. (Handling PIP Variables has further information.)

CPEVSESL
is the length of the session instance identifier (field CPEVSESS). Valid lengths are 2 to 8.

CPEVLLUL
is the length of the local fully-qualified LU name (field CPEVLLUN). Valid lengths are 1 to 17.

CPEVRLUL
is the length of the remote fully-qualified LU name (field CPEVRLUN). Valid lengths are 1 to 17.

CPEVMODE
is the mode name.

CPEVSESS
is the session instance identifier. If the connection is being made by a noncommunication server, this field contains binary zeros. Refer to *VTAM Programming for LU 6.2* for more information about the session instance identifier.

CPEVLKL
is the locally known LU name used by the connecting program as its target. This consists of two 8-byte fields:

CPEVLUQ
the LU name qualifier (gateway, name) used on the initial connection.

CPEVTLU
the LU name of the target of the connection.

See Table 74 on page 420 for the LU name qualifiers and target LU names.

CPEVPLKL
is the locally known LU name that the target of the initial connect could use to connect back to the source's LU. (Note that to connect back to the source LU, a program also needs to specify a TPN and also needs to specify *USERID,*userid*. See Table 74 on page 420.)

This consists of two 8-byte fields:

CPEVPLUQ
the LU name qualifier (gateway) that the program uses to connect back to the source's LU.

CPEVPTLU
the LU name of the original source connecting program.

See Table 74 on page 420 for the LU name qualifiers and target LU names.

CPEVLLUN
is the local fully-qualified LU name. If the connection was made by a noncommunication server, this field contains binary zeros.

CPEVRLUN
is the remote fully-qualified LU name. If the connection was made by a noncommunication server, this field contains binary zeros.

See *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2* for details on the local and remote fully-qualified LU names.

CPEVALT
is an alternate user ID the target virtual machine uses. The user ID must be left-justified and padded on the right with blanks as necessary.

This alternate user ID is *assigned* to the target virtual machine when that machine accepts the connection, and it is unassigned when that machine severs the connection. Connection pending interrupts are not presented to a virtual machine that has an alternate user ID assigned; such interrupts are deferred until the alternate user ID has been unassigned. No user ID is assigned if this field contains all zeros.

CPEVCIT
contains the virtual configuration identification token (VCIT) of the virtual machine requesting the connection, if that virtual machine is located in the same system as you are. If the partner virtual machine is on a different system, then this field contains binary zeros.

VCITs are used as an alternative to a user ID as a way of identifying a virtual machine for certain operations, such as the PERMIT function of the CP ADRSPACE macro.

CPEVIDFL
contains access control information. Valid values are:

1... ....
POSIX user ID (UID) and group ID (GID) information has been filled in.

.1.. ....
Reserved

..1. ....
Reserved

...1 ....
Reserved

.... 1...
Reserved

.... .1..
Reserved

.... ..1.
Reserved

....1
Reserved

If bit zero is on,
the access control information (effective and real UIDs, effective and real GIDs, and supplementary GIDs) has been supplied.

All other bits
are reserved.

CPEVEUID
contains one of the following:

- the effective UID of the process that issued the CONNECT
- the database UID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVEGID
contains one of the following:

- the effective GID of the process that issued the CONNECT
- the database GID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVRUID
contains one of the following:

- the real UID of the process that issued the CONNECT
- the database UID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVRGID
contains one of the following:

- the real GID of the process that issued the CONNECT
- the database GID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVIDLN
contains the length of the supplementary GID area, CPESGIDV.

CPEVVARL
contains the length of the VM-defined variable-length section of the CPED.

CPEVCBPL
contains the connect-back partner LU name. This is the LU name that should be used (rather than CPEVPLKL) by the target of the initial connect to connect back to the source's LU. (Note that to connect back to the source LU, a program also needs to specify a TPN. See Table 74 on page 420.) CPEVCBPL consists of two 8-byte fields:

CPEVCBLQ
The LU name qualifier (gateway) that the program uses to connect back to the source's LU.

CPEVCBTL
The LU name of the original source connecting program.

See Table 74 on page 420 for the LU name qualifiers and target LU names.

CPED Part Two—FMH5: APPC (LU Type 6.2) uses an FMH5 to carry a request for a conversation to be established between two transaction programs. This header identifies the transaction program that is the target of a connection and that is invoked. An FMH5 is variable length.

It is possible for your program to receive FMH5 information from the SNA network. This FMH5 may be different from the FMH5 generated by VM. For more information on what you can expect from the SNA network, see *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2* for more information on the FMH5.)

In APPC/VM, you are concerned with a certain type of FMH5—an Attach FMH5. Throughout the rest of this document, we refer to the Attach FMH5 as simply FMH5.

If CP creates the FMH5, it specifies the values shown below. Fields CPEFLEN through CPEFTPN are created for any connection; fields CPEFASIL and the security area are created only for connections with SECURITY(SAME) or SECURITY(PGM).

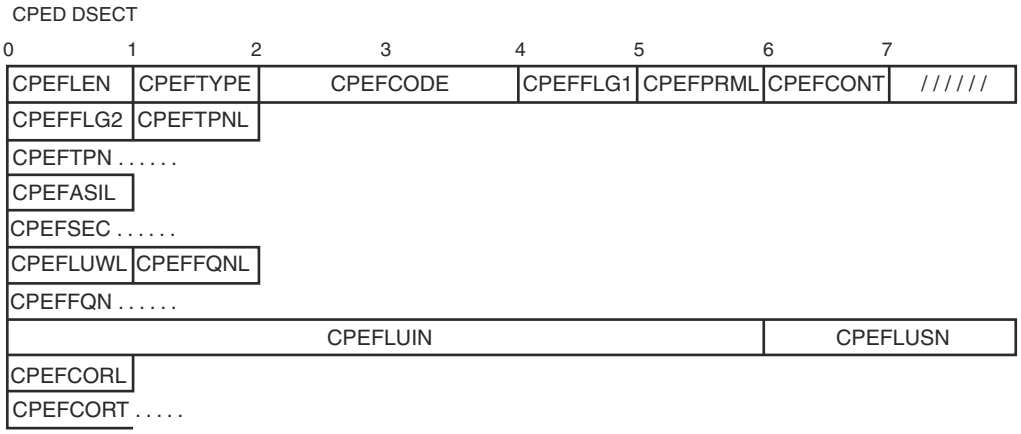


Figure 31. Connection Pending Extended Data, Part Two: FMH5

CPEFLEN
is the total length, in hexadecimal, of the Attach FMH5 (including this length byte).

CPEFTYPE
is the type code for FMH5, X'05'.

CPEFCODE
is the command code for APPC Attach, X'02FF'.

CPEFFLG1
may contain one or more of the following flags:

CPEFSECI (X'80')
indicates that access security user ID is already verified.

CPEFPIP (X'08')
indicates a PIP (program initialization parameter) variable is present.

CPEFPRML
is the total length of the **fixed length parameters** field, currently three bytes.

Note: Programs should be sensitive to the fact that this length could change.

CPEFCONT
contains the conversation type.

CPEFBASC (X'D0')
indicates a basic conversation

CPEFMAPC (X'D1')
indicates a mapped conversation

CPEFFLG2
contains one of the following flags:

CPEFSYNO (X'00')
indicates that no confirmation requests can be issued (SYNCLVL=NONE).

CPEFSYCF (X'40')
indicates that confirmation requests can be issued (SYNCLVL=CONFIRM).

CPEFSYSP (X'80')
indicates that requests for confirmation, taking a synchronization point, or making a backout can be issued (SYNCLVL=SYNCPT).

CPEFTPNL
is the length of the transaction program name.

CPEFTPN
is the transaction program name, which is a variable length.

Note: For connections inbound from an SNA network, CPEFTPN can be from 1 to 64 bytes in length. However, in the case where CPEFTPN is greater than 8, CP uses only the first 8 bytes.

For SYNCLVL=SYNCPT conversions, only the first 24 bytes of the TPN length register the path to the sync-point manager.

CPEFASIL
is the length of the security area (CPEFSEC). CP only creates this for SECURITY(SAME) and SECURITY(PGM).

CPEFSSEC
is a variable-length area that contains one or two subfields of security information, depending on the security level and the destination:

- For a SECURITY(SAME) connection, there will be one subfield—for a user ID.
- For a SECURITY(PGM) connection, there will be two subfields—one for a user ID and one for a password.

The format of a security subfield is shown in the following figure.

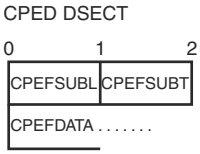


Figure 32. Security Subfield in an Attach FMH5 for VM

The items in a security subfield are as follows:

CPEFSUBL
is the length of CPEFSUBT (one byte) plus the length of CPEFDATA.

CPEFSUBT
is a flag indicating the contents of CPEFDATA:

CPEFPROF (X'00')
if CPEFDATA contains a profile

Note: For connections inbound from an SNA network CPEFSUBT can contain a profile flag (CPEFPROF).

CPEFPASS (X'01')
if CPEFDATA contains a password

CPEFUSER (X'02')
if CPEFDATA contains a user ID.

CPEFDATA
contains either a user ID or a password, depending on the value for CPEFSUBT. For VM, this user ID or password can be from one to eight bytes in length.

Note: For connections inbound from an SNA network, CPEFDATA can be from one to ten bytes in length. However, the 9th and 10th byte, if present, must be blank.

CPEFLUWL
is the length of the LUWID, not including this length byte. The actual LUWID is composed of the following parts:

CPEFFQNL
is the length of the fully qualified LU name, not including this length byte.

CPEFFQN
is the name of the network fully qualified LU name. This is a variable length.

CPEFLUIN
is the instance number of the LUWID.

CPEFLUSN
is the sequence number of the LUWID.

CPEFCORL
is the length of the sender's conversation correlator.

CPEFCORT
is the conversation correlator of the sender's transaction. This has a variable length.

CPED Part Three—VM-Defined Variable-Length Section: Immediately following the FHM5 is an optional part of the allocate area, the VM-defined variable-length section. The length of this section is defined in field CPEVVARL in the VM area, and is zero if this section does not exist.

The VM-defined variable-length section contains the supplementary GID area, a part of the POSIX access control information. The length of the supplementary GID area is also defined in the VM area (CPEVIDLN).

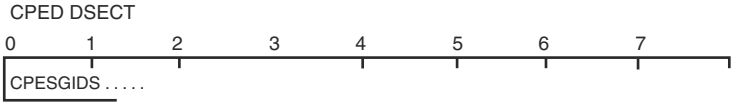


Figure 33. Connection Pending Extended Data, Part Three: VM-Defined Variable-Length Section

Examples of Allocate Data for a Connection to a VM Private Resource: The following three scenarios show examples of portions of allocate data for a connection to a private resource within a TSAF collection. Assume that the extension information is provided either directly by the program or through a CMS communication directory file.

1. If a source program, running in a virtual machine whose user ID is USER1, issues a connection with the following information specified in the parameter list and extension:
   - A resource ID (TPN) of PAYROLL
   - Mode name omitted
   - SYNCLVL=CONFIRM
   - MAPPED=NO
   - SECURITY(NONE).

   the VM area and FMH5 is in this format (assuming PAYROLL is managed by USER2):

```
                          VM architected area starts here
X'0040003800000000'       Lengths
X'0000000000000000'       Reserved
X'0000000000000000'       Mode name omitted
```

```
X'0000000000000000'     Reserved
                        Locally known LU name:
                        (from source program's point of view)
X'5CE4E2C5D9C9C440'     LU name qualifier, *USERID
X'E4E2C5D9F2404040'     Target LU name, USER2
                        Partner program's locally known LU name:
                        (from priv. res. manager's point of view)
X'5CE4E2C5D9C9C440'     Partner's LU name qualifier, *USERID
X'E4E2C5D9F1404040'     Partner's target LU name, USER1
                        FMH5 starts here
X'120502FF0003D00040'   First 9 bytes
X'08D7C1E8D9D6D3D340'   Transaction program named PAYROLL
```

2. If the source program specifies SECURITY(SAME) in the last example instead of SECURITY(NONE), the FMH5 is in this format:

```
                        FMH5 starts here
X'1A0502FF8003D00040'   Already verified flag is set
X'08D7C1E8D9D6D3D340'   Transaction program named PAYROLL
X'07'                   Length of security area
X'0602E4E2C5D9F1'       User ID, USER1
```

3. If the source program specifies SECURITY(PGM) with an access user ID of USER3 and an access password of PASS3 instead of SECURITY(NONE), the FMH5 is in the format below. Note that CP places only the access user ID into the FMH5 before the FMH5 is sent to the private resource manager program. (The target LU verifies the password, and in this case the target LU is the same as the source LU.)

```
                        FMH5 starts here
X'1A0502FF8003D00040'   Already verified flag is set
X'08D7C1E8D9D6D3D340'   Transaction program named PAYROLL
X'07'                   Length of security area
X'0602E4E2C5D9F3'       Access user ID, USER3
```

Considerations for Communication Servers

A communication server is an intermediate program (a *middleman*) that makes connections on behalf of another program. Communication servers must be authorized in the CP directory with the OPTION COMSRV statement.

When a communication server makes a connection on behalf of another user, it specifies FMH5=YES on its APPCVM CONNECT. This means that its connection parameter list extension contains allocate data. CP does not create this FMH5 and does not verify the format or contents of the supplied FMH5; CP simply passes it on to your partner.

CP can receive any valid FMH5 information from the SNA network and it may be different from the FMH5 generated from VM. For example, even though VM never generates the CPEFPROF flag in the CPEFSUBT security subfield, it may receive an FMH5 from the SNA network that does use this flag.

For more information on valid FMH5 information, see *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*.

The VM area being created by a communication server must be in the same format as part one of the CPED data (see Connection Pending Extended Data). However, a communication server creating a VM area must note special considerations for several VM area fields:

CPEVSRVL
when a communication server is making a connection with PIP data on behalf of another virtual machine, it must supply a value of 8 in this field. The value 8 is the length of the VM communication server area which points to the PIP variable (see note). Refer to Handling PIP Variables for more information.

Note: When a communication server specifies the length of the connection parameter list extension (BUFLEN=) and a PIP variable is present, the maximum length of the extension is 915 bytes. Refer to note "3" on page 415 for the BUFLEN= parameter when FMH5=YES is specified.

CPEVALT
is an alternate user ID that the target virtual machine uses. The user ID must be left-justified and padded on the right with blanks as necessary.

This alternate user ID is *assigned* to the target virtual machine when that machine accepts the connection, and it is unassigned when the connection is severed. Connection pending interrupts that contain an alternate user ID are not presented to a virtual machine that already has an alternate user ID assigned; such interrupts are deferred until the alternate user ID is unassigned. (However, these types of connection pending interrupts are presented if the virtual machine is a communication server.) No user ID is assigned if this field contains all zeros.

To specify this field, a virtual machine must be authorized in the CP directory for DIAGNOSE code X'D4'. Note that this defaults to class B privilege. However, using VM's user class modification, a virtual machine can be authorized to specify this alternate user ID without being given other class B privileges. For more information about modifying user classes, see *z/VM: CP Planning and Administration*.

CPEVIDFL
is a 1-byte field consisting of access control information. For valid values, see the description of "CPEVIDFL" on page 436.

CPEVEUID
contains one of the following:

- the effective UID of the process that issued the CONNECT
- the database UID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVEGID
contains one of the following:

- the effective GID of the process that issued the CONNECT
- the database GID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVRUID
contains one of the following:

- the real UID of the process that issued the CONNECT
- the database UID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVRGID
contains one of the following:

- the real GID of the process that issued the CONNECT
- the database GID value associated with the userid determined by the rules governing access security and based on the access security type specified.

Refer to Additional Considerations for POSIX Security Values for details.

CPEVIDLN
contains the length of the supplementary GID area, CPESGIDV.

CPESGIDS (in the VM-defined variable-length section)
contains the list of supplementary GIDs, a list of 4-byte GIDs immediately following each other.

Refer to Connection Pending Extended Data for a complete description of the FMH5, and also *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*.

Note: If you specify FMH5=YES, CMS communication directory resolution is disabled on any subsequent CMSIUCV CONNECT or CMSIUCV RESOLVE functions.

When a communication server is establishing a connection for another virtual machine, the communication server should use the ALTID keyword on APPCVM CONNECT. This parameter specifies the user ID of the virtual machine that made the original connection and is the virtual machine for which the connection is being made.

The communication server should use the user ID sent in the IPVMID field of the connection pending interrupt for accounting and problem determination. The value of the user ID sent in the IPVMID field of the connection pending interrupt depends on whether the connection is inbound from the SNA network or outbound to the SNA network.

If the connection is inbound from the SNA network to a TSAF collection and the SECURITY value is:

- NONE, then IPVMID in the connection pending interrupt is set to binary zeros.
- SAME, then IPVMID in the connection pending interrupt is set to the source user ID.
- PGM, then IPVMID is set to the verified access user ID from the security field of the FMH5.

If the connection is outbound from a TSAF or CS collection to the SNA network, the IPVMID field in the connection pending interrupt contains a source user ID. The source user ID is one of the following:

- The user ID specified with the ALTID parameter of APPCVM CONNECT
- An alternate user ID set by a DIAGNOSE code X'D4'
- The logon user ID of the virtual machine issuing the APPCVM CONNECT.

Note: The TSAF virtual machine gets an 8-byte node ID preceding the VM area, the FMH5, and the VM-defined variable-length section when it is handling a private resource connection.

Additional Considerations for POSIX Security Values: The VM area and the supplementary GID area together contain a set of fields known collectively as the POSIX access control information. These fields are:

- CPEVIDFL
- CPEVEUID
- CPEVEGID
- CPEVRUID
- CPEVRGID
- CPEVIDLN
- CPEVSGIDS

If the communications server knows the appropriate values to fill in for these fields, it should do so and ensure that bit zero of flag CPEVIDFL is one. If the communications server does not fill in these fields, it must ensure that bit zero of flag CPEVIDFL is zero. CP will then fill in the information.

How the VM systems are connected together and the level of each of the systems involved makes a difference in determining the source of the security data that is provided to the server.

- A server that resides on the same system as the requestor is provided with the POSIX security values for the currently-active process.
- A server accessible via ISFC (the system is a member of a CS collection) where all systems involved in the route from the requestor to the server are at least at the VM/ESA Version 2 Release 1.0 level is provided with the POSIX security values for the currently-active process. If there is a system with CP at a level prior to VM/ESA Version 2 Release 1.0 in the route, the POSIX security values are provided for the currently-active process, but the supplementary GIDs are stripped off.
- A server accessible via TSAF where all systems involved in the route from the requestor to the server are at least at the VM/ESA Version 2 Release 1.0 level and systems at prior levels have the appropriate APAR installed (see the Conversion Notebook for details) is provided with the POSIX security values for the currently-active process.
- A server accessible via AVS is provided with the database POSIX security values
- In each of the previous cases, if there is no currently-active process, the user's database POSIX security values are provided.

Examples of Allocate Data for an Outbound Connection: The following three scenarios show examples of portions of allocate data that the SNA communication server (AVS) gets for an outbound connection. Assume that the extension information is provided either by the source program or through a CMS communication directory file.

1. If a source program, running in a virtual machine whose user ID is USER1, issues a connection with the following information specified in the parameter list and extension:
   - A mode name of FAST
   - An LU name qualifier GATE1
   - A target LU name of ENDICOTT
   - A resource ID of PAYROLL
   - SYNCLVL=CONFIRM
   - MAPPED=NO
   - SECURITY(NONE).

   The SNA communication server gets a VM area, an FMH5, and a VM-defined variable-length section in this format:

```
                          VM architected area starts here
X'0040003800000000'       Lengths
X'0000000000000000'       Reserved
X'C6C1E2E340404040'       Mode name, FAST
X'0000000000000000'       Reserved
                          Locally known LU name:
                          (from the source program's point of view)
X'C7C1E3C5F0404040'       LU name qualifier, GATE1
X'C5D5C4C9C3D6E3E3'       Target LU name, ENDICOTT
                          Partner program's locally known LU name:
                          (from the communication server's point of view)
X'5CE4E2C5D9C9C440'       Partner's LU name qualifier, *USERID
X'E4E2C5D9F1404040'       Partner's target LU name, USER1
                          FMH5 starts here
X'120502FF0003D00040'     First 9 bytes
```

```
X'08D7C1E8D9D6D3D340'     Transaction program named PAYROLL
```

2. If the source program specifies SECURITY(SAME) in the last example instead of SECURITY(NONE) the communication server gets an FMH5 in this format:

```
                          FMH5 starts here
X'1A0502FF8003D00040'     Already verified flag is set
X'08D7C1E8D9D6D3D340'     Transaction program named PAYROLL
X'07'                     Length of security area
X'0602E4E2C5D9F1'         User ID, USER1
```

3. If the source program specifies SECURITY(PGM) with an access user ID of USER3 and an access password of PASS3 instead of SECURITY(NONE), the communication server gets an FMH5 in the format below. Note that in this case, CP places the access user ID **and** the access password in the FMH5 before the FMH5 is sent to the communication server. (The target LU verifies the password, and in this case the target LU is not the same as the source LU.)

```
                          FMH5 starts here
X'210502FF0003D00040'     First 9 bytes
X'08D7C1E8D9D6D3D340'     Transaction program named PAYROLL
X'0E'                     Length of security area
X'0602E4E2C5D9F3'         Access user ID, USER3
X'0601D7C1E2E2F3'         Access password, PASS3
```

Handling PIP Variables: Communication servers typically make connections on behalf of other users. They receive a connection pending interrupt, receive the allocate data, and pass on the allocate data using APPCVM CONNECT, FMH5=YES. The allocate data, consisting of a VM area and FMH5, is already contained in the parameter list extension.

Communication servers handle PIP variables a little differently. If the IPPIPLEN field in the connection pending interrupt is greater than 0, indicating PIP data is present, the communication server must do the following:

1. Receive the PIP variable using APPCVM RECEIVE PIP=YES.

   Note: If the communication server program is running in CMS, CMS issues the APPCVM RECEIVE PIP=YES. In CMS, when a user exit is driven for a connection pending interrupt, register 4 (R4) points to the PIP variable if it is nonzero. The communication server can thus access the PIP variable before accepting the conversation.
2. Set up the PIP variable using a single buffer or multiple buffers.
3. Create a **VM communication server area** that passes the PIP variable to the final target.
4. Specify a value of 8 in field CPEVSRVL of the VM area to denote the length of the VM communication server area.
5. Include the length of the VM communication server area (eight bytes) when specifying the BUFLEN parameter on APPCVM CONNECT.

The VM communication server area must follow the VM-defined variable-length section, and it has the format shown in the following figure.

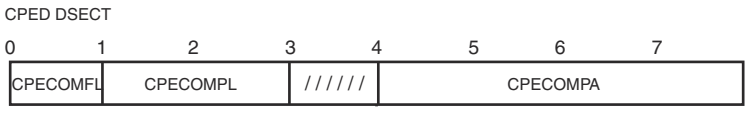


Figure 34. VM Communication Server Area

CPECOMFL
is a byte that contains the following flag:

CPECOMBL (X'40')
specifies that CPECOMPA contains the address of a list of multiple buffer addresses and lengths for the PIP variable.

CPECOMPL
is the **total** number of bytes (in binary) for the PIP variable. Valid values are 0, 4, or any value from 8 to 32,767. (CPECOMPL should be equal to the IPPIPLEN field specified in the connection pending interrupt.)

//////
is reserved and set to zeros.

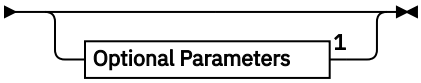
CPECOMPA
specifies either the address of a single data buffer, or the address of a list that contains multiple buffer addresses and lengths for the PIP variable.

If the VM communication server area is created, a communication server must add the area's length to the allocate data length (the BUFLEN parameter or IPBFLN2F field) when it issues the APPCVM CONNECT, FMH5=YES. If the communication server does not append the length, IPRCODE 50 results.

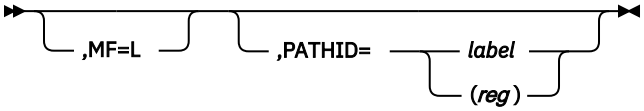
When the virtual machine that is the target of the APPCVM CONNECT receives the allocate data (using APPCVM RECEIVE), the VM communication server area is not presented. As a result, the CPEVSRVL field in the VM area contains binary zeros, and the allocate data length in the connection pending interrupt reflects only the lengths of the VM area, the FMH5, and the VM-defined variable-length section.

Considerations when using an External Security Manager with security label checking: An APPCVM CONNECT from a communication server will fail with a condition code of 1 (cc=1) and a return code of 15 (IPRCODE=X'0F') if security label checking is enabled on the system.

APPCVM QRYSTATE (Query State)



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

Purpose

The QRYSTATE function gets the current state of any APPC/VM path on the issuing virtual machine.

Note: This is issued only by the CMS Protected Conversation Adapter (PCA).

Parameters

Required Parameters:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM QRYSTATE.

PATHID=
specifies the path ID on which the state is to be queried.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

Input Parameter List: The APPCVM QRYSTATE parameter list has the following input format:

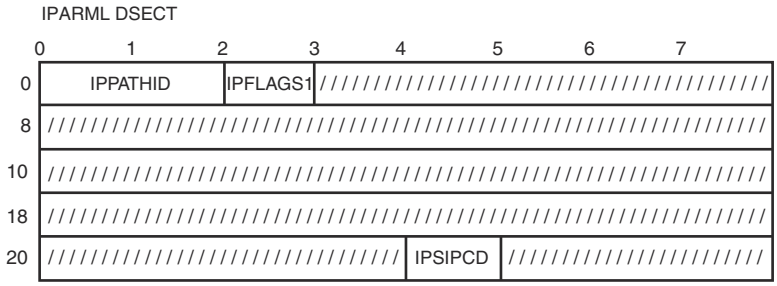


Figure 35. APPCVM QRYSTATE Input Parameter List

The parameters are:

IPPATHID
contains the path ID on which the state is to be queried.

IPFLAGS1
contains the following input bit flag:

IPAPPC (X'08')
This is an APPC function.

IPSIPCD
this value should always be set.

X'00'
this value is reserved for IBM use only.

X'01'
this value is reserved for IBM use only.

X'02'
this value is reserved for IBM use only.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
Not Possible	X	X	Not Possible

QRYSTATE always completes immediately.

CC=1
An error occurred before the QRYSTATE was initiated. The output parameter list is the same as the input shown in QRYSTATE Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Meaning
X'01'	1	The specified path ID is not yet established.
X'1E'	30	This is a non-APPC path.

CC=2
QRYSTATE completed (also see SENDREQ Completion). The output parameter list when CC=2 is:

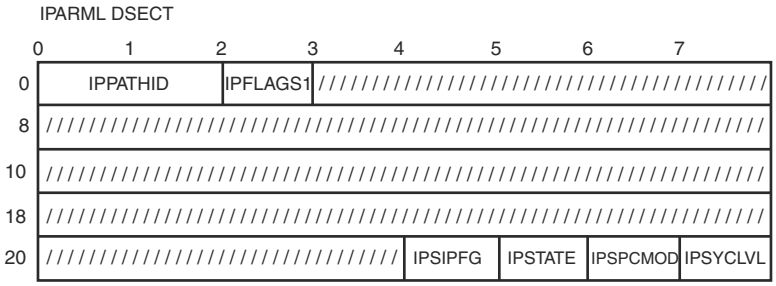


Figure 36. APPCVM QRYSTATE Output Parameter List

The parameters are:

IPPATHID
contains the path ID on which QRYSTATE completed.

IPFLAGS1
may contain the following output bit flag:

IPLRECL (X'10')
A logical record is in progress.

IPSIPFG
is reserved for IBM use only.

X'00'
this value is reserved for IBM use only.

X'01'
this value is reserved for IBM use only.

IPSTATE
contains the current state for this path, which may have one of the following values:

IPENDING (X'00')
A function is pending on the path. When the function completes, the state change will occur.

IPRESET (X'01')
The path is in the Reset state.

IPCONNCT (X'02')
The path is in the Connect state.

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPCONFRM (X'05')
The path is in the Confirm state.

IPSEVER (X'06')
The path is in the Sever state.

IPDEFRCV (X'07')
The path is in the Defer_Receive state.

IPDEFSVR (X'08')
The path is in the Defer_Sever state.

IPREPARE (X'09')
The path is in the Prepare_Received state.

X'0A'
this value is reserved for IBM use only.

IPURQCMT (X'0B')
The path is in the Unsolicited_Request_Commit_Received state.

IPBKOUT (X'0D')
The path is in the Backout_Received state.

IPBKREQ (X'0E')
The path is in the Backout_Required state.

IPSPCMOD
is reserved for IBM use only.

X'03'
this value is reserved for IBM use only.

X'04'
this value is reserved for IBM use only.

X'06'
this value is reserved for IBM use only.

IPSYCLVL
indicates the synchronization level for the path.

IPSYCNON (X'00')
SYNCLVL=NONE.

IPSYCCNF (X'01')
SYNCLVL=CONFIRM.

IPSYCSP (X'02')
SYNCLVL=SYNCPT.

Program Exceptions

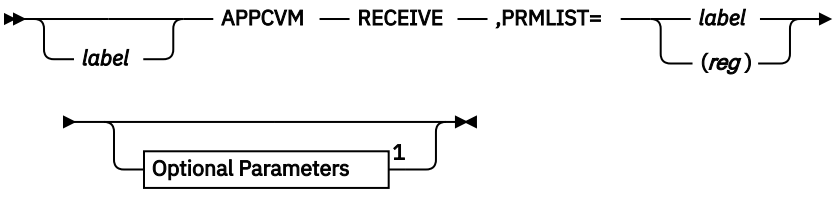
The program exceptions for QRYSTATE are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an application interrupt buffer has not been declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

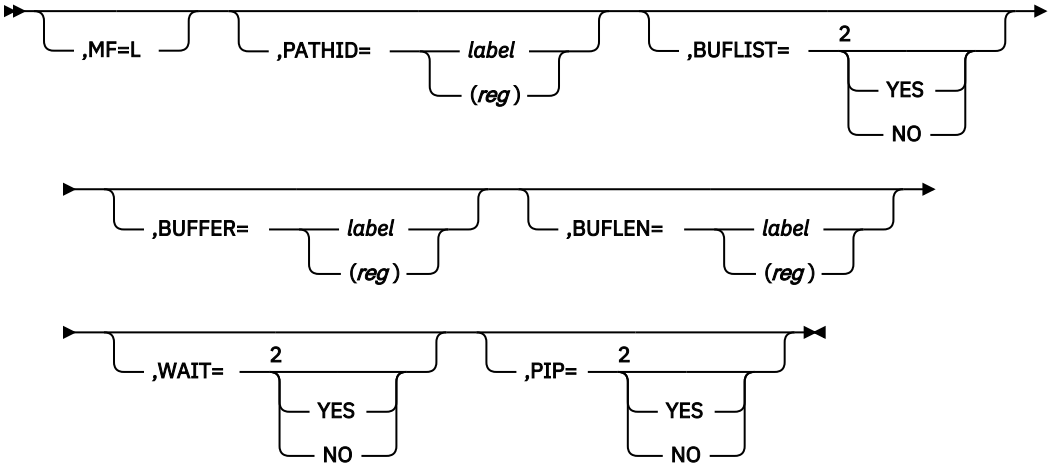
State Checks and State Changes

There are no state changes associated with the QRYSTATE function.

APPCVM RECEIVE



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

Use the RECEIVE function to receive data that is sent to your program.

In addition to receiving regular application data (sent by APPCVM SENDDATA), you can receive these special types of data:

- Log data, sent from your communication partner so that you can diagnose an error indication.
- Allocate data, sent from your communication partner to describe its connection request to you. You can receive this before accepting a connection.
- A program initialization parameter (PIP) variable. You can receive this before accepting a connection.

Refer to Message Pending External Interrupt for format and a description of the message pending interrupt.

When you issue RECEIVE and no message is currently pending on the path, the receive area that you specify is allocated for future messages on the path.

Parameters

Required Parameter:

PRMLIST=
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address is within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM RECEIVE function.

PATHID=
lets you identify the path on which to receive data.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

BUFLIST=
specifies the type of buffer address that the BUFFER parameter refers to. (See Specifying Buffers on RECEIVE.)

YES
refers to a list of addresses.

NO
refers to a single address.

BUFFER=
specifies the addresses of the areas into which CP places the data received. If BUFLIST=YES, this BUFFER address is the start of a list of addresses of discontiguous buffers. (See Specifying Buffers on RECEIVE.)

label
is the relocatable label in storage where CP places the data received.

(*reg*)
is the register number that contains the address in storage where CP places the data received. This storage area is where CP places the message.

BUFLEN=
specifies the total length of the area(s) into which APPC/VM places the data. (See Specifying Buffers on RECEIVE.)

label
is the relocatable label of the fullword that contains the length.

(*reg*)
is the register number that contains the length.

WAIT=
lets you specify when control is returned to your virtual machine.

YES
returns control to your virtual machine after the RECEIVE completes.

NO
returns control to your virtual machine when you initiate the RECEIVE. If the RECEIVE does not complete immediately, you are notified with a function complete interrupt when it does complete.

PIP=
lets you specify whether or not the APPCVM RECEIVE is for a PIP variable.

YES
specifies that this APPCVM RECEIVE is for the PIP variable. PIP=YES can only be specified from the Connect state.

You can determine the total length of the PIP variable from the IPPIPLEN field in the connection pending interrupt. If you receive only part of the PIP variable or have not yet received any portion of the PIP variable, and then either accept or sever the connection, any remaining portion of the PIP variable gets purged.

You can receive the PIP variable before any allocate data is received, after all the allocate data is received, or after only a portion of the allocate data is received.

NO
specifies that this APPCVM RECEIVE is not for the PIP variable.

Input Parameter List: The APPCVM RECEIVE parameter list has the input format shown in the following figure.

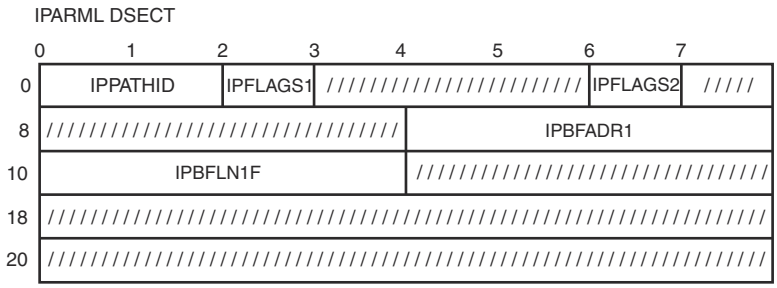


Figure 37. APPCVM RECEIVE Input Parameter List

IPPATHID
contains the path ID over which you receive data.

IPFLAGS1
contains one of the following input bit flags:

IPBUFLST (X'40')
a buffer list was specified.

IPAPPC (X'08')
an APPC function was issued.

IPFLAGS2
contains the following input bit flag:

IPWAIT (X'80')
a synchronous return is desired (WAIT=YES was specified). IPWAIT is ignored if IPPIP is on.

IPPIP (X'20')
this program is receiving the PIP variable (PIP=YES was specified).

IPBFADR1
contains the address of the area where APPC/VM stores the received data or the address of the list. See the following section, Specifying Buffers on RECEIVE.

IPBFLN1F
contains receive area length that IPBFADR1 specifies. See Specifying Buffers on RECEIVE.

Usage Notes

Specifying Buffers on RECEIVE: You can use the buffer support with APPCVM RECEIVE to receive normal application data, log data, allocate data, and PIP variables. You can receive data by specifying a single buffer using one address and one length, or by specifying multiple buffers using a list of addresses and lengths.

When you specify a single buffer using one address and one length:

- BUFFER specifies the address.
- BUFLEN specifies the length.
- BUFLIST must be equal to NO.

When you specify multiple buffers with a list of addresses and lengths:

- BUFFER specifies the address of the list.
- BUFLEN specifies the sum of the lengths of the buffers in the list.
- BUFLIST must be equal to YES.

When specifying address lists (BUFLIST=YES), note the following:

1. The list must begin on a doubleword boundary.
2. Each list entry must be two fullwords:
   - The first fullword is the address of that portion of the list.
   - The second fullword is the length of that portion of the list.
3. The addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete.
4. APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified by BUFLEN.

Note: Do not code a receiving program so that the length of the receive buffer is based on the length of your communication partner's send buffer.

When a program on the local system sends data, the length in the message pending interrupt is the actual length of the data sent. However, when a program on a remote system sends data, intermediate communication servers (like the TSAF virtual machine) could break up a single data stream into multiple data streams, or combine numerous data streams. As a result, the length shown in your message pending interrupt is the length of the data sent by the intermediate communication server, *not* the length of the data sent by the source program. For example, your communication partner could send a data stream of 0 length followed by a data stream of 100 bytes. However, your program, the target, might get only one data stream message with a length of 100 bytes.

When participating in a basic conversation, your program should examine the 2-byte logical record length (LL) field to determine how much data your communication partner has sent on a logical record. See Setting Up the Data To Send for more information on logical records.

Condition Codes and Return Codes

CC=0

The RECEIVE started successfully, but has not yet completed. If your virtual machine is enabled for function complete interrupts, one is presented to your virtual machine when RECEIVE completes. The interrupt format is the same as the RECEIVE output parameter list (see CC=2 or CC=3). When you get the function complete interrupt, check the IPAUDIT field for error information.

When control is returned to your virtual machine with CC=0, the parameter list may have been altered.

Note: CC=0 is not possible when WAIT=YES or PIP=YES.

CC=1

An error occurred before the RECEIVE was initiated. The parameter list format is the same as the input shown in RECEIVE Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.
X'03'	3	A function is pending on this path.
X'06'	6	A storage protection exception occurred on one of your partner's buffers that was used for one of the following: • Application data (from APPCVM SENDDATA) • The PIP variable (from APPCVM CONNECT) • Log data (from APPCVM SENDERR or APPCVM SEVER) This applies only when your RECEIVE was issued after a message was pending on the path.
X'07'	7	An addressing exception occurred on one of your partner's buffers that was used for one of the following: • Application data (from APPCVM SENDDATA) • The PIP variable (from APPCVM CONNECT) • Log data (from APPCVM SENDERR or APPCVM SEVER) This applies only when your RECEIVE was issued after a message was pending on the path.
X'0A'	10	Your receive buffer length is negative.
X'14'	20	The PIP variable cannot be received because the originator has severed the path.
X'16'	22	Your communication partner's send buffer list or PIP variable buffer list is invalid.
X'17'	23	A length specified in your receive buffer list is negative.
X'18'	24	The total length specified (BUFLEN) is not the total of the receive buffer lengths in your list.
X'1A'	26	The receive buffer list address is not on a doubleword boundary.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	RECEIVE is an invalid function from the Connect state. (See RECEIVE State Checks and State Changes.)
X'21'	33	RECEIVE PIP=YES is an invalid function from the Send state.
X'22'	34	RECEIVE PIP=YES is an invalid function from the Receive state.
X'23'	35	RECEIVE is an invalid function from the Confirm state.
X'24'	36	RECEIVE is an invalid function from the Sever state when the receive is not for log data.
X'2B'	43	There is an invalid logical record length in your communication partner's data stream.
X'2C'	44	Before issuing RECEIVE, you started—but did not finish—sending a logical record.
X'2D'	45	Your communication partner started—but did not finish—sending a logical record and tried to change to the Receive state.
X'38'	56	WAIT=YES was specified on a function issued to this same virtual machine.

Hex Code	Decimal Code	Why the Error Occurred
X'3B'	59	Your partner specified an invalid GDS ID within the PIP variable.
X'3C'	60	Your partner specified an invalid length for a PIP subfield, or the total length in bytes 0 and 1 of the PIP variable is not the sum of the lengths of the PIP subfields.
X'44'	68	RECEIVE is invalid from the Reset state.
X'45'	69	RECEIVE is invalid from the Defer_Receive state.
X'46'	70	RECEIVE is invalid from the Defer_Sever state.
X'47'	71	RECEIVE is invalid from the Prepare_Received state.
X'49'	73	RECEIVE is invalid from the Unsolicited_Request_Commit_Received state.
X'4B'	75	RECEIVE is invalid from the Backout_Received state.
X'4C'	76	RECEIVE is invalid from the Backout_Required state.
X'5B'	91	A paging or storage error was detected in your communication partner's SEND data area
X'5C'	92	A paging or storage error was detected in the RECEIVE data area

These return codes can only occur if a message is pending for the specified path at the time you issued the RECEIVE: X'06', X'07', X'16', X'17', X'18', X'2B', and X'2D'.

If no message is pending for the specified path at the time you issued the RECEIVE, CP reports the above error conditions (X'06', X'07', X'16', X'17', X'18', X'2B', and X'2D') to you in the corresponding IPAUDIT flags of your RECEIVE output parameter list, when your RECEIVE completes.

Notes:

1. Whenever CC=1, fields in the parameter list may have been altered.
2. For return codes X'16', X'17', X'18', X'2B', X'2D', X'3B', and X'3C' some data may have been received, but the amount is unpredictable.

CC=2 or CC=3

The RECEIVE is complete. (Also see RECEIVE Completion.) When CC=2, the function completed with no errors; when CC=3, there is error information in IPAUDIT.

Note: CC=3 is not possible when WAIT=NO or PIP=YES.

The output parameter list when CC=2 or CC=3 is shown in the following figure.

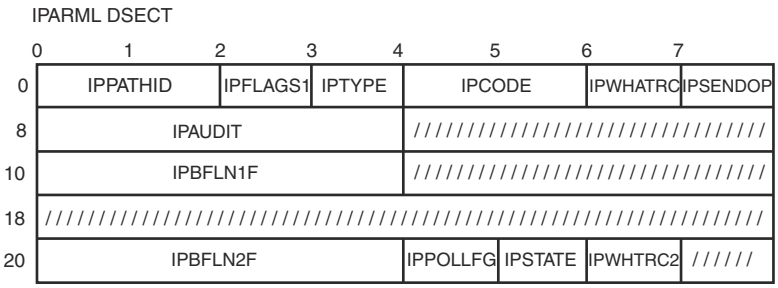


Figure 38. APPCVM RECEIVE Output Parameter List (Function Complete Interrupt)

IPPATHID
contains the path ID on which the function is complete.

IPFLAGS1
may contain one or more of the following bit flags:

IPFLUSH (X'40')
for a non-sync-point conversation, your partner issued SENDDATA FLUSH=YES. In this case, this flag can only be seen by a communication server.

For a sync-point conversation, this flag is set under two conditions:

- Your partner issued SENDDATA FLUSH=YES BUFLEN=0 from the Send state. In this case, you must be a communication server to see this flag.
- Your partner issued a SENDDATA FLUSH=YES from the Defer_Receive state. In this case, you see this flag even if you are not a communication server.

X'20'
This value is reserved for IBM use only.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
contains the error, sever, or sense code from the partner's SENDERR, SEVER, or attempt to initiate a backout sync-point. IPCODE is only valid when IPWHATRC=IPERROR, IPSABEND, or X'14'. See “APPC/VM Sever, Error, and Sense Codes That You Can Get” on page 399.

IPWHATRC
contains the what-received code:

IPDATA (X'01')
only data was received, with no other indications.

IPSEND (X'02')
Your partner has switched the conversation around (using RECEIVE, SENDDATA RECEIVE=YES, or SENDDATA FLUSH=YES from the Defer_Receive state) and you are now in the Send state.

IPERROR (X'03')
Your partner issued SENDERR.

IPCNFRM (X'04')
Your partner issued SENDCNF TYPE=NORMAL.

IPCNFSEV (X'05')
Your partner issued SENDCNF TYPE=SEVER.

IPSNORM (X'08')
Your partner issued a SEVER TYPE=NORMAL.

IPSABEND (X'09')
Your partner issued a SEVER TYPE=ABEND.

IPALLOCD (X'0B')
The allocate data was received.

IPSNDCNF (X'0C')
Your partner issued SENDCNF TYPE=PREPRECV.

IPLGDATA (X'0D')
The log data was received.

IPPIPDAT (X'0E')
The PIP variable has been received.

IPPREPAR (X'0F')
The function was completed with an indication that your partner initiated a CRR commit sync-point. You should commit the CMS work unit of which this conversation is a part.

IPREQCOM (X'10')
The function was completed with an indication that your partner initiated a CRR commit sync-point. You should commit the CMS work unit of which this conversation is a part.

IPBACK (X'14')
The function was completed with an indication that your partner initiated a CRR backout sync-point. You should back out the CMS work unit of which this conversation is a part.

X'16'
This value is reserved for IBM use only.

Notes:

1. Data may have been received for any IPWHATRC value.
2. You get IPDATA in IPWHATRC until you receive all the data that your partner (or intermediate communication server) sent with or before a nondata function. For example, IPWHATRC would be IPDATA if:
   a. Your partner (or intermediate communication server) issued SENDDATA RECEIVE=YES with a data length of 200 bytes, and
   b. You did a RECEIVE for 199 bytes.

   When you issue a RECEIVE for the 200th byte, then IPWHATRC would become IPSEND.

IPSENDOP
contains the SEND option code:

IPRECV (X'0A')
The function being completed is a RECEIVE.

IPAUDIT
has four fields that may contain error information.

Note: In the following descriptions,

- **Receive area** refers to either a receive buffer specified directly on an APPCVM RECEIVE, BUFFER= or a receive buffer that is part of a buffer list.
- **Send area** refers to a buffer specified for the PIP variable (on APPCVM CONNECT), a buffer specified directly on an APPCVM SENDDATA, BUFFER=, or a buffer that is part of a buffer list.

IPAUDIT1 (first byte of IPAUDIT)
may contain one of the following bit flags:

IPADANPX (X'10')
A protection exception occurred on your receive area. This only applies if your RECEIVE was issued before a message was pending on the path. (It does not apply when receiving log data.)

IPADANAX (X'08')
An addressing exception occurred on your receive area. This only applies if your RECEIVE was issued before a message was pending on the path. (It does not apply when receiving log data.)

IPAUDIT2 (second byte of IPAUDIT)
May contain one of the following bit flags:

IPADRPPX (X'20')
A protection exception occurred on your communication partner's send area. This only applies if your RECEIVE was issued before a message was pending on the path. (It does not apply when log data is being sent.)

IPADRPAX (X'10')
An addressing exception occurred on your communication partner's send area. This only applies if your RECEIVE was issued before a message was pending on the path. (It does not apply when log data is being sent.)

IPADRLST (X'04')
Your communication partner had an invalid SEND list.

IPAUDIT3 (third byte of IPAUDIT)
May contain one of the following bit flags:

IPADALEN (X'40')
A bad length is in your RECEIVE buffer list.

IPADATOT (X'10')
Your RECEIVE buffer length is invalid.

IPADTINV (X'08')
Your communication partner's data stream has an invalid logical record length.

IPADTTRN (X'02')
Your communication partner started, but did not finish, sending a logical record and tried to change to the Receive state.

IPASYRC (fourth byte of IPAUDIT)
May contain one of the following error codes (for which and appropriate IPRCODE was given to your communication partner):

Hex Code	Decimal Code	Meaning
X'5B'	91	A paging or storage error was detected in your communication partner's SEND data area
X'5C'	92	A paging or storage error was detected in the RECEIVE data area.

Note: IPAUDIT X'5D' and X'5E' are possible on a RECEIVE when the partner issues a RECEIVE prior to the SENDDATA (RECEIVE ahead).

IPBFLN1F
contains the length of pending log data for you to receive. This field is only meaningful when IPWHATRC is equal to IPSABEND or IPERROR.

IPBFLN2F
contains one of the following depending on the value of IPWHATRC:

- If IPWHATRC is equal to IPDATA, IPBFLN2F contains the number of bytes that were sent but did not fit into your defined RECEIVE area. This length is the byte length of your communication partner's (or communication server's) application data, allocation data, log data, or PIP variable, minus the length that you already received.

  For example, your communication partner or communication server issues SENDDATA with a data length of 100, and you issue RECEIVE with a buffer length of 40, then IPBFLN2F would contain 60.
- If IPWHATRC is not equal to IPDATA, then IPBFLN2F contains the number of unused bytes remaining in your RECEIVE area.

  For example, your communication partner or communication server issues SENDDATA RECEIVE=YES with a data length of 150, and you issue RECEIVE with a buffer length of 200, then IPWHATRC would contain IPSEND and IPBFLN2F would contain 50.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
Contains the current state for this path, which may be any of the following values:

IPCONNCT (X'02')
The path is in the Connect state.

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPCONFRM (X'05')
The path is in the Confirm state.

IPSEVER (X'06')
The path is in the Sever state.

IPREPARE (X'09')
The path is in the Prepare_Received state.

IPURQCMT (X'0B')
The path is in the Unsolicited_Request_Commit_Received state.

IPBKOUT (X'0D')
The path is in the Backout_Received state.

IPBKREQ (X'0E')
The path is in the Backout_Required state.

IPWHTRC2
further qualifies the IPWHATRC=IPPREPAR and IPWHATRC=IPREQCOM.

IPTPSEND (X'03')
Your partner requests to be in the Send state upon the successful completion of the commit sync-point.

IPTPRECV (X'04')
Your partner requests to be in the Receive state upon the successful completion of the commit sync-point.

IPTPSEVR (X'06')
Your partner requests to be in the Sever state upon the successful completion of the commit sync-point.

Program Exceptions

The program exceptions for RECEIVE are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine. An addressing exception also occurs for any of the following: • An invalid buffer address in the RECEIVE parameter list • An invalid buffer list address in the RECEIVE parameter list • An invalid buffer address in the RECEIVE buffer list. This only applies if the RECEIVE is issued after a message is pending on the path. (This includes messages for application data, the PIP variable, and log data.)
Operation	Either an external interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.

Type	Description
Protection	The storage key of the parameter list address does not match the key of the user. A protection exception also occurs for any of the following: • The buffer address in the parameter list is protected. • The buffer list address in the parameter list is protected. • A buffer address in the buffer list is protected. This only applies if the RECEIVE is issued after a message is pending on the path. (This includes messages for application data, the PIP variable, and log data.)
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check results when APPCVM RECEIVE is issued from an improper state (you receive an IPRCODE when you issue a function from the improper state). You get a state check (IPRCODE=X'20') if you issue an APPCVM RECEIVE from the Connect state under the following conditions:

- After you have received all the allocate data
- After you have received all the PIP variable (when PIP=YES)
- When no PIP variable was sent (when PIP=YES)
- You issue an APPCVM RECEIVE PIP=YES after a previous APPCVM RECEIVE PIP=YES was invalid.

You get a state check (IPRCODE=X'2B' or X'2C') if you issue APPCVM RECEIVE PIP=YES from the Receive or the Send state. You also get a state check (IPRCODE=X'2C') if you start, but do not finish, sending a logical record on this path. See the list of APPCVM RECEIVE return codes for all state check conditions.

No state change occurs when CC=1. State changes can occur when either of the following happens:

- The function completes, and control is returned to the virtual machine (CC=2 or 3).
- The function complete interrupt is accepted by your virtual machine (or you use IUCV TESTCMPL to discover that the function completed).

The state change depends on the IPWHATRC value you got in the output parameter list:

IPWHATRC Value	New State	Cause
IPDATA	Receive or Connect or Sever	Depends on your state when you invoke RECEIVE: • If you are **not** in the Connect state or the Sever state, you enter the Receive state. The RECEIVE completed without you receiving any nondata indications. Your communication partner sent data to complete the RECEIVE, or the receive length was 0. • If you are in the Connect state, you remain in the Connect state. An IPDATA indication means there is still some allocation data or a portion of PIP variable left for you to receive. • If you are in the Sever state, you remain in the Sever state. An IPDATA indication means that there is still some log data left for you to receive.

IPWHATRC Value	New State	Cause
IPSEND	Send	Your communication partner issued RECEIVE or SENDDATA RECEIVE=YES to complete the RECEIVE. Otherwise, your communication partner issued SENDDATA FLUSH=YES, BUFLEN=0 from the Defer_Receive state.
IPERROR	Receive	Your communication partner issued a SENDERR to complete the RECEIVE.
IPCNFRM	Confirm	Your communication partner issued a SENDCNF TYPE=NORMAL to complete the RECEIVE.
IPSNDCNF	Confirm	The partner is requesting confirmation that it can enter the Receive state. Your communication partner issued a SENDCNF TYPE=PREPRECV from the Send state to complete the RECEIVE. Your communication partner issued SENDCNF TYPE=PREPRECV from the Defer_Receive state.
IPCNFSEV	Confirm	Your communication partner issued a SENDCNF TYPE=SEVER to complete the RECEIVE.
IPSNORM	Sever	Your communication partner issued a SEVER TYPE=NORMAL to complete the RECEIVE.
IPSABEND	Sever	Your communication partner issued a SEVER TYPE=ABEND to complete the RECEIVE.
IPALLOCD	Connect	You completed the RECEIVE of the allocate data.
IPPIPDAT	Connect	You completed the RECEIVE of the PIP variable.
IPLGDATA	Receive or Sever	You completed the RECEIVE of the log data. If you were in the Receive state, you remain in the Receive state; if you were in the Sever state, you remain in the Sever state.
IPPREPAR	Prepare_Received	Your communications partner initiated a commit sync-point to complete the RECEIVE.
IPREQCOM	Unsolicited_Request_ Commit_Received	Your communications partner initiated a commit sync-point to complete the RECEIVE.
IPBACK	Backout_Received	Your communications partner initiated a backout sync-point to complete the RECEIVE.
X'16'		This state is reserved for IBM use only.

Completion Conditions

You cannot issue another RECEIVE, SEND, or SEVER TYPE=NORMAL on the same path until your outstanding RECEIVE completes. (In this case, SEND refers to the set of APPC/VM send functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.) Your RECEIVE completes immediately when it is issued for the following:

- A pending message that has no data
- A pending message that has enough data to fill your receive area.

If the amount of pending data is not enough to fill your receive area, your program's RECEIVE waits until either it gets more data or gets a nondata indication from your partner.

Your RECEIVE also completes when your receive area is larger than the data in the pending message, but you get an indication (in IPWHATRC) that:

- Your communication partner issued RECEIVE, SENDCNF, SENDDATA RECEIVE=YES, SENDERR, or SEVER.
- You have received all allocate data, PIP variables, or log data.

When your receive area has a 0 length and you are in:

- The Receive state, the RECEIVE completes immediately
- The Send state, the RECEIVE completes when your communication partner or an intermediate communication server receives notice that it is in the Send state.

Note the following:

- When the RECEIVE is for allocate data, you get IPDATA in IPWHATRC if you only partially receive allocate data. When you receive all the allocate data, you get IPALLOCD in IPWHATRC.
- When the RECEIVE is for a PIP variable, you get IPDATA in IPWHATRC if you only partially receive the PIP variable. When you receive the entire PIP variable, you get IPPIPDAT in IPWHATRC.
- When the RECEIVE is for log data, you get IPDATA in IPWHATRC if you only partially receive log data. When you receive all the log data, you get IPLGDATA in IPWHATRC.
- Your RECEIVE always completes immediately if your receive area is larger than pending allocate data, PIP variable, or log data.
- Except when receiving allocation data or PIP data, if your communication partner severs, your RECEIVE completes immediately. The IPWHATRC value in your output parameter list indicates the sever, and you are presented with a sever interrupt.

What Happens to Your VM Communication Partner

If an intermediate communication server (like TSAF, ISFC, or AVS) handles communication between you and your partner, the information in this section also describes what happens to the intermediate server when you issue RECEIVE.

After you issue APPCVM RECEIVE, your communication partner may get a function complete interrupt, a message pending interrupt, or no indication at all.

Your partner gets a function complete interrupt for the following conditions, your partner:

- Has a SENDDATA RECEIVE=NO or SENDDATA RECEIVE=YES with a 0 answer length outstanding on its half of the path, and you have received all the data sent.
- Issued SENDERR without sending log data.
- Issued SENDERR with log data, and you have received all the log data.
- Issued APPCVM SEVER TYPE=ABEND with log data, and you have received all the log data.

If your communication partner has a SENDCNF or SENDDATA RECEIVE=YES outstanding on its half of the path, your partner does not get any notification of your actions on that path until you respond.

Your communication partner does not get an indication that you have received allocate data or PIP variable.

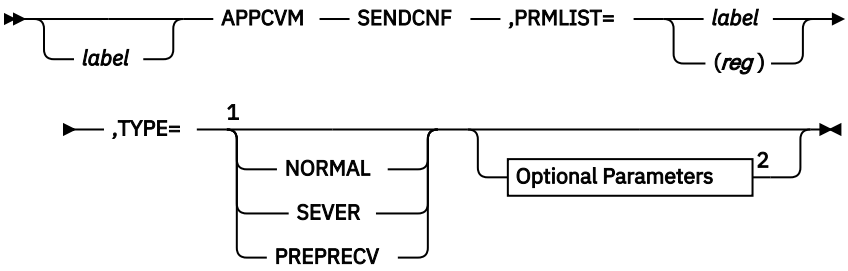
Note: Your communication partner should not reuse the data buffers for the PIP variable until the connection request has completed.

Your partner gets a message pending interrupt if your partner:

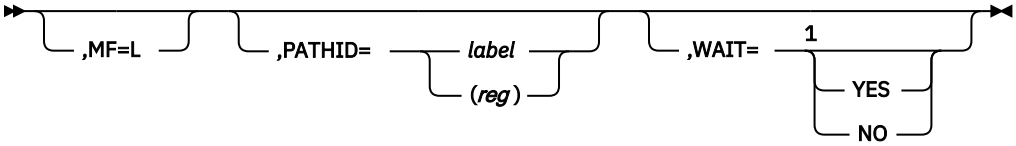
- Has no function outstanding on its half of the path
- Is in the Receive state on its half of the path
- Is enabled for message pending interrupts.

See Message Pending External Interrupt.

APPCVM SENDCNF (Send Confirm)



Optional Parameters



Notes:

[1] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

[2] Optional parameters can be entered in any order.

Purpose

Use the SENDCNF (Send Confirm) function to send a confirmation request from your program to your communication partner. Requesting confirmation helps to establish synchronization between two programs.

For either program in a conversation to use SENDCNF, the program establishing a conversation must specify SYNCLVL=CONFIRM or SYNCLVL=SYNCPT on the APPCVM CONNECT.

Your partner program can reply to your SENDCNF by sending a positive response using SENDCNFD (Send Confirmed), sending a negative response using SENDERR, or issuing SEVER.

Parameters

Required Parameters:

PRMLIST=
Specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

TYPE=
specifies what type of confirmation is being requested.

NORMAL
requests a normal confirmation.

SEVER
requests a confirmation that would let you issue a SEVER.

Note: For a SYNCLVL=SYNCPT conversation, this parameter can only be specified by an authorized communication server.

PREPRECV
also requests a normal confirmation, but if this confirmation is successful, your program switches to the Receive state.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SENDCNF.

PATHID=
lets you identify the path on which to send the confirmation request.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

WAIT=
specifies when control is returned to your virtual machine.

YES
returns control to your virtual machine when the SENDCNF is complete.

NO
returns control to your virtual machine as soon as the SENDCNF request is initiated. When the SENDCNF completes, you are notified with a function complete interrupt.

Input Parameter List: The APPCVM SENDCNF parameter list has the input format shown in the following figure.

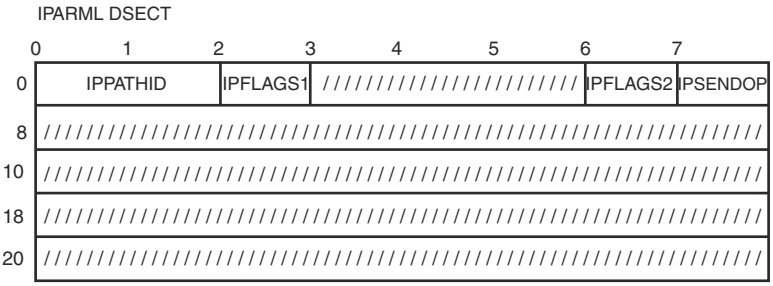


Figure 39. APPCVM SENDCNF Input Parameter List

IPPATHID
contains the path ID on which the confirmation request is sent.

IPFLAGS1
contains the following input bit flag:

IPAPPCSN (X'02')
indicates an APPC SEND function was issued.

IPFLAGS2
may contain one or more of the following input bit flags:

IPWAIT (X'80')
a synchronous return was requested.

IPCOMSRV (X'20')
SENDCNF was issued by a communication server.

Note: This bit must be set on by an authorized communication server if SENDCNF TYPE=SEVER is issued on a SYNCLVL=SYNCPT conversation.

IPSENDOP
contains one of the following SEND option codes:

IPCNFRM (X'04')
You are requesting normal confirmation from your communication partner.

IPCNFSEV (X'05')
You are requesting confirmation from your communication partner that you can issue a SEVER.

IPPREPRC (X'0C')
You are requesting confirmation from your communication partner that you can enter the Receive state.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	Not Possible

CC=0
SENDCNF started successfully, but has not yet completed. When it completes, CP presents your virtual machine with a function complete interrupt. The function complete interrupt buffer has the same format as the SENDCNF output parameter list (see CC=2, below).

Note: When WAIT=YES, CC=0 is not possible.

CC=1
An error occurred. The output parameter list is the same as the input shown in SENDCNF Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.
X'03'	3	A function is pending on this path.
X'1D'	29	You are not authorized as a communication server.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	SENDCNF is an invalid function from the Connect state.
X'22'	34	SENDCNF is an invalid function from the Receive state.
X'23'	35	SENDCNF is an invalid function from the Confirm state.
X'24'	36	SENDCNF is an invalid function from the Sever state.
X'25'	37	The connection was established with SYNCLVL=NONE.
X'26'	38	The IPSENDOP field contains an invalid value.
X'2C'	44	Before invoking SENDCNF, you started, but did not finish, sending a logical record.
X'38'	56	WAIT=YES was specified on a function issued to this same virtual machine.
X'44'	68	SENDCNF is invalid from the Reset state.
X'45'	69	SENDCNF TYPE=NORMAL or TYPE=SEVER is invalid from the Defer_Receive state.

Hex Code	Decimal Code	Why the Error Occurred
X'46'	70	SENDCNF TYPE=NORMAL/SEVER is invalid from the Defer_Sever state.
X'47'	71	SENDCNF is invalid from the Prepare_Received state.
X'49'	73	SENDCNF is invalid from the Unsolicited_Request_Commit_Received state.
X'4B'	75	SENDCNF is invalid from the Backout_Received state.
X'4C'	76	SENDCNF is invalid from the Backout_Required state.
X'4F'	79	SENDCNF TYPE=SEVER is invalid on a SYNCLVL=SYNCPT conversation when the IPCOMSRV bit is not set on.

CC=2

SENDCNF completed (see SENDCNF Completion), with no errors.

The output parameter list when CC=2 is shown in the following figure.

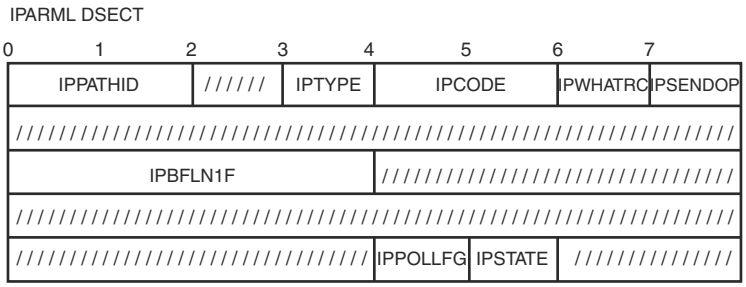


Figure 40. APPCVM SENDCNF Output Parameter List (Function Complete Interrupt)

IPPATHID
contains the path ID on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
contains the error code from the partner's SENDERR or sever code from the partner's SEVER. IPCODE is only valid when IPWHATRC=IPERROR or IPSABEND. See “APPC/VM Sever, Error, and Sense Codes That You Can Get” on page 399.

IPWHATRC
contains one of the following what-received codes:

IPCOMP (X'00')
Your partner's SENDCNFD completed the function.

IPERROR (X'03')
Your partner issued SENDERR.

IPSABEND (X'09')
Your partner issued a SEVER TYPE=ABEND.

IPBACK (X'14')
Your partner initiated a backout sync-point.

IPSENDOP
contains one of the following SEND option codes:

IPCNFRM (X'04')
The SENDCNF TYPE=NORMAL is being completed.

IPCNFSEV (X'05')
The SENDCNF TYPE=SEVER is being completed.

IPPREPRC (X'0C')
The SENDCNF TYPE=PREPRECV is being completed.

IPBFLN1F
contains the length of pending log data for you to receive. This field is meaningful only when IPWHATRC is equal to IPSABEND or IPERROR.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
contains one of the following values that show the current state for this path:

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPSEVER (X'06')
The path is in the Sever state.

IPBKOUT (X'0D')
The path is in the Backout_Received state.

IPBKREQ (X'0E')
The path is in the Backout_Required state.

Program Exceptions

The program exceptions for SENDCNF are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an external interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check results when your virtual machine issues APPCVM SENDCNF and it is not in the Send state on this path. A state check also occurs (IPRCODE=X'2C') if you started—but did not finish—sending a logical record on this path. (See the list of APPCVM SENDCNF return codes for all state check conditions.)

No state change occurs when CC=1. State changes can occur when the function completes, and one of the following occurs:

- You regain control (CC=2).
- You accept the function complete interrupt (CC=0), or you use TESTCMPL to discover that the function was completed.

The state change depends on the IPWHATRC value:

IPWHATRC Value	New State	Cause
IPCOMP	Send	The SENDCNF TYPE=NORMAL was completed by the communication partner issuing a SENDCNFD.
IPCOMP	Sever	The SENDCNF TYPE=SEVER was completed by the communication partner issuing a SENDCNFD.
IPCOMP	Receive	The SENDCNF TYPE=PREPRECV, issued from Send or Defer_Receive state, was completed by the communication partner issuing a SENDCNFD.
IPERROR	Receive	The SENDCNF was completed by the communication partner issuing a SENDERR.
IPSABEND	Sever	The SENDCNF was completed by the communication partner issuing a SEVER TYPE=ABEND.
IPBACK	Backout_Received	Your communication partner initiated a backout sync-point.

Note: When you issue SENDCNF, you have no way of telling if the SENDERR or SEVER indication received is in response to your confirmation request or if it was issued before your SENDCNF.

Completion Conditions

After issuing a SENDCNF, you cannot issue another SEND, RECEIVE, or SEVER TYPE=NORMAL on that path until the outstanding SENDCNF is complete. (SEND generally refers to the APPC/VM send functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.) SENDCNF is complete when the communication partner responds with a SENDCNFD, SENDERR, or SEVER.

What Happens to Your VM Communication Partner

Your communication partner's outstanding function may complete, or your partner may get a message pending interrupt.

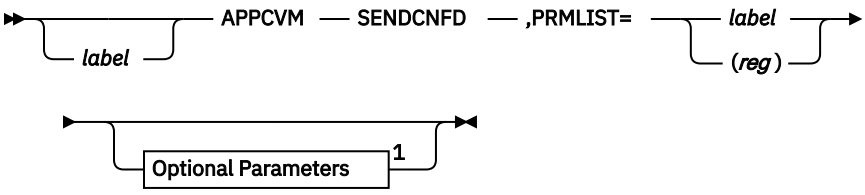
If your partner has a RECEIVE, SENDDATA RECEIVE=YES, or SENDERR outstanding on its half of the path, your partner's outstanding function is completed.

Your partner gets a message pending interrupt, if your partner:

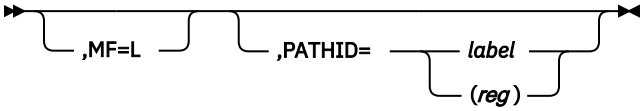
- Has no function outstanding on its half of the path
- Is in the Receive state on its half of the path
- Is enabled for message pending interrupts.

See Message Pending External Interrupt.

APPCVM SENDCNFD (Send Confirmed)



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the SENDCNFD (Send Confirmed) function to send a confirmation response from your program to another program. You should issue this as a positive response to your partner sending a SENDCNF (or to confirm your partner's backout sync-point). (For a negative response, invoke SENDERR; see "APPCVM SENDERR (Send Error)" on page 490.)

Parameters

Required Parameter:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SENDCNFD.

PATHID=
identifies the path on which to send the confirmation.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

Input Parameter List: The APPCVM SENDCNFD parameter list has the input format shown in the following figure.

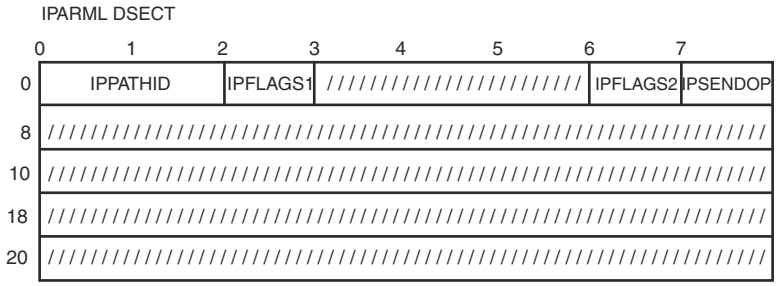


Figure 41. APPCVM SENDCNFD Input Parameter List

IPPATHID
contains the path ID on which the confirmation is sent.

IPFLAGS1
contains the following input bit flag:

IPAPPCSN (X'02')
The APPC SEND function was issued.

IPFLAGS2
Contains the following input bit flag:

IPCOMSRV (X'20')
The SENDCNFD was issued from a communication server.

Note: The flag must be set by an authorized communication server. Virtual machines that are not communication servers, should not set this flag.

IPSENDOP
contains the SEND option code:

IPCNFRMD (X'06')
your communication partner is sending confirmation as requested.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
Not Possible	X	X	Not Possible

SENDCNFD always completes immediately.

CC=1
An error occurred. The parameter list format is the same as the input shown in the SENDCNFD Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	SENDCNFD is an invalid function from the Connect state.
X'21'	33	SENDCNFD is an invalid function from the Send state.
X'22'	34	SENDCNFD is an invalid function from the Receive state.
X'24'	36	SENDCNFD is an invalid function from the Sever state.
X'26'	38	The IPSENDOP field contains an invalid value.

Hex Code	Decimal Code	Why the Error Occurred
X'44'	68	SENDCNFD is invalid from the Reset state.
X'45'	69	SENDCNFD is invalid from the Defer_Receive state.
X'46'	70	SENDCNFD is invalid from the Defer_Sever state.
X'47'	71	SENDCNFD is invalid from the Prepare_Received state.
X'49'	73	SENDCNFD is invalid from the Unsolicited_Request_Commit_Received state.
X'4C'	76	SENDCNFD is invalid from the Backout_Required state.

CC=2

SENDCNFD completed (see SENDCNFD Completion).

The output parameter list when CC=2 is shown in the following figure.

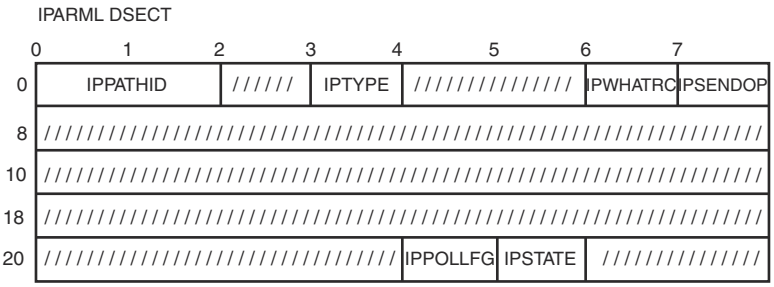


Figure 42. APPCVM SENDCNFD Output Parameter List (Function Complete Interrupt)

IPPATHID
contains the path ID on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPWHATRC
contains one of the following what-received codes:

IPCOMP (X'00')
SENDCNFD completed in response to a SENDCNF TYPE=NORMAL.

IPSNORM (X'08')
SENDCNFD completed in response to a SENDCNF TYPE=SEVER.

IPSNDCNF (X'0C')
SENDCNFD completed in response to a SENDCNF TYPE=PREPRECV.

IPCNFBK (X'15')
SENDCNFD completed in response to your partner's backout sync-point.

IPSENDOP
contains the SEND option code:

IPCNFRMD (X'06')
the SENDCNFD is being completed.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV

DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
contains one of the following values for the state of this path:

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPSEVER (X'06')
The path is in the Sever state.

Program Exceptions

The program exceptions for SENDCNFD are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an external interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check results when your virtual machine issues APPCVM SENDCNFD and it is not in the Confirm state or the Backout_Received state on this path. (You receive an IPRCODE if you issue a function from the wrong state.) See the list of APPCVM SENDCNFD return codes for all state check conditions.

No state change occurs when CC=1. State changes do occur when the function completes; that is, when control is returned to the virtual machine (CC=2). The state change depends on the value of IPWHATRC:

IPWHATRC Value	New State	Cause
IPCOMP	Receive	The SENDCNFD was in response to SENDCNF TYPE=NORMAL.
IPSNORM	Sever	The SENDCNFD was in response to SENDCNF TYPE=SEVER.
IPSNDCNF	Send	The SENDCNFD was in response to SENDCNF TYPE=PREPRECV.
IPCNFBK	*saved_state*	The SENDCNFD was issued from the BACKOUT_Received state to confirm a backout.

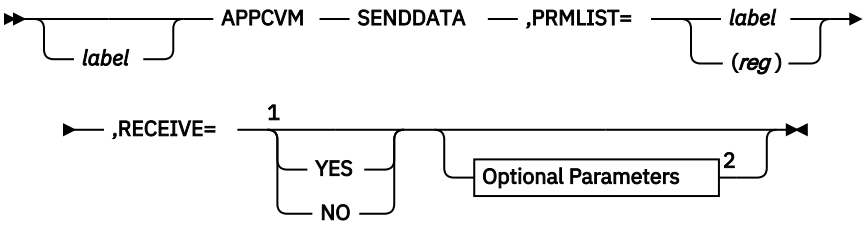
Completion Conditions

Because the SENDCNFD function completes immediately, you can issue another function on the same path as soon as your virtual machine regains control.

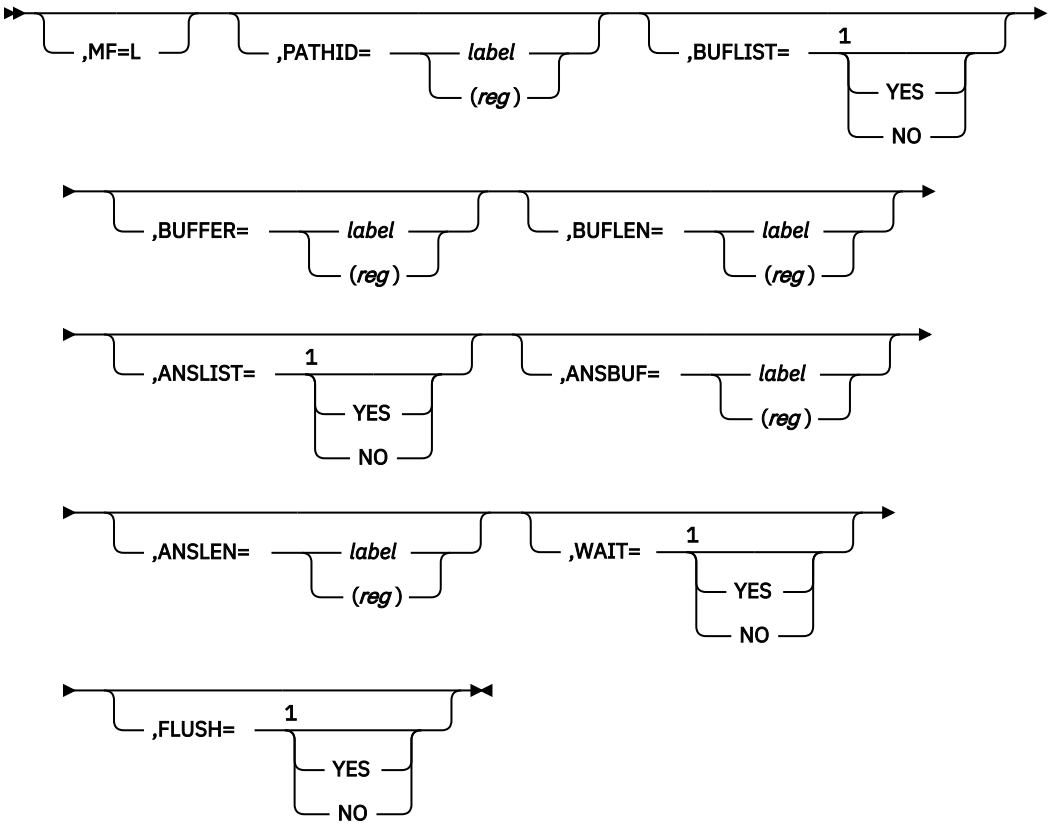
What Happens to Your VM Communication Partner

You can only issue SENDCNFD in response to a SENDCNF for SYNCLVL=CONFIRM or SYNCLVL=SYNCPT conversations. SENDCNFD always causes the completion of the SENDCNF. For SYNCLVL=SYNCPT conversations, SENDCNFD can also be issued in response to your partner's backout sync-point. If your communication partner issued SENDCNF with WAIT=NO, and your partner is enabled for function complete interrupts, then your partner gets a function complete interrupt.

APPCVM SENDDATA



Optional Parameters



Notes:

[1] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

[2] Optional parameters can be entered in any order.

Purpose

Use the SENDDATA function to send data from your program to your communication partner. You can also use this function to switch the conversation state from the Send state to the Receive state and to define an answer area for your partner's SENDDATA.

Parameters

Required Parameters:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

RECEIVE=
specifies whether to switch from Send to Receive state and define an answer area to get data back from your communication partner.

YES
switches the conversation state and defines an answer area.

Note: If you specify RECEIVE=YES, FLUSH=YES is invalid for this SENDDATA.

NO
keeps your program in the Send state and does not define an answer area.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SENDDATA.

PATHID=
specifies the path ID on which you send the data.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

BUFLIST=
specifies whether the BUFFER parameter refers to a single buffer address or an address for a list of buffers. (For more information, see Specifying Buffers on SENDDATA.)

YES
specifies that BUFFER refers to a list of addresses.

NO
specifies that BUFFER refers to a single address.

BUFFER=
specifies the areas in storage where CP gets the data to send. (See Specifying Buffers on SENDDATA.)

label
is the relocatable label for this storage area.

(*reg*)
is the register number that contains the address of this storage area.

BUFLEN=
specifies the total length, in bytes, of the data to be sent. See Specifying Buffers on SENDDATA.

Note that this length is not related to the length of a logical record, because the data does not have to be sent in complete logical records. See Setting Up the Data To Send.

label
is the relocatable label of the fullword that contains the length.

(*reg*)
is the register number that contains the length.

ANSLIST=
specifies whether the ANSBUF parameter refers to a single buffer address or an address for a list of buffers. (See Specifying Buffers on SENDDATA.)

YES
specifies that ANSBUF refers to a list of addresses.

NO
specifies that ANSBUF refers to a single address.

ANSBUF=
specifies the areas in storage where CP places data that is sent by your communication partner. (See Specifying Buffers on SENDDATA.)

label
is the relocatable label for this storage area.

(*reg*)
is the register number that contains the address of this storage area.

ANSLEN=
specifies the total length of the data (in bytes) sent by your communication partner. (See Specifying Buffers on SENDDATA.)

label
is the relocatable label of the fullword that contains the length.

(*reg*)
is the register number that contains the length.

WAIT=
specifies when control is returned to your virtual machine.

YES
returns control to your virtual machine when the SENDDATA is complete.

NO
returns control to your virtual machine when the SENDDATA request is initiated.

After you issue a SENDDATA with WAIT=NO, do not assume that the data is moved out of the buffer until you receive a function complete indication.

FLUSH=
specifies whether your local LU should flush its send buffer.

YES
causes your local LU to flush its send buffer when SENDDATA is complete.

Note: FLUSH=YES is invalid if you also specify RECEIVE=YES on this SENDDATA.

NO
does not cause your local LU to flush its send buffer when SENDDATA is complete.

Input Parameter List: The APPCVM SENDDATA parameter list has the input format shown in the following figure.

IPARML DSECT

	0	1	2	3	4	5	6	7
0	IPPATHID		IPFLAGS1	//////////////////////			IPFLAGS2	IPSENDOP
8	////////////////////////////////			IPBFADR1				
10	IPBFLN1F			////////////////////////////////				
18	////////////////////////////////			IPBFADR2				
20	IPBFLN2F			////////////////////////////////				

Figure 43. APPCVM SENDDATA Input Parameter List

IPPATHID
contains the path ID over which you send the data.

IPFLAGS1
contains one of the following input bit flags:

IPBUFLST (X'40')
BUFLIST was specified.

IPANSLST (X'08')
ANSLIST was specified.

IPAPPCSN (X'02')
The APPC SEND function was issued.

X'80'
this value is reserved for IBM use only.

IPFLAGS2
contains one of the following input bit flags:

IPWAIT (X'80')
A synchronous return is desired.

IPNOFLSH (X'40')
FLUSH=NO is specified. This flag is ignored unless IPSENDOP=IPDATA is set.

IPSENDOP
contains one of the following SEND option codes:

IPDATA (X'01')
You are sending data.

IPSNDRCV (X'02')
You are sending the data. The conversation is to be turned around, and an answer area is defined by IPBFADR2 and IPBFLN2F.

IPBFADR1
contains the address from which CP gets the data to send. This address is either the address of a buffer or the address of a list of buffer addresses. (See Specifying Buffers on SENDDATA.)

IPBFLN1F
contains the total length of the data being sent. This length is not related to the length of a logical record; it is used only to determine the length of the data to be moved by this SENDDATA. (See Specifying Buffers on SENDDATA.)

IPBFADR2
contains the address where CP places the answer data from your communication partner. This address is either the address of a buffer or the address of a list of buffer addresses. IPBFADR2 is only valid when IPSENDOP=IPSNDRCV. (See Specifying Buffers on SENDDATA.)

IPBFLN2F
contains the total length of the answer data received from your communication partner. IPBFADR2 is only valid when IPSENDOP=IPSNDRCV. (See Specifying Buffers on SENDDATA.)

Usage Notes

Setting Up the Data To Send: Using APPCVM SENDDATA, data is sent from a source program to a target program in buffers. APPC defines a **logical record** so that applications can communicate without depending on each other's buffering priorities and the priorities of any intermediate communication servers.

A logical record consists of a 2-byte logical record length field (LL) followed by a data field as shown in the following figure.

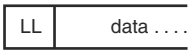


Figure 44. APPC Logical Record Format

The LL field, in its low-order 15 bits, contains the **total** length of the logical record. A discussion of the high-order bit of the LL field follows later in this section. This total logical record length includes the 2-bytes for the LL field, plus the length of the data field; as a result, the total logical record length must always be at least 2 bytes. If the data field has a length of 0, the logical record contains only the 2-byte length field.

The data field in a single logical record can range from 0 to 32,765 bytes long. If your program sends more than 32,765 bytes of data, it must break up the data into multiple logical records.

APPC defines two kinds of conversations, basic conversations and mapped conversations.

Basic Conversations: For ***basic conversations***, application programs send data directly in the APPC-defined logical record format. They need to include a 2-byte length field (LL) followed by the data field, as described above. For basic conversations, the LUs do not examine the high-order bit of the LL field. The following logical record length values are invalid, because the total length must be at least 2 and the high-order bit is ignored: X'0000', X'0001', X'8000', and X'8001'.

The length of the data that a program actually sends on a single APPCVM SENDDATA is independent of the logical record length. The data actually sent by a program on one SENDDATA could consist of one or more complete logical records, the beginning of a record, the middle of a record, or the end of a record. For example, you can specify any of the following combinations:

- One or more complete records, followed by the beginning of a record
- The end of a record, followed by one or more complete records
- The end of a record, followed by one or more complete records, followed by the beginning of a record
- The end of a record, followed by the beginning of a record.

Note: However, you only specify the 2-byte logical record length (LL) once per logical record—even if that logical record spans more than one SENDDATA.

Mapped Conversations: A program using the APPC/VM assembler language API and attempting to conduct a mapped conversation with its partner must be aware that CP does not encode or decode mapped conversation data streams; rather, it is the responsibility of the APPC/VM assembler application.

Note: CPI Communication (also known as SAA communications interface) performs the data stream encoding and decoding functions for the applications it serves. If you use CPI communication, you do not have to encode or decode mapped conversation data streams.

An APPC/VM assembler application allocates a mapped conversation by specifying TYPE=MAPPED on the APPCVM CONNECT macro (this results in the IPMAPPED bit in the IPFLAGS2 field of the CONNECT parameter list being set). A receiving APPC/VM assembler application can determine whether an inbound conversation is mapped by checking the IPMAPPED bit of the IPFLAGS2 field in the connection pending interrupt buffer.

A technique called *general data stream* organizes the data flowing on an APPC mapped conversation. (In fact, GDS formats organize not only APPC mapped conversations, but other line flows in SNA as well.) For more information on GDS formats, and on how GDS formats to organize APPC mapped conversations, see *Systems Network Architecture: Formats*, GA27-3136.

Specifying Buffers on SENDDATA: With APPCVM SENDDATA, you can specify a single buffer using one address and one length, or multiple buffers using a list of addresses and lengths.

When you specify a single buffer using one address and one length:

- BUFFER (or ANSBUF) specifies the address
- BUFLEN (or ANSLEN) specifies the length
- BUFLIST (or ANSLIST) must be equal to NO.

When you specify multiple buffers with a list of addresses and lengths:

- BUFFER (or ANSBUF) specifies the address of the list
- BUFLEN (or ANSLEN) specifies the sum of the lengths of the buffers in the list
- BUFLIST (or ANSLIST) must be equal to YES.

You must follow these conventions when you use address lists (BUFLIST=YES or ANSLIST=YES):

- The list must begin on a doubleword boundary.
- Each list entry must be two fullwords:
  - The first is the address of that portion of the list.
  - The second is the length of that portion of the list.

When you use an address list, the addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete. Also, APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified (by BUFLEN or ANSLEN).

Each application involved in the communication must determine the amount of data sent in each data stream. You can choose buffer sizes based on whatever is important to your application, such as the size of free storage buffers or efficient storage utilization.

Note: Even if you are responsible for both ends of the communication, do not code a receiving program so that the length of the receive buffer is based on the length of the send buffer.

When a program on the local system sends data, the length in the message pending interrupt is the actual length of the data sent. However, when a program on a remote system sends data, intermediate communication servers (like the TSAF virtual machine) could break up a single data stream into multiple data streams, or combine numerous data streams; as a result, the length shown in the target's message pending interrupt is the length of the data sent by the intermediate communication server, **not** the length of the data sent by the source program. For example, you could send a data stream of 0 length followed by a data stream of 100 bytes. However, the target program might get only one data stream message with the length of 100 bytes.

For basic conversations, your communication partner should examine the 2-byte logical record length field to determine how much data you sent on a logical record.

Condition Codes and Return Codes

CC=0

The SENDDATA started successfully, but has not yet completed. If your virtual machine is enabled for function complete interrupts, one is presented to your virtual machine when SENDDATA completes. The interrupt format is the same as the SENDDATA output parameter list (see CC=2, 3). When you get the function complete interrupt, check the IPAUDIT field for error information.

When control is returned to your virtual machine with CC=0, the parameter list may have been altered.

Note: CC=0 is not possible when WAIT=YES.

CC=1

An error occurred before the SENDDATA was initiated. The output parameter list is the same as the input shown in SENDDATA Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.
X'03'	3	A function is pending on this path.
X'06'	6	A protection exception occurred on your communication partner's **predefined** answer or receive area.
X'07'	7	An addressing exception occurred on your communication partner's **predefined** answer or receive area.

Hex Code	Decimal Code	Why the Error Occurred
X'0A'	10	Your send buffer length (BUFLEN) or answer buffer length (ANSLEN) is negative.
X'16'	22	Your communication partner's predefined answer list or receive list is invalid.
X'17'	23	A length specified in your send buffer list is negative.
X'18'	24	The total length that you specified (BUFLEN) is not the total of the lengths in your send buffer list.
X'1A'	26	The send buffer list does not begin on a doubleword boundary.
X'1B'	27	The answer buffer list does not begin on a doubleword boundary.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	SENDDATA is an invalid function from the Connect state.
X'22'	34	SENDDATA is an invalid function from the Receive state.
X'23'	35	SENDDATA is an invalid function from the Confirm state.
X'24'	36	SENDDATA is an invalid function from the Sever state.
X'26'	38	There was an invalid value in IPSENDOP field.
X'2A'	42	There is an invalid logical record length in your data stream.
X'2C'	44	You started, but did not finish, sending a logical record. (This can only occur if you specified RECEIVE=YES on the SENDDATA.)
X'38'	56	WAIT=YES was specified on a function issued to this same virtual machine.
X'44'	68	SENDDATA is invalid from the Reset state.
X'45'	69	SENDDATA FLUSH=NO or SENDDATA FLUSH=YES, BUFLEN > 0 is invalid from the Defer_Receive state.
X'46'	70	SENDDATA is invalid from the Defer_Sever state.
X'47'	71	SENDDATA is invalid from the Prepare_Received state.
X'49'	73	SENDDATA is invalid from the Unsolicited_Request_Commit_Received state.
X'4B'	75	SENDDATA is invalid from the Backout_Received state.
X'4C'	76	SENDDATA is invalid from the Backout_Required state.
X'5B'	91	A paging or storage error was detected in the SEND data area.
X'5C'	92	A paging or storage error was detected in the RECEIVE data area.
X'5D'	93	A paging or storage error was detected in the ANSWER data area.
X'5E'	94	A paging or storage error was detected in the REPLY data area.

The following return codes can only occur if your communication partner defined an answer area or receive area before you issued the SENDDATA:

X'06', X'07', X'16', X'17', X'18', X'2A', and X'2C'.

If your communication partner did not define an answer area or receive area before you issued the SENDDATA, CP reports the above error conditions (X'06', X'07', X'16', X'17', X'18', X'2A', and X'2C') to you in the IPAUDIT flags of your SENDDATA output parameter list, when your partner's RECEIVE completes.

Your partner learns of the error through one of the following:

- A protection exception from CP
- An addressing exception from CP

- A return code on the RECEIVE
- The IPAUDIT flags of the RECEIVE output parameter list, when the RECEIVE completes.

For the following return codes, data may be copied to your communication partner's virtual machine before the error is detected (the amount of data copied in this case is unpredictable):

X'16', X'17', X'18', X'2A', and X'2C'

Note that for CC=1, other fields in the parameter list might have been altered.

CC=2 or

CC=3

The SENDDATA completed. (Also see SENDDATA Completion.) When CC=2, then the function completed with no errors; when CC=3, there is some error information in IPAUDIT.

Note: When WAIT=NO, CC=3 is not possible.

The output parameter list when CC=2 or 3 is shown in the following figure.

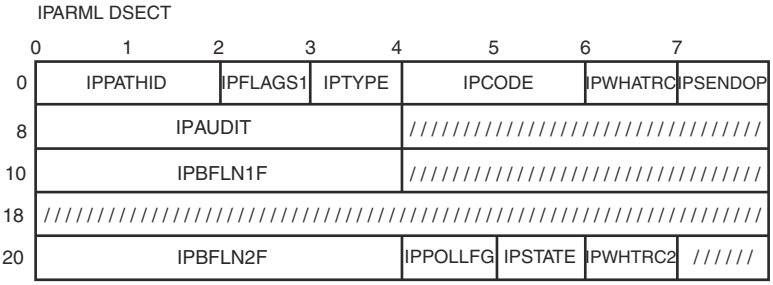


Figure 45. APPCVM SENDDATA Output Parameter List (Function Complete Interrupt)

IPPATHID
contains the path ID on which the function is complete.

IPFLAGS1
may contain the following output bit flag:

X'20'
This value is reserved for IBM use only.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
contains the error or sever code from the partner's SENDERR or SEVER. IPCODE is only valid when IPWHATRC=IPERROR or IPSABEND. See "APPC/VM Sever, Error, and Sense Codes That You Can Get" on page 399.

IPWHATRC
contains the what-received code. For RECEIVE=YES or RECEIVE=NO:

IPCOMP (X'00')
Either of the following has occurred:

- The SENDDATA RECEIVE=NO completed normally.
- The SENDDATA RECEIVE=YES or NO completed with an error on your SEND buffer or on your partner's answer or RECEIVE buffer. See the IPAUDIT description.

IPERROR (X'03')
Your partner issued SENDERR.

IPSABEND (X'09')
Your partner issued a SEVER TYPE=ABEND.

IPBACK (X'14')
The function was completed with an indication that your partner initiated a CRR backout sync-point. You should back out the CMS work unit of which this conversation is a part.

For RECEIVE=YES only:

IPDATA (X'01')
Only data was received.

IPSEND (X'02')
Your partner switched the conversation around, and you are now in the Send state.

IPCNFRM (X'04')
Your partner is requesting confirmation.

IPCNFSEV (X'05')
Your partner is requesting confirmation that it can issue a SEVER.

IPSNORM (X'08')
Your partner issued a SEVER TYPE=NORMAL.

IPSNDCNF (X'0C')
Your partner is requesting confirmation that it can enter the Receive state.

IPPREPAR (X'0F')
The function was completed with an indication that your partner initiated a CRR commit sync-point. You should commit the CMS work unit of which this conversation is a part. (This is not possible when RECEIVE=NO.)

IPREQCOM (X'10')
The function was completed with a SENDRQCM indication. (This is not possible when RECEIVE=NO.)

Notes:

1. When SENDDATA RECEIVE=YES is specified, data might have been received for any IPWHATRC value.
2. You do not get a nondata indication in IPWHATRC until you do a SENDDATA RECEIVE=YES for all the data that your partner (or intermediate communication server) sent with or before the nondata function. For example, if both of the following were true:
   a. You did a SENDDATA RECEIVE=YES with a 199-byte answer area
   b. Your partner (or intermediate communication server) issued SENDDATA RECEIVE=YES with a data length of 200 bytes.

   IPWHATRC would be IPDATA. When you issue a RECEIVE for the 200th byte, IPWHATRC would become IPSEND.

IPSENDOP
Contains one of the following SEND option codes:

IPDATA (X'01')
Your SENDDATA RECEIVE=NO is completing.

IPSNDRCV (X'02')
Your SENDDATA RECEIVE=YES is completing.

IPAUDIT
Has four fields that may contain error information.

Note: In the following descriptions,

- **Send area** refers to either a send buffer specified directly on an APPCVM SENDDATA, BUFFER= or a send buffer that is part of a buffer list.
- **Answer area** refers to either an answer buffer specified directly on an APPCVM SENDDATA, ANSBUF= or an answer buffer that is part of a buffer list.
- **Receive area** refers to either a receive buffer specified directly on an APPCVM RECEIVE, BUFFER= or a receive buffer that is part of a buffer list.

(See Specifying Buffers on SENDDATA for more information.)

IPAUDIT1 (first byte of IPAUDIT)
May contain one of the following bit flags:

IPADSNPX (X'40')
A protection exception occurred on your send area. This only applies if your partner did not have a receive area defined when your data was sent.

IPADSNAX (X'20')
An addressing exception occurred on your send area. This only applies if your partner did not have a receive area defined when your data was sent.

IPADANPX (X'10')
A protection exception occurred on your answer area.

IPADANAX (X'08')
An addressing exception occurred on your answer area.

IPAUDIT2 (second byte of IPAUDIT)
May contain one of the following bit flags:

IPADRCPX (X'80')
A protection exception occurred on your communication partner's receive area (if your partner did not have the receive area defined when your data was sent), or your communication partner's answer area.

IPADRCAX (X'40')
An addressing exception occurred on your communication partner's receive area (if your partner did not have the receive area defined when your data was sent), or your communication partner's answer area.

IPADRPPX (X'20')
A protection exception occurred on your communication partner's send area.

IPADRPAX (X'10')
An addressing exception occurred on your communication partner's send area.

IPADRLST (X'04')
Your communication partner had an invalid send, answer, or receive buffer list.

IPAUDIT3 (third byte of IPAUDIT)
May contain one of the following bit flags:

IPADBLEN (X'80')
A bad length is in your send buffer list.

IPADALEN (X'40')
A bad length is in your answer buffer list.

IPADBTOT (X'20')
Your total send buffer length is invalid.

IPADATOT (X'10')
Your total answer buffer length is invalid.

IPADTINV (X'08')
There is an invalid logical record length in your communication partner's data stream.

IPADIINV (X'04')
There is an invalid logical record length in your data stream.

IPADTTRN (X'02')
Your communication partner started, but did not finish, sending a logical record and tried to change to the Receive state.

IPADITRN (X'01')
You started, but did not finish, sending a logical record and you tried to change to the Receive state.

IPASYRC (fourth byte of IPAUDIT)
May contain one of the following error codes (for which an appropriate IPRCODE was given to your communication partner):

Hex Code	Decimal Code	Meaning
X'5B'	91	A paging or storage error was detected in the SEND data area
X'5C'	92	A paging or storage error was detected in the RECEIVE data area.
X'5D'	93	A paging or storage error was detected in the ANSWER data area.
X'5E'	94	A paging or storage error was detected in the REPLY data area.

Note: IPRCODEs X'5D' and X'5E' are possible on a SENDDATA when the partner issues a RECEIVE prior to the SENDDATA (RECEIVE ahead).

IPBFLN1F
Contains the length of pending log data for you to receive. This field is only meaningful when IPWHATRC is equal to IPSABEND or IPERROR.

IPBFLN2F
Contains one of the following depending on the value of IPWHATRC. If IPWHATRC is:

- Equal to IPDATA, IPBFLN2F contains the number of bytes that were sent by your communication partner, but did not fit into the defined answer area. This length is not the length of the APPC data stream being sent, rather, it is the length of the data that has arrived and is ready to receive.
- Not equal to IPDATA, IPBFLN2F contains the number of bytes left in your defined answer area.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
Contains the current state for this path, which may be any of the following values:

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPCONFRM (X'05')
The path is in the Confirm state.

IPSEVER (X'06')
The path is in the Sever state.

IPREPARE (X'09')
The path is in the Prepare_Received state.

IPURQCMT (X'0B')
The path is in the Unsolicited_Request_Commit_Received state.

IPBKOUT (X'0D')
The path is in the Backout_Received state.

IPBKREQ (X'0E')
The path is in the Backout_Required state.

IPWHTRC2
Further qualifies the IPWHATRC=IPPREPAR and IPWHATRC=IPREQCOM:

IPTPSEND (X'03')
Your partner requests to be in the Send state upon the successful completion of the commit sync-point.

IPTPRECV (X'04')
Your partner requests to be in the Receive state upon the successful completion of the commit sync-point.

IPTPSEVR (X'06')
Your partner requests to be in the Sever state upon the successful completion of the commit sync-point.

Program Exceptions

The program exceptions for SENDDATA are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine. An addressing exception also occurs for any of the following: • An invalid buffer address in SENDDATA's parameter list • An invalid buffer list address in SENDDATA's parameter list • An invalid buffer address in SENDDATA's buffer list. This only applies if your partner had a receive area defined when you sent the data. (Answer areas are always predefined.)
Operation	Either an external interrupt buffer was not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user. A protection exception also occurs for any of the following: • An invalid buffer address in SENDDATA's parameter list • An invalid buffer list address in SENDDATA's parameter list • An invalid buffer address in SENDDATA's buffer list. This only applies if your partner had a receive area defined when you sent the data. (Answer areas are always predefined.)
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check results when your virtual machine issues APPCVM SENDDATA and it is not in the Send state for this path. (You also receive an IPRCODE since you issued a function from an improper state.) A state check also occurs (IPRCODE=X'2C') if you started, but did not finish, sending a logical record on this path at the completion of the send portion of your SENDDATA RECEIVE=YES. See the list of APPCVM SENDDATA return codes for all state check conditions.

When you issue SENDDATA RECEIVE=YES, your communication partner receives notice of this as if you had issued the following sequence of functions:

1. SENDDATA RECEIVE=NO
2. RECEIVE.

When the receive part of the SENDDATA RECEIVE=YES begins, you should have completed sending any outstanding logical records. For example, the following sequence would cause an error:

1. You issue SENDDATA RECEIVE=YES BUFLEN=999 to send a logical record with a logical record length of 1000 bytes.
2. Your partner does a RECEIVE of the 999 bytes.
3. The receive portion of your SENDDATA RECEIVE begins.

In this situation, the error is caused because you did not send all 1000 bytes; therefore, you did not complete sending the outstanding logical record.

No state change occurs when CC=1. State changes occur when:

- The function completes; that is, control returns to the virtual machine (CC=2 or 3).
- The function complete interrupt is accepted by the virtual machine (CC=0) or you complete the function using TESTCMPL.

The state change depends on the IPWHATRC value:

IPWHATRC Value	New State	Cause
IPCOMP	No state change occurs.	Either of the following could be the cause: • The SENDDATA RECEIVE=NO has completed normally. • The SENDDATA RECEIVE=YES or NO completed with an error on your SEND buffer or on your partner's answer or RECEIVE buffer.
IPCOMP	Receive	The SENDDATA FLUSH=YES is issued from the Defer_Receive state.
IPDATA	Receive	Either of the following could be the cause: • The SENDDATA RECEIVE=YES with a nonzero length answer area was completed by your partner sending data. • The SENDDATA RECEIVE=YES with a 0 length answer area was completed by your partner receiving the data sent.
IPSEND	Send	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a RECEIVE or SENDDATA RECEIVE=YES.
IPERROR	Receive	The SENDDATA was completed by the communication partner issuing a SENDERR.
IPCNFRM	Confirm	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a SENDCNF TYPE=NORMAL.
IPCNFSEV	Confirm	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a SENDCNF TYPE=SEVER.
IPSNDCNF	Confirm	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a SENDCNF TYPE=PREPRECV.
IPSNORM	Sever	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a SEVER TYPE=NORMAL.
IPSABEND	Sever	The SENDDATA was completed by the communication partner issuing a SEVER TYPE=ABEND.

IPWHATRC Value	New State	Cause
IPPREPAR	Prepare_Received	Your communication partner initiated a commit sync-point to complete the SENDDATA.
IPREQCOM	Unsolicited_Request_ Commit_Received	Your communication partner initiated a commit sync-point to complete the SENDDATA.
IPBACK	Backout_Received	Your communication partner issued a SENDBACK to complete the SENDATA.

Completion Conditions

After issuing a SENDDATA, you cannot issue another SEND (see note), RECEIVE, or SEVER TYPE=NORMAL on that path until the outstanding SENDDATA is complete.

Note: SEND generally refers to all of the APPC/VM send functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.

When the SENDDATA completes for a communication program depends on the value you give to the RECEIVE parameter of SENDDATA:

- When RECEIVE=NO, your SENDDATA is complete when all of the data is copied out of your SEND buffer, or when your communication partner issues a SENDERR or a SEVER.
- Your SENDDATA completes immediately when RECEIVE=NO, FLUSH=NO, and you specify a 0-length send buffer.
- When RECEIVE=NO, FLUSH=YES, and you specify a 0-length send buffer (this is called a **pure flush**), your SENDDATA completes immediately, unless your partner is notified of the flush. (See IPFLUSH (X'40') in the IPFLAGS1 Return Code for “APPCVM RECEIVE” on page 451 for more information.)
- When RECEIVE=YES and you specify a 0 answer area, your SENDDATA is complete when all of the data is copied out of your SEND buffer.
- When RECEIVE=YES and you specify a nonzero answer area, then the SENDDATA is complete when all of the data is copied out of your SEND buffer and your communication partner:
  - Sends messages to your virtual machine to completely fill the answer area specified on your virtual machine's SENDDATA
  - Issues RECEIVE, SENDCNF, SENDDATA RECEIVE=YES, SENDERR, or SEVER.

Remember, when you specify SENDDATA RECEIVE=YES with a nonzero answer area length, you get one function complete interrupt when your communication partner or an intermediate communication server issues a function. But, when you specify SENDDATA RECEIVE=NO followed by a RECEIVE, you receive two function complete interrupts. The first interrupt is a result of the data being copied out of your SEND buffer; the second interrupt is when your RECEIVE is completed.

What Happens to Your VM Communication Partner

Note: If an intermediate communication server (like TSAF, ISFC, or AVS) handles communication between you and your partner, the information in this section also describes what happens to the intermediate server when you issue SENDDATA.

When you issue an APPCVM SENDDATA, your communication partner's outstanding function may complete, or your partner may get a message pending interrupt:

- If your partner has a SENDERR outstanding on its half of the path, your partner's function completes.
- If your partner has a SENDDATA RECEIVE=YES or RECEIVE outstanding on its half of the path and you issued SENDDATA RECEIVE=YES or RECEIVE, then your partner's function completes.

- If your partner has a SENDDATA RECEIVE=YES outstanding on its half of the path and you issued SENDDATA RECEIVE=NO, your partner's function completes only when you send enough data to fill your partner's predefined receive area.
- Your partner gets a message pending interrupt if it:
  - Has no function outstanding on its half of the path
  - Is in the Receive state on its half of the path
  - Is enabled for message pending interrupts.

Message Pending External Interrupt: Your program can get a message pending interrupt when your communication partner issues APPCVM SENDDATA, RECEIVE, SENDCNF, SENDERR, or initiates a commit or backout sync-point.

The APPC/VM message pending external interrupt has the format shown in the following figure.

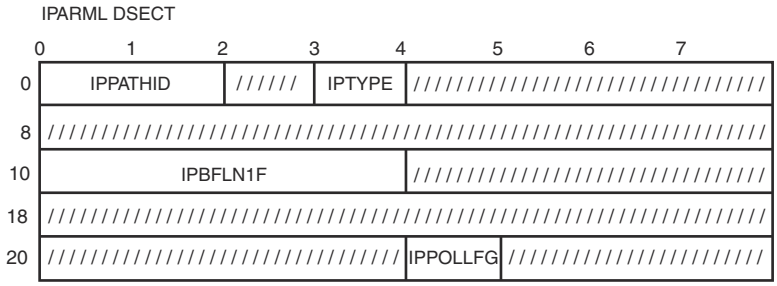


Figure 46. Message Pending External Interrupt

IPPATHID
contains the path ID on which a message is pending.

IPTYPE
contains the interrupt type for a message pending (IPTYPMPA, X'89').

IPBFLN1F
contains the length of the pending message. This length is the length of the data that has arrived and is ready to receive.

When the data is sent by a virtual machine on your system, IPBFLN1F specifies the entire length of the data sent. When the data is sent by a virtual machine on a remote system, IPBFLN1F specifies only the length of the data sent by the communication server (typically TSAF or AVS).

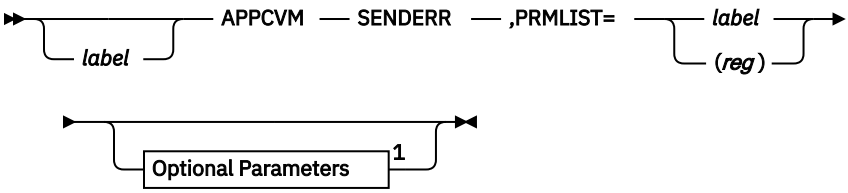
IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

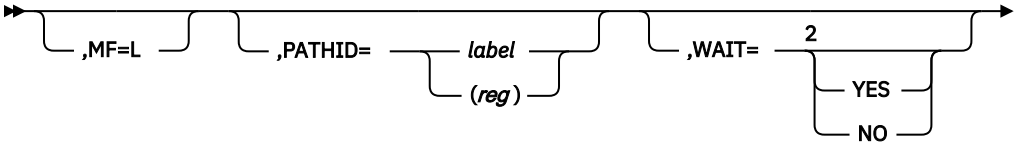
Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

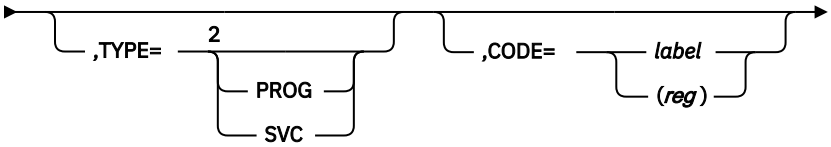
Your program gets a message pending interrupt for a path only when its half of the path is in the Receive state. If your program is not in the Receive state on that path, the message pending interrupt is kept pending until your half of the path enters the Receive state; then you get the message pending interrupt. When you receive the message, check the condition code or the IPWHATRC field to find what to do next. See “APPCVM RECEIVE” on page 451 for more information.

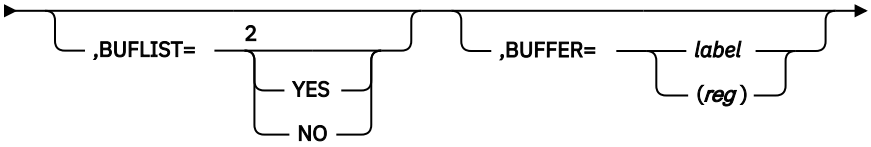
APPCVM SENDERR (Send Error)

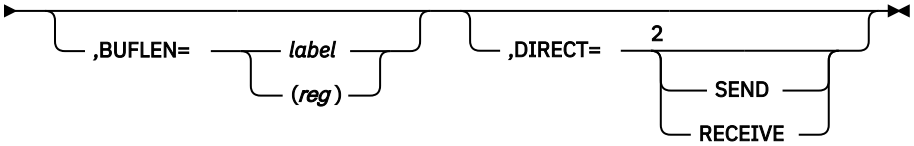


Optional Parameters









Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value from the last time the instruction was issued.

Purpose

Use the SENDERR function to notify your communication partner that an error has occurred and causes a break in a typical send/receive sequence. You can also issue SENDERR as a negative response to a confirmation request (SENDCNF). After issuing SENDERR, your program can send error information to your communication partner using SENDDATA.

Your program can optionally specify log data directly on the SENDERR call.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SENDERR.

PATHID=
specifies the path ID of the path on which you send the error notice.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

WAIT=
specifies when control is returned to your virtual machine.

YES
returns control to your virtual machine when the SENDERR is complete.

NO
returns control to your virtual machine as soon as you issue the SENDERR request. When the SENDERR completes, you are notified with a function complete interrupt.

TYPE=
describes the level of error that CP reports to your communication partner. CP generates the appropriate error code to correspond to this value. TYPE is invalid if CODE is specified.

PROG
indicates that a user application program error is being reported. The error code will be one of the following: X'0410', X'0420', or X'0430'.

SVC
indicates that an LU services error is being reported. For example, this error type is used by the programs providing mapped conversation support to report errors. (CP does not check to ensure that SENDERR TYPE=SVC is issued on a mapped conversation.) The error code sent will be one of the following: X'0510', X'0520', or X'0530'.

See Error Codes that CP and Communication Servers Can Issue for the APPC meanings of these error codes.

CODE=
specifies a 2-byte error code that a communication server sends to your communication partner. CODE is invalid if TYPE is specified.

label
is a relocatable label in the storage area that contains the error code.

(*reg*)
is the register number that contains the error code in the low-order halfword.

Only communication servers (authorized by OPTION COMSRV in their directory entries) can specify CODE. Communication servers should use the APPC/VM-defined error codes; they should not define error codes for their own use. See Error Codes that CP and Communication Servers Can Issue.

If your program is not a communication server, it should just issue the APPCVM SENDERR function without an error code; CP generates the appropriate error code based on the states of the programs.

BUFLIST=
specifies the type of buffer address to which the BUFFER parameter refers.

YES
refers to a list of addresses.

NO
refers to a single address.

BUFFER=
specifies the address of the areas from which CP takes the log data.

label
is the relocatable label in storage where CP gets the log data to send.

(*reg*)
is the register number that contains the address of the storage area. This storage area is where CP gets the log data to send.

BUFLEN=
specifies the length of the areas from which APPC/VM takes the log data to be sent. The minimum length is eight bytes; the maximum length is 600 bytes. This length is not related to the length of a logical record.

label
is the relocatable label of the fullword that contains the length.

(*reg*)
is the register number that contains the length.

DIRECT=
specifies the state (SEND or RECEIVE) your program was in when you detected the error. Specifying the state ensures that the partner gets the correct error code.

DIRECT is ignored unless this is the first function issued after receiving both data and a SEND indication on the last function to complete. DIRECT is invalid if CODE is specified.

SEND
indicates that the error was detected while the program was preparing to send data. The error code issued for this case indicates that the logical record was not truncated.

RECEIVE
indicates that the error was detected while the program was receiving the data. The error code issued for this case indicates that the remaining data was purged.

Input Parameter List: The APPCVM SENDERR parameter list has the input format shown in the following figure.

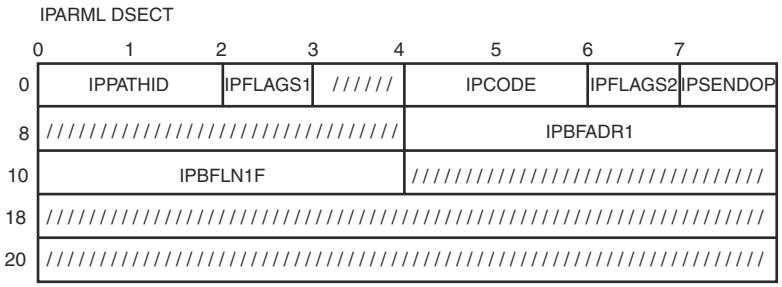


Figure 47. APPCVM SENDERR Input Parameter List

IPPATHID
contains the path ID over which to send the SENDERR.

IPFLAGS1
contains one of the following input bit flags:

IPBUFLST (X'40')
You specified the buffer list option.

IPAPPCSN (X'02')
The APPC SEND function was issued.

IPCODE
contains the error code you are sending on the APPCVM SENDERR. IPCODE is only valid when IPCOMSRV is set. Refer to Error Codes that CP and Communication Servers Can Issue.

IPFLAGS2
contains one of the following input bit flags:

IPWAIT (X'80')
You specified a synchronous return.

IPCHGDIR (X'40')
You specified DIRECT=RECEIVE.

IPCOMSRV (X'20')
The SENDERR is being issued for another user. Only an authorized communication server (OPTION COMSRV in the directory entry) can specify IPCOMSRV. When you do specify IPCOMSRV, CP does not generate a SENDERR code, but, instead, uses the one that you provide. It is your responsibility to ensure that the code is appropriate.

IPTYPSVC (X'10')
You specified an LU services error.

IPSENDOP
contains the SEND option code:

IPERROR (X'03')
indicates the SENDERR.

IPBFADR1
contains the address of the area from which APPC/VM takes one of the following:

- The log data
- The address of a list of buffer addresses.

See Specifying Log Data.

IPBFLN1F
contains the length of the log data being sent. If IPBFLN1F is 0, then no log data is being sent. Otherwise, log data is being sent.

Usage Notes

Specifying Log Data: An application program can choose to set up log data and send that log data using buffers specified on APPCVM SENDERR. Log data conveys error information to an LU, where it is added to the system error log. This error information can be used in debugging and error recovery.

When using APPCVM SENDERR to send log data to your communication partner, you use buffers. You can specify a single buffer using one address and one length or specify multiple buffers using a list of addresses and lengths.

When you specify a single buffer using one address and one length,

- BUFFER specifies the address
- BUFLEN specifies the length
- BUFLIST must be equal to NO.

When you specify multiple buffers with a list of addresses and lengths,

- BUFFER specifies the address of the list
- BUFLEN specifies the sum of the lengths of the buffers in the list
- BUFLIST must be equal to YES.

When specifying address lists (BUFLIST=YES), note the following:

- The list must begin on a doubleword boundary.

- Each list entry must be two fullwords; the first is the address of that portion of the list, and the second is the length.
- The addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete.
- APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified by BUFLEN.

The log data you are sending in buffers must be coded into an error log general data stream (GDS) variable. An error log GDS variable has the format as defined by SNA LU 6.2 and shown in the following figure.

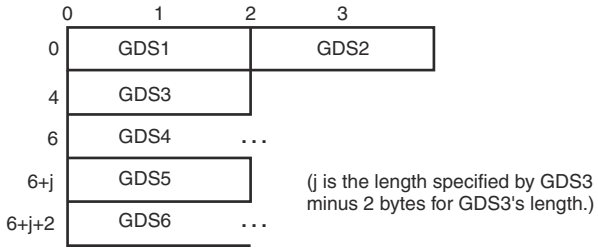


Figure 48. Error Log GDS Variable Format

GDS1
contains the length, in binary, of the error log GDS variable, including this length field. This must be in the range from 8 to 600 bytes.

GDS2
contains a GDS identifier for an error log variable, X'12E1'.

GDS3
contains the length, in binary, of the product set ID (GDS4) including this length field.

Note: The length field is always present; a value of 2 indicates no product set ID subvector follows.

GDS4
contains the product set ID subvector. This subvector, which is variable length, uniquely identifies the product.

GDS5
contains the length, in binary, of the log data including this length field.

Note: The length field is always present; a value of 2 indicates no log data follows.

GDS6
contains the log data.

For more information about the error log GDS variable and product set ID, see *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*.

Error Codes That CP and Communication Servers Can Issue: See Table 75 on page 494 for the codes that CP or a communication server can issue on an APPCVM SENDERR.

This table also shows the APPC error condition associated with each error code. For more information on what these APPC conditions mean and when to use these error codes, see *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*.

Note: A communication server within a TSAF collection can specify only codes X'0410', X'0420', and X'0430'.

Table 75. APPC/VM-Defined SENDERR Codes

APPC/VM Code	APPC Error Condition
X'0410'	PROG_ERROR_NO_TRUNC

Table 75. APPC/VM-Defined SENDERR Codes (continued)

APPC/VM Code	APPC Error Condition
X'0420'	PROG_ERROR_TRUNC
X'0430'	PROG_ERROR_PURGING
X'0510'	SVC_ERROR_NO_TRUNC
X'0520'	SVC_ERROR_TRUNC
X'0530'	SVC_ERROR_PURGING

If your program is not a communication server, it should just issue the APPCVM SENDERR function without an error code; CP generates the appropriate error code based on the states of the programs.

Sever Codes That You May Receive in Response to a SENDERR: If the SENDERR was completed by the communication partner issuing a SEVER, a sever code is returned. The sever code you get when your SENDERR completes depends on the state you are in when you issue SENDERR. If you are in the:

- Send state, you may get any valid sever code.
- Receive state, you may get either sever code X'0610' or X'0620'. You would receive notice of any other sever condition on a subsequent APPC/VM function on which an error can be reported.

  In addition, you get an indication of SEVER TYPE=NORMAL (IPWHATRC=IPSNORM) when your SENDERR completes, if your communication partner issued SEVER TYPE=NORMAL.

  You also get an indication of SEVER TYPE=NORMAL when your SENDERR completes, if your partner issued SEVER TYPE=ABEND with any of the sever codes X'210', X'220', X'230'. In this case, you will not see the sever code.
- Confirm state, you may get either sever code X'0610' or X'0620'. You would receive notice of any other sever condition on a subsequent APPC/VM function on which an error can be reported.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	X

CC=0

SENDERR started successfully, but has not yet completed. When the function completes, a function complete interrupt is presented to your virtual machine. The function complete interrupt has the same format as the SENDERR output parameter list (see CC=2, 3). When you get the function complete interrupt, check the IPAUDIT field for error information.

Note: When you specify WAIT=YES, CC=0 is not possible.

CC=1

An error occurred. The output parameter list is the same as the input shown in SENDERR Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.
X'03'	3	A function is pending on this path.
X'0A'	10	Your buffer length is negative.

Hex Code	Decimal Code	Why the Error Occurred
X'17'	23	A length specified in your SEND buffer list is negative.
X'18'	24	The total length that you specified is not the total of the lengths in your SEND buffer list.
X'1A'	26	The buffer list address is not on a doubleword boundary.
X'1D'	29	You are not authorized to act for another user.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	SENDERR is an invalid function from the Connect state.
X'24'	36	SENDERR is an invalid function from the Sever state.
X'26'	38	There is an invalid value in IPSENDOP field.
X'33'	51	You must receive pending log data before issuing SENDERR.
X'37'	55	Your buffer length is either less than eight or greater than 600.
X'38'	56	WAIT=YES was specified on a function issued to this same virtual machine.
X'44'	68	SENDERR is invalid from the Reset state.
X'45'	69	SENDERR is invalid from the Defer_Receive state.
X'46'	70	SENDERR is invalid from the Defer_Sever state.
X'4B'	75	SENDERR is valid from the Backout_Received state only when issued by a communication server.
X'4C'	76	SENDERR is invalid from the Backout_Required state.

CC=2 or

CC=3

SENDERR completed (also see SENDERR Completion). When CC=2, the function completed with no errors; when CC=3, there is error information in IPAUDIT.

Note: When you specify WAIT=NO, CC=3 is not possible.

The output parameter list when CC=2 or 3 is shown in the following figure.

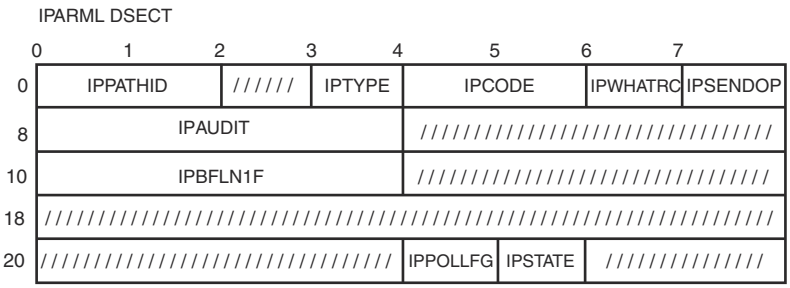


Figure 49. APPCVM SENDERR Output Parameter List (Function Complete Interrupt)

IPPATHID
contains the path ID on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
contains the error or sever code from the partner's SENDERR or SEVER. IPCODE is only valid when IPWHATRC=IPERROR or IPSABEND. See Sever Codes That You May Receive in Response to a SENDERR. Also see "APPC/VM Sever, Error, and Sense Codes That You Can Get" on page 399 for a complete list of the error and sever codes.

IPWHATRC
contains the what-received code:

IPCOMP (X'00')
SENDERR completed with nothing received.

IPERROR (X'03')
Your partner issued SENDERR.

IPSNORM (X'08')
Your partner issued a SEVER TYPE=NORMAL, or SEVER TYPE=ABEND with sever codes X'0210', X'0220', or X'0230'.

IPSABEND (X'09')
Your partner issued a SEVER TYPE=ABEND with sever codes other than X'0210', X'0220', or X'0230'.

IPBACK (X'14')
The function completed with an indication that your partner initiated a CRR backout sync-point. You should back out the CMS work unit of which this conversation is a part.

IPSENDOP
contains the SEND option code:

IPERROR (X'03')
Means that the SENDERR is being completed.

IPAUDIT
has four fields that may contain error information.

Note: In the following descriptions:

- *Send area* refers to either a send buffer specified directly on an APPCVM SENDERR, BUFFER= or a send buffer that is part of a buffer list. These buffers are used for sending log data.
- *Receive area* refers to either a receive buffer specified directly on your partner's APPCVM RECEIVE, BUFFER= or a receive buffer that is part of a buffer list. These buffers are used for receiving log data.

(See Specifying Log Data for more information on specifying buffers and buffer lists for log data.)

IPAUDIT1 (first byte of IPAUDIT)
may contain one of the following bit flags:

IPADSNPX (X'40')
A protection exception occurred on your send area.

IPADSNAX (X'20')
An addressing exception occurred on your send area.

IPAUDIT2 (second byte of IPAUDIT)
may contain one of the following bit flags:

IPADRCPX (X'80')
A protection exception occurred on your communication partner's receive area for log data.

IPADRCAX (X'40')
An addressing exception occurred on your communication partner's receive area for log data.

IPADRLST (X'04')
Your communication partner had an invalid receive buffer list.

IPAUDIT3 (third byte of IPAUDIT)
may contain one of the following bit flags:

IPADBLEN (X'80')
A bad length is in your send buffer list.

IPADBTOT (X'20')
Your total send buffer length is invalid.

IPADIINV (X'04')
There is an invalid logical record length in your data stream.

IPASYRC (fourth byte of IPAUDIT)
Reserved.

IPBFLN1F
contains the length of pending log data for you to receive. This field is only meaningful when IPWHATRC is equal to IPSABEND or IPERROR.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
Contains the current state for this path, which may be one of the following values:

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPSEVER (X'06')
The path is in the Sever state.

IPBKOUT (X'0D')
The path is in the Backout_Received state.

IPBKREQ (X'0E')
The path is in the Backout_Required state.

Program Exceptions

The program exceptions for SENDERR are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an external interrupt buffer was not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check results when APPCVM SENDERR is issued from an improper state. (You also receive an IPRCODE when you issue a function from the wrong state.) See the list of APPCVM SENDERR return codes for all state check conditions.

No state change occurs when CC=1. State changes occur when:

- The function completes, that is, when control is returned to the virtual machine (CC=2 or CC=3).
- The function complete interrupt is accepted by the virtual machine or you use TESTCMPL to discover that the function was completed.

The state change depends on the IPWHATRC value in the output parameter list:

IPWHATRC Value	State	Cause
IPCOMP	Send	The SENDERR has completed.
IPERROR	Receive	The SENDERR was completed by the communication partner issuing a SENDERR from the Receive state.
IPSNORM	Sever	The SENDERR was completed by the communication partner issuing a SEVER TYPE=NORMAL, or SEVER TYPE=ABEND with sever codes X'0210', X'0220', or X'0230'.
IPSABEND	Sever	The SENDERR was completed by the communication partner issuing a SEVER TYPE=ABEND with sever codes other than X'0210', X'0220', or X'0230'.
IPBACK	Backout_ Received	The SENDERR was completed by the communication partner initiating a backout sync-point. Your program must respond with a SENDCNFD.

Completion Conditions

After you issue a SENDERR, you cannot issue another SEND (see note), RECEIVE, or SEVER TYPE=NORMAL on that path until the outstanding SENDERR is complete. SENDERR is complete when your communication partner or an intermediate communication server gets notified of the SENDERR by an IPERROR indication in IPWHATRC (in the function complete interrupt).

Note: In this case, SEND refers to the set of APPC/VM send functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.

APPC/VM notifies your communication partner of the SENDERR when your partner's SENDDATA, SENDCNF, SENDERR, or RECEIVE completes. SENDERR causes your partner's outstanding functions to complete. If none of these functions are outstanding when the SENDERR is issued, the SENDERR indication will not be presented to the communications partner until the next function is issued.

If your communication partner is in the Receive state and sends a SENDERR before it receives your SENDERR notice, your partner's SENDERR is invoked over yours. In this case, your partner would enter the Send state, and you would be switched to the Receive state.

If your communication partner specified that it is willing to receive log data (LOGDATA=YES on either APPCVM CONNECT or IUCV ACCEPT, whichever is applicable) your SENDERR completes when all of the log data is copied out of your buffer. If your partner specified that it would not receive log data (LOGDATA=NO on either APPCVM CONNECT or IUCV ACCEPT, whichever is applicable), the log data is considered to be copied out of your buffer and the SENDERR completes.

When SENDERR completes, CP resets your logical record count to zero, as well as your communication partner's; that is, your next SENDDATA would be a new logical record.

What Happens to Your VM Communication Partner

Your communication partner's outstanding function may complete, or your partner may get a message pending interrupt.

If your partner has a RECEIVE, SENDDATA, SENDCNF, or SENDERR outstanding on its half of the path, your partner's function is completed.

Your partner gets a message pending interrupt, if your partner:

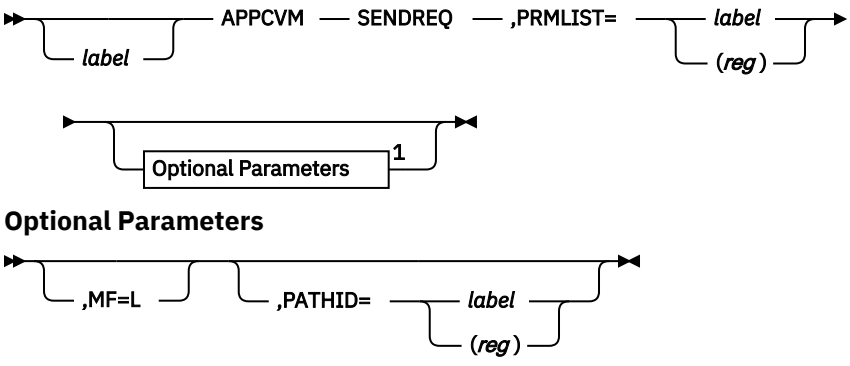
- Has no function outstanding on its half of the path
- Is in the Receive state on its half of the path
- Is enabled for message pending interrupts.

See Message Pending External Interrupt.

If you do not specify CODE=, the SENDERR code (IPCODE) that your communication partner gets depends on the state of the conversation and whether a logical record is being truncated.

For communication server programs, the code that your communication partner gets depends on the code specified on the CODE parameter of APPCVM SENDERR.

APPCVM SENDREQ (Send Request)



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the SENDREQ (Send Request) function to signal your communication partner that you would like to send data. The communication partner can ignore your request.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SENDREQ.

PATHID=
specifies the path ID on which to send the request.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

Input Parameter List: The APPCVM SENDREQ parameter list has the input format shown in the following figure.

Figure 50. APPCVM SENDREQ Input Parameter List

IPPATHID
contains the path ID on which the request to send is sent.

IPFLAGS1
contains the following input bit flag:

IPAPPCSN (X'02')
The APPC SEND function is issued.

IPSENDOP
contains the SEND option code:

IPREQSND (X'07')
indicates the request to send.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
Not possible	X	X	Not possible

SENDREQ always completes immediately.

CC=1
An error occurred. The output parameter list is the same as the input shown in SENDREQ Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	SENDREQ is an invalid function from the Connect state.
X'24'	36	SENDREQ is an invalid function from the Sever state.
X'26'	38	There is an invalid value in the IPSENDOP field.
X'33'	51	You must receive pending log data before issuing SENDREQ.
X'44'	68	SENDREQ is invalid from the Reset state.
X'45'	69	SENDREQ is invalid from the Defer_Receive state.
X'46'	70	SENDREQ is invalid from the Defer_Sever state.
X'4B'	75	SENDREQ is invalid from the Backout_Received state.
X'4C'	76	SENDREQ is invalid from the Backout_Required state.

CC=2
SENDREQ completed (also see SENDREQ Completion). The output parameter list when CC=2 is shown in the following figure.

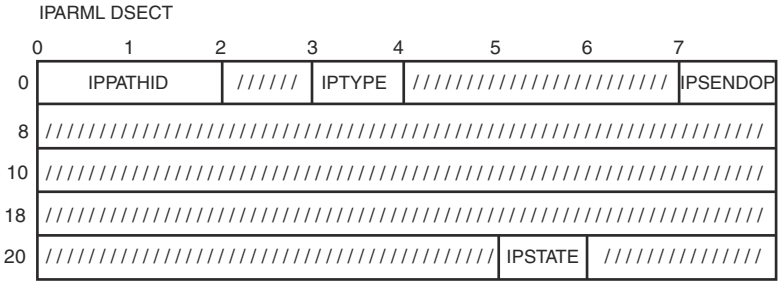


Figure 51. APPCVM SENDREQ Output Parameter List

IPPATHID
contains the path ID on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPSENDOP
contains the SEND option code:

IPREQSND (X'07')
The SENDREQ is being completed.

IPSTATE
contains one of the following values for the path's state:

IPSENDST (X'03')
The path is in the Send state.

IPRECVST (X'04')
The path is in the Receive state.

IPCONFRM (X'05')
The path is in the Confirm state.

Program Exceptions

The program exceptions for SENDREQ are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an external interrupt buffer has not been declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check results when APPCVM SENDREQ is issued from an improper state. (You also receive an IPRCODE when you issue a function from an improper state.) See the list of APPCVM SENDREQ return codes for all state check conditions.

No state change occurs.

Completion Conditions

Because the SENDREQ function completes immediately, you can issue another SEND (see note) or RECEIVE on the path when your virtual machine regains control.

Note: In this case, SEND refers to the set of APPC/VM send functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.

You can issue more than one SENDREQ. Your communication partner does not get additional SENDREQs until it receives an indication of any preceding SENDREQs. Those SENDREQs, sent before previous SENDREQs have been indicated to your partner, are lost. CP does not notify you when your communication partner actually gets the SENDREQ interrupt. You also do not receive an error message if you issue another SENDREQ before your partner receives notification of previous SENDREQs.

You can issue SENDREQ even when another function is pending on the path. If the pending function is a SENDCNF TYPE=SEVER, then your partner may not be informed of your SENDREQ.

If you issue a SEVER before your communication partner learns of your SENDREQ, your communication partner may not be informed of your SENDREQ.

What Happens to Your VM Communication Partner

If your communication partner is enabled for SENDREQ interrupts, it gets the SENDREQ interrupt shown in the following figure.

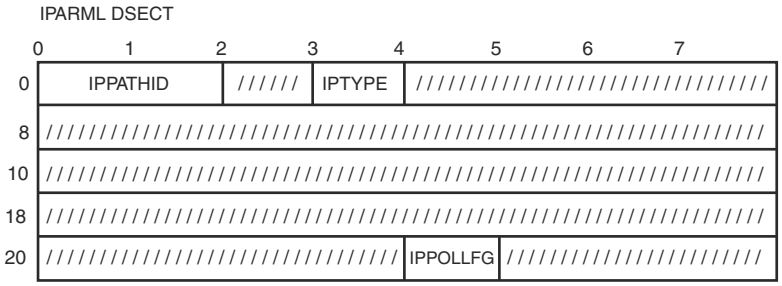


Figure 52. SENDREQ (Request-to-Send) Interrupt

IPPATHID
is the path ID on which you get the SENDREQ notice.

IPTYPE
is the interrupt type for a SENDREQ notification (IPTYPSRA, X'88').

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

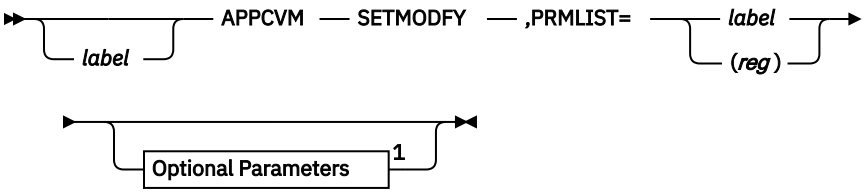
Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

CP does not queue more than one SENDREQ interrupt on a single path for the communication partner at one time. Thus, the number of SENDREQ interrupts reflected to your communication partner may be less than the number of SENDREQs issued.

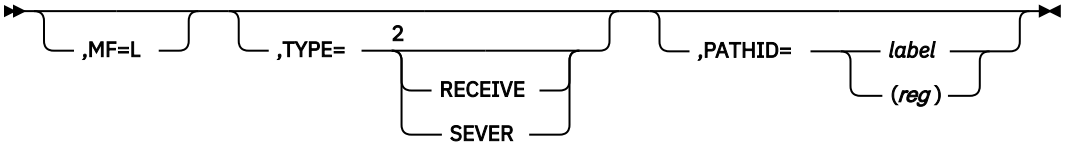
You cannot receive SENDREQ indicators with the RECEIVE function. They are only presented as an interrupt or with the DESCRIBE function.

If your connection to your VM partner is routed over a VTAM link, then the partner may not receive a SENDREQ interrupt for the conversation until it performs some other APPC/VM operation on the conversation. For more information, see Chapter 10, “APPC Mapped with APPC/VM,” on page 573.

APPCVM SETMODFY (Set Modify)



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

The SETMODFY function sets the state to Defer_Receive or Defer_Sever state, and sets the sync-point control modifier to Receive or Sever. The sync-point control modifier specifies the state to occur after a subsequent sync-point completes.

Note: If SETMODFY is not issued, a program is in the Send state when the sync-point completes.

Parameters

Required Parameters:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

TYPE=
specifies the sync-point control modifier, which indicates the state that a program wishes to be in at the end of the sync-point sequence:

RECEIVE
the program is in the Receive state when the sync-point sequence completes.

SEVER
the program is in the Sever state when the sync-point sequence completes.

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SETMODFY.

PATHID=
specifies the path ID on which the sync-point control modifier is being specified.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

Input Parameter List: The APPCVM SETMODFY parameter list has the input format shown in the following figure.

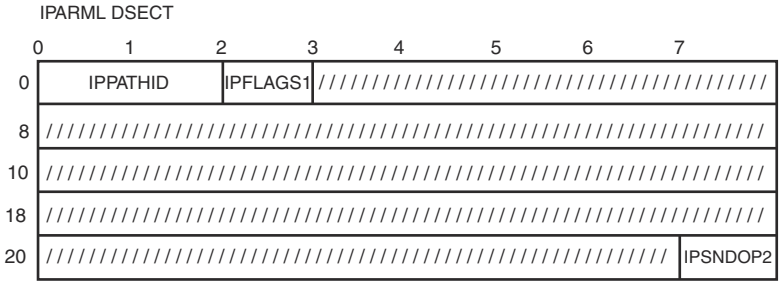


Figure 53. APPCVM SETMODFY Input Parameter List

IPPATHID
contains the path ID on which the sync-point control modifier is being specified.

IPFLAGS1
contains the following input bit flag:

IPAPPC (X'08')
specifies that this is an APPC function.

IPSNDOP2
contains one of the following SEND option codes:

IPTPRECV (X'04')
a Receive state was requested when the sync-point sequence completes.

IPTPSEVR (X'06')
a Sever state was requested when the sync-point sequence completes.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
Not possible	X	X	Not possible

SETMODFY always completes immediately.

CC=1
An error occurred before the SETMODFY was initiated. The output parameter list is the same as the input shown in SETMODFY Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	The specified path ID is not yet established.
X'03'	3	A function is pending on this path.
X'1E'	30	This is a non-APPC path.

Hex Code	Decimal Code	Why the Error Occurred
X'20'	32	SETMODFY is invalid from the Connect state.
X'22'	34	SETMODFY is invalid from the Receive state.
X'23'	35	SETMODFY is invalid from the Confirm state.
X'24'	36	SETMODFY is invalid from the Sever state.
X'25'	37	The connection was established with SYNCLVL=NONE or CONFIRM.
X'2C'	44	Before invoking SETMODFY, the program started but did not finish sending a logical record.
X'44'	68	SETMODFY is invalid from the Reset state.
X'45'	69	SETMODFY is invalid from the Defer_Receive state.
X'46'	70	SETMODFY is invalid from the Defer_Sever state.
X'47'	71	SETMODFY is invalid from the Prepare_Received state.
X'49'	73	SETMODFY is invalid from the Unsolicited_Request_Commit_Received state.
X'4B'	75	SETMODFY is invalid from the Backout_Received state.
X'4C'	76	SETMODFY is invalid from the Backout_Required state.
X'4E'	78	IPSNDOP2 contains an invalid value.

CC=2

SETMODFY completed with no error caused by the invoker. The output parameter list when CC=2 is shown in the following figure.

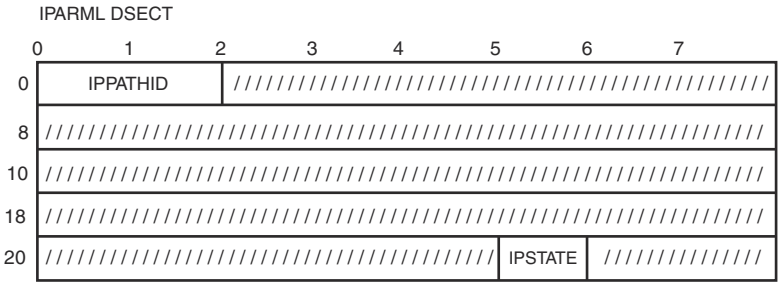


Figure 54. APPCVM SETMODFY Output Parameter List

IPPATHID
contains the path ID on which SETMODFY completed.

IPSTATE
contains the current state for this path, which may have one of the following values:

IPDEFRCV (X'07')
The path is in the Defer_Receive state.

IPDEFSEV (X'08')
The path is in the Defer_Sever state.

Program Exceptions

The program exceptions for SETMODFY are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.

Type	Description
Operation	Either an external interrupt buffer has not been declared, or the invoking virtual machine is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check occurs (see the IPRCODEs) if the path is not in the Send state.

The state change depends on the value of the TYPE= parameter:

- When TYPE=RECEIVE, the state changes to the Defer_Receive state.
- When TYPE=SEVER, the state changes to the Defer_Sever state.

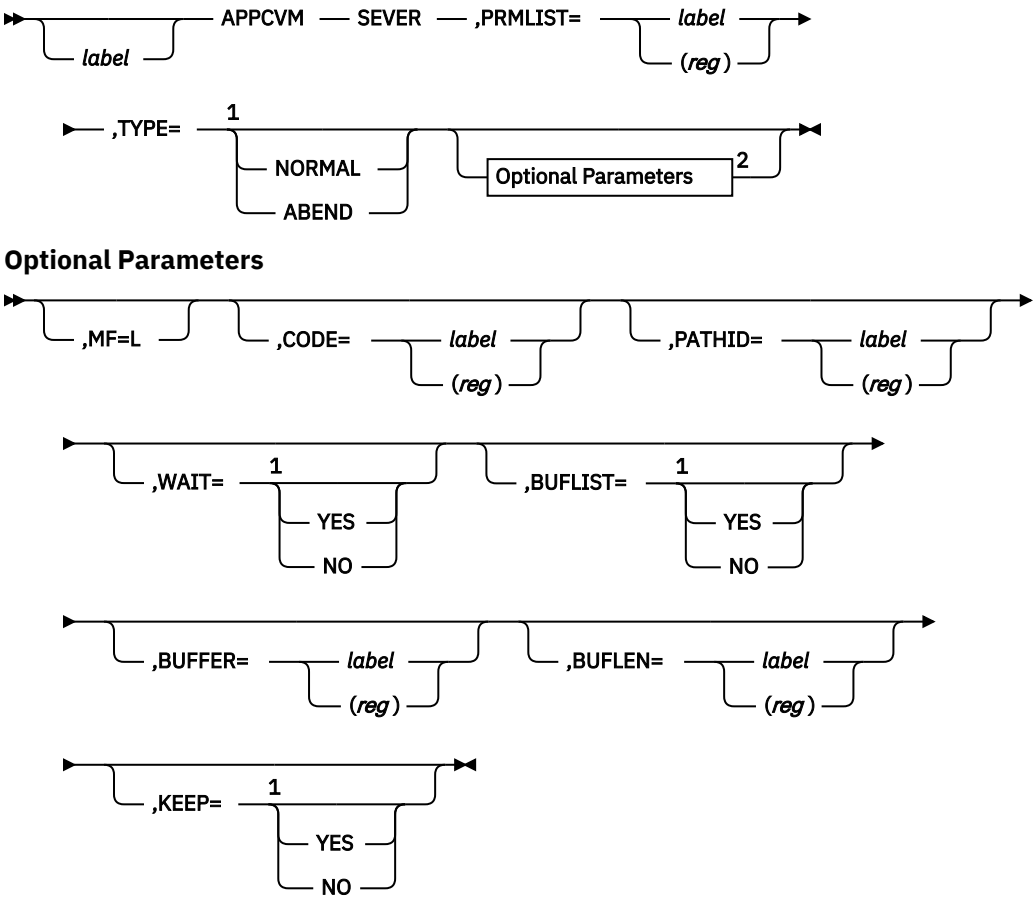
Completion Conditions

The SETMODFY function always completes immediately. This allows the program to issue another function on the same path as soon as the virtual machine regains control.

What Happens to Your VM Communication Partner

The communication partner is not affected by the SETMODFY function.

APPCVM SEVER



Optional Parameters

Notes:

[1] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

[2] Optional parameters can be entered in any order.

Purpose

Use the SEVER function to break a communication path with another virtual machine or your own virtual machine. After severing the connection with the other virtual machine, you cannot send or receive any other messages on that connection. Remember that your communication partner cannot receive any of the data that has not yet been copied out of your storage. However, your partner can receive log data that you send with APPCVM SEVER.

Parameters

Required Parameters:

PRMLIST=
specifies the address of the APPC/VM parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

TYPE=
indicates the type of SEVER performed.

NORMAL
severs the path normally. You can only do this if you are in the Send state (see note) and not in the middle of sending a logical record, or if you are in the Sever state.

Note: For a SYNCLVL=SYNCPT conversation, only communication servers can issue TYPE=NORMAL from the Send state.

ABEND
severs the path abnormally. APPC/VM invokes SEVER TYPE=ABEND from the Send, Receive, or Confirm state, even if there is a function that still has not completed on the specified path.

TYPE=NORMAL is not valid if BUFLIST, BUFFER, and BUFLEN are specified.

Optional Parameters:

CODE=
specifies a 2-byte sever code. CODE is only valid when you specify TYPE=ABEND. IBM defines all the codes; applications may not define error or sever codes for their own use.

label
is the relocatable label of the 2-byte sever code.

(*reg*)
is the register number that contains the sever code.

When CP issues a SEVER, or your communication partner issues an IUCV SEVER or RTRVBFR, CP determines the sever code to reflect. See What Happens to Your VM Communications Partner for more information on what this sever code could be. Also see Sever Codes That You Can Issue for information on what you can issue.

MF=L
generates the instructions necessary to initialize the APPC/VM parameter list as specified, but does not invoke the APPCVM SEVER.

PATHID=
specifies the path ID that is severed.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

WAIT=
lets you specify when control is returned to your virtual machine.

YES
returns control to your virtual machine when the SEVER is complete.

NO
returns control to your virtual machine when the SEVER is initiated.

BUFLIST=
specifies the type of buffer address to which the BUFFER parameter refers.

YES
refers to a list of addresses.

NO
refers to a single address.

This parameter is not valid if TYPE=NORMAL.

BUFFER=
specifies the address of the areas from which CP takes the log data.

label
is the relocatable label in storage where CP gets the log data to send.

(*reg*)
is the register number that contains the address of the storage area. This storage area is where CP gets the log data to send.

This parameter is not valid if TYPE=NORMAL.

BUFLEN=
specifies the length of the areas from which APPC/VM takes the log data to be sent. The minimum length is eight bytes; the maximum length is 600 bytes. This length is not related to the length of a logical record.

label
is the relocatable label of the fullword that contains the length.

(*reg*)
is the register number that contains the length.

This parameter is not valid if TYPE=NORMAL.

KEEP=
indicates whether the path ID may be reassigned by CP for another conversation immediately after the APPCVM SEVER.

YES
indicates that the path ID is not to be freed for reuse by CP for another conversation.

Note: This value is invalid when TYPE=NORMAL.

NO
indicates that the path ID is to be freed for reuse by CP for another conversation.

Input Parameter List: The APPCVM SEVER parameter list has the input format shown in the following figure.

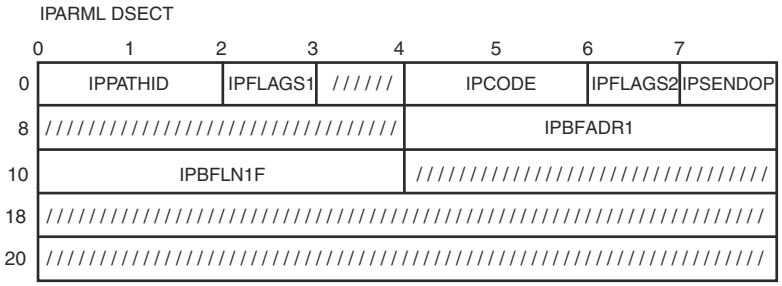


Figure 55. APPCVM SEVER Input Parameter List

IPPATHID
contains the path ID being severed.

IPFLAGS1
contains one of the following input bit flags:

IPBUFLIST (X'40')
You specified the buffer list option. This flag is ignored unless IPSENDOP=IPSABEND.

IPAPPC (X'08')
The APPC SEVER function was issued.

IPKEEP (X'10')
indicates that the path ID is not to be freed for reuse after the APPCVM SEVER completes.

Note: This flag is ignored unless IPSENDOP=IPSABEND.

IPCODE
contains the sever code. IPCODE is only valid when IPSENDOP=IPSABEND. See Sever Codes That You Can Issue.

IPFLAGS2
contains one of the following input bit flags:

IPWAIT (X'80')
You specified the wait option. This flag is ignored unless LOGDATA was specified.

IPCOMSRV (X'20')
Means that the SEVER is on behalf of another user. Only an authorized user (OPTION COMSRV in directory) can specify IPCOMSRV.

IPSENDOP
contains one of the following SEND option codes:

IPSNORM (X'08')
You requested that the path be severed normally.

IPSABEND (X'09')
You requested that the path be severed abnormally.

IPBFADR1
contains the address of the area from which APPC/VM takes the log data or the address of the address list or the length list. This parameter is ignored unless IPSENDOP=IPSABEND.

IPBFLN1F
contains the length of the log data being sent. If IPBFLN1F is 0, then no log data is being sent. Otherwise, log data is being sent. This parameter is ignored unless IPSENDOP=IPSABEND.

Usage Notes

Specifying Log Data: An application program can choose to set up log data and to send that log data using buffers specified on APPCVM SEVER. Log data conveys error information to an LU, where it is added to the system error log. This error information can be used in debugging and error recovery.

When using APPCVM SEVER to send log data to your communication partner, you use buffers. You can specify a single buffer using one address and one length or specify multiple buffers using a list of addresses and lengths.

When you specify a single buffer using one address and one length:

- BUFFER specifies the address
- BUFLEN specifies the length
- BUFLIST must be equal to NO.

When you specify multiple buffers with a list of addresses and lengths:

- BUFFER specifies the address of the list
- BUFLEN specifies the sum of the lengths of the buffers in the list
- BUFLIST must be equal to YES.

When specifying address lists (BUFLIST=YES), note the following:

1. The list must begin on a doubleword boundary.
2. Each list entry must be two fullwords; the first is the address of that portion of the list, and the second is the length.
3. The addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete.
4. APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified by BUFLEN.

Note: Log data is ignored on an incomplete path.

The log data you are sending in buffers must be coded into an error log general data stream (GDS) variable. An error log GDS variable has the following format, defined by SNA LU 6.2:

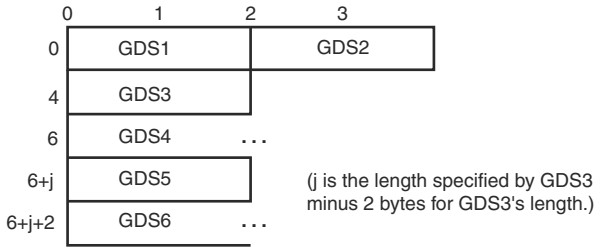


Figure 56. Error Log GDS Variable Format

GDS1
contains the length, in binary, of the error log GDS variable, including this length field. This must be in the range from 8 to 600 bytes.

GDS2
contains a GDS identifier for an error log variable, X'12E1'.

GDS3
contains the length, in binary, of the product set ID (GDS4) including this length field.

Note: The length field is always present; a value of 2 indicates no product set ID subvector follows.

GDS4
contains the product set ID subvector. This subvector, which is variable length, uniquely identifies the product.

GDS5
contains the length, in binary, of the log data including this length field.

Note: The length field is always present; a value of 2 indicates no log data follows.

GDS6
contains the log data.

For more information about the error log GDS Variable and product set ID, see *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*.

Sever Codes That You Can Issue: You can specify a sever code using the CODE= parameter on an APPCVM SEVER with TYPE=ABEND. The point at which you can specify a sever code depends on the state of your path. (However, you can issue an IUCV SEVER function on any APPC path at any time.)

The table below shows the APPC error condition associated with each sever code. Refer to *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2* for an explanation of what these APPC conditions mean and when to use these sever codes.

- If you are in the Connect state, you can only issue IUCV SEVER, which has no associated sever codes.
- After you ACCEPT the connection, and before you issue any other function on the path, you can issue APPCVM SEVER with any of the following APPC/VM sever codes:

APPC/VM Code	APPC Error Condition
X'0120'	CONVERSATION_TYPE_MISMATCH
X'0130'	SYNC_LEVEL_NOT_SUPPORTED_BY_PGM
X'0140'	TRANS_PGM_NOT_AVAIL_NO_RETRY
X'0141'	TRANS_PGM_NOT_AVAIL_RETRY
X'0142'	TPN_NOT_RECOGNIZED
X'0150'	PIP_NOT_SPECIFIED_CORRECTLY

APPC/VM Code	APPC Error Condition
X'0151'	PIP_NOT_ALLOWED
X'0160'	SECURITY_NOT_VALID
X'0210'	DEALLOCATE_ABEND_PROG
X'0220'	DEALLOCATE_ABEND_SVC
X'0230'	DEALLOCATE_ABEND_TIMER

- After the path is established (that is, CONNECT or ACCEPT is complete) you can issue APPCVM SEVER with the sever codes shown as follows:

APPC/VM Code	APPC Error Condition
X'0210'	DEALLOCATE_ABEND_PROG
X'0220'	DEALLOCATE_ABEND_SVC
X'0230'	DEALLOCATE_ABEND_TIMER

- Communication servers can sever with any defined SEVER code. See page “Currently-Defined APPC/VM Sever Codes” on page 399 for all currently-defined sever codes. The communication server is responsible for using only valid sever codes based on the conversation state according to the APPC architecture.

The sever type and code presented to your partner may not always be the sever type and code that you specified. For example, if your partner issues a SENDERR from the Receive state, a sever code of DEALLOCATE_ABEND_PROG is presented to your partner as DEALLOCATE_NORMAL in the completion data of your partner's SENDERR. See “APPCVM SENDERR (Send Error)” on page 490 for more information.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	X

CC=0
The APPCVM SEVER started successfully, but has not yet completed. If your virtual machine is enabled for function complete interrupts, one is presented to your virtual machine when SEVER completes. When you get the function complete interrupt, check the IPAUDIT field for error information. The function complete interrupt has the same format as the SEVER output parameter list (see CC=2, 3).

When control is returned to your virtual machine with CC=0, the parameter list may have been altered.

Note: CC=0 is not possible when WAIT=YES.

CC=1
An error occurred. The parameter list format is the same as the input shown in SEVER Input Parameter List, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	01	You specified a path ID that is not yet established.
X'03'	03	A function is pending on this path.
X'0A'	10	Your buffer length is negative.

Hex Code	Decimal Code	Why the Error Occurred
X'17'	23	A length specified in your SEND buffer list is negative.
X'18'	24	The total length that you specified is not the total of the lengths in your SEND buffer list.
X'1A'	26	The buffer list address is not on a doubleword boundary.
X'1D'	29	You are not authorized to act for another user.
X'1E'	30	You specified an APPC/VM function on a non-APPC path.
X'20'	32	APPCVM SEVER is an invalid function from the Connect state.
X'22'	34	APPCVM SEVER TYPE=NORMAL is an invalid function from the Receive state.
X'23'	35	APPCVM SEVER TYPE=NORMAL is an invalid function from the Confirm state.
X'24'	36	APPCVM SEVER TYPE=ABEND is an invalid function from the Sever state.
X'26'	38	There is an invalid value in the IPSENDOP field.
X'2C'	44	Before invoking APPCVM SEVER TYPE=NORMAL, you started but did not finish, sending a logical record.
X'2E'	46	You specified an invalid sever code.
X'37'	55	Your buffer length is either less than 8 or greater than 600.
X'38'	56	WAIT=YES was specified on a function issued to this same virtual machine.
X'44'	68	APPCVM SEVER is an invalid function from the Reset state.
X'45'	69	APPCVM SEVER TYPE=NORMAL is invalid from the Defer_Receive state.
X'46'	70	APPCVM SEVER TYPE=NORMAL is invalid from the Defer_Sever state.
X'47'	71	APPCVM SEVER TYPE=NORMAL is invalid from the Prepare_Received state.
X'49'	73	APPCVM SEVER TYPE=NORMAL is invalid from the Unsolicited_Request_Commit_Received state.
X'4B'	75	APPCVM SEVER TYPE=NORMAL is invalid from the Backout_Received state.
X'4C'	76	APPCVM SEVER TYPE=NORMAL is invalid from the Backout_Required state.
X'4F'	79	APPCVM SEVER TYPE=NORMAL is invalid on a SYNCLVL=SYNCPT conversation when issued by: • A communication server from a state other than the Sever or the Send state • A noncommunication server from a state other than the Sever state.
X'57'	87	An APPCVM SEVER with KEEP=YES is invalid after a previous APPCVM SEVER that specified log data.

CC=2 or CC=3

The SEVER completed (also see SEVER Completion). When CC=2, the function completed with no errors; when CC=3, there is error information in IPAUDIT.

Note: When you specify WAIT=NO, CC=3 is not possible.

The output parameter list when CC=2 or CC=3 is as shown in the following figure.

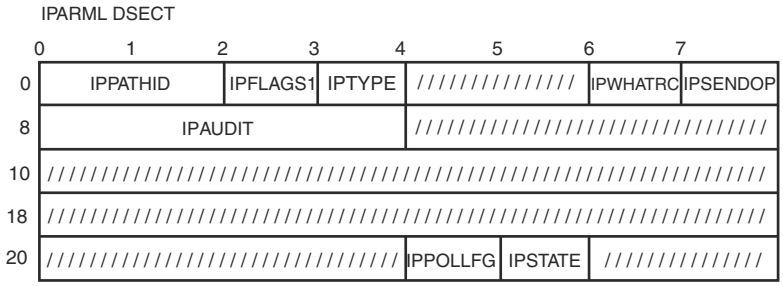


Figure 57. APPCVM SEVER Output Parameter List (Function Complete Interrupt)

IPPATHID
contains the path ID on which the function is complete.

IPFLAGS1
may contain one or more of the following output bit flags:

X'20'
This value is reserved for IBM use only.

X'02'
This value is reserved for IBM use only.

X'01'
This value is reserved for IBM use only.

IPTYPE
Contains the function complete interrupt code (IPTYPFCA, X'87').

IPWHATRC
Contains the what-received code.

IPCOMP (X'00')
SEVER completed normally.

IPSENDOP
contains one of the following SEND option codes:

IPSNORM (X'08')
SEVER TYPE=NORMAL has completed.

IPSABEND (X'09')
SEVER TYPE=ABEND has completed.

IPAUDIT
has three fields that may contain error information.

Note: In the following descriptions:

- *Send area* refers to either a send buffer specified directly on an APPCVM SEVER, BUFFER= or a send buffer that is part of a buffer list. This area is used for sending log data.
- *Receive area* refers to either a receive buffer specified directly on an APPCVM RECEIVE, BUFFER= or a receive buffer that is part of a buffer list. This area is used for receiving log data.

(See Specifying Log Data for more information on specifying buffers and buffer lists.)

IPAUDIT1 (first byte of IPAUDIT)
may contain one of the following bit flags:

IPADSNPX (X'40')
A protection exception occurred on your send area for log data.

IPADSNAX (X'20')
An addressing exception occurred on your send area for log data.

IPAUDIT2 (second byte of IPAUDIT)
may contain one of the following bit flags:

IPADRCPX (X'80')
A protection exception occurred on your communication partner's receive area for log data.

IPADRCAX (X'40')
An addressing exception occurred on your communication partner's receive area for log data.

IPADRLST (X'04')
Your communication partner had an invalid receive buffer list.

IPAUDIT3 (third byte of IPAUDIT)
may contain one of the following bit flags:

IPADBLEN (X'80')
A bad length is in your send buffer list.

IPADBTOT (X'20')
Your total send buffer length is invalid.

IPADIINV (X'04')
There is an invalid logical record length in your data stream.

IPASYRC (fourth byte of IPAUDIT)
Reserved.

Even when an error is reported in the IPAUDIT field, the path is severed.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
contains the current state for this path.

IPRESET (X'01')
Indicates that the path is in the Reset state.

IPSENDST (X'03')
Indicates that the path is in the Send state.

IPRECVST (X'04')
Indicates that the path is in the Receive state.

Program Exceptions

The program exceptions for SEVER are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an external interrupt buffer was not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check occurs (IPRCODE=X'44') if the path is in the Reset state. (When you issue a function from an improper state, you receive an IPRCODE.) Other state checks depend on whether TYPE=NORMAL or TYPE=ABEND. See the list of APPCVM SEVER return codes for all state check conditions.

When your virtual machine regains control after successfully completing the SEVER (CC=2 or CC=3), you enter the Reset state. However, if APPCVM SEVER TYPE=ABEND was issued by a noncommunication server while a sync-point was in progress, the state changes to the *saved_state* value (either the Send or the Receive state).

The state of a path on a SYNCLVL=SYNCPT conversation is **saved** when the conversation is initialized and when each SYNCPT completes. This is so the conversation can back out to the saved state when a backout occurs.

No state change occurs when CC=1.

Completion Conditions

After APPCVM SEVER completes, you cannot issue any other functions on that path. The timing of an APPCVM SEVER completion depends on whether log data was specified.

If you do not specify log data on the APPCVM SEVER, it always completes immediately. If you **do** specify log data, the APPCVM SEVER function completes when all of the log data is copied out of your buffer. An APPCVM SEVER with log data also completes when your partner issues a SEVER.

If your communication partner specified that it does not accept log data, the log data is automatically considered copied out of your buffer. If log data is specified and control is returned to your virtual machine with CC=0, this indicates that the function started successfully. The only functions that can be issued on the path are:

- IUCV SEVER, KEEP=NO
- APPCVM SEVER, KEEP=NO, TYPE=ABEND, with no log data.

If you issue either of these two functions after issuing an APPCVM SEVER with log data, your communication partner cannot receive any log data that has not yet been copied out of your storage.

You cannot issue APPCVM SEVER TYPE=NORMAL if there is another function outstanding on the path. You can, however, issue APPCVM SEVER TYPE=ABEND without log data if there is an outstanding function on a path. (However, if the pending function is a syncpoint function, IPCODEs of X'210', X'220', and X'230' may not be specified.) In this case, CP may not present the outstanding function to your communication partner. For example, if you issue the following sequence of functions, your communication partner is notified of the SEVER, but not the SENDERR:

1. SENDERR
2. SEVER TYPE=ABEND (before your partner receives the SENDERR).

Also, in the following example sequence of functions, your communication partner cannot receive more than the amount of data specified in the RECEIVE.

1. You issue a SENDDATA BUFLEN=200.
2. Your communication partner issues a RECEIVE BUFLEN=100.
3. You issue a SEVER TYPE=ABEND.

CP notifies your communication partner of the APPCVM SEVER with a sever interrupt. In addition, CP notifies your partner the next time your partner issues a function on which CP can report the APPCVM SEVER.

If KEEP=YES is not specified for APPCVM SEVER, the path ID is no longer valid when the SEVER completes. If another function is then issued for that path ID, the function completes with CC=1 and an IPRCODE of X'01' (specified path ID not established).

What Happens to Your VM Communication Partner

Any time after you and your partner have established a path (after the CONNECT/ACCEPT sequence is complete), you can issue an APPCVM SEVER with TYPE=NORMAL or TYPE=ABEND.

Note: Before a path is established, you cannot issue an APPCVM SEVER, only an IUCV SEVER, unless your program is a communication server. Refer to "IUCV SEVER" on page 556.

When you issue an APPCVM SEVER, your partner gets a sever interrupt (assuming it is enabled for sever interrupts). In addition, if your partner has a RECEIVE, SENDDATA, SENDCNF, or SENDERR outstanding on its half of the path, the function completes. Otherwise, if your partner issues one of those four functions after you have issued the APPCVM SEVER, their function completes immediately with indication of IPWHATRC=IPSNORM or IPWHATRC=IPSABEND.

A sever interrupt does not change your partner's path to the Sever state (the partner could have been in the Sever state already). Your partner only enters the Sever state after it issues a function that completes with IPWHATRC=IPSNORM or IPWHATRC=IPSABEND.

When you issue an APPCVM SEVER with log data, the way your partner receives it depends on what state it was in when it got the Sever interrupt:

- If your partner was already in the Sever state when it got the Sever interrupt, it must next issue an APPCVM RECEIVE to get the log data.
- If your partner was not in the Sever state when it got the Sever interrupt, it finds out about the log data when it issues a function that completes with IPWHATRC=IPSABEND. It must then issue an APPCVM RECEIVE to get the log data.

If your connection to your VM partner is routed over a VTAM link, then the partner may not receive a SEVER interrupt for the conversation until it performs some other APPC/VM operation on the conversation. For more information, see Chapter 10, "APPC Mapped with APPC/VM," on page 573.

SEVER External Interrupt: A SEVER external interrupt on an APPC/VM path can result from an APPCVM SEVER or an IUCV SEVER. In either case, it has the format shown in the following figure.

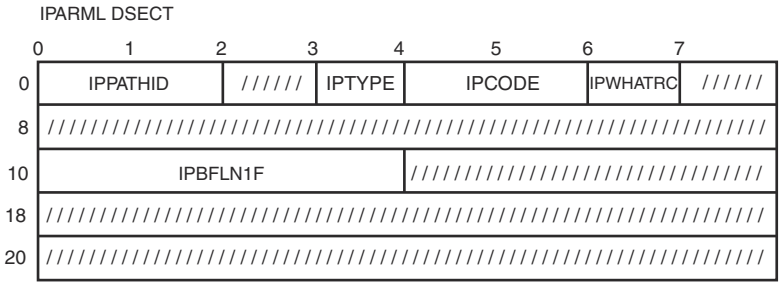


Figure 58. SEVER External Interrupt

IPPATHID
contains the path ID being severed.

IPTYPE
contains the interrupt type for SEVER (IPTYPSVA, X'83').

IPCODE
contains the sever code from the partner's SEVER. See "APPC/VM Sever, Error, and Sense Codes That You Can Get" on page 399 for a description of the sever codes.

IPWHATRC
contains the what-received code:

IPSNORM (X'08')
Your partner issued a SEVER TYPE=NORMAL.

IPSABEND (X'09')
Your partner issued SEVER TYPE=ABEND.

IPBFLN1F
contains the length of the pending log data.

For an IUCV SEVER, CP ignores the user data field, issues an APPCVM SEVER TYPE=ABEND to your communication partner, and then reflects a sever code. For an explanation of possible sever codes, see What Happens to Your VM Communications Partner.

Chapter 8. IUCV Macro Functions for Use in APPC/VM

This chapter describes, in detail, each IUCV macro function that can be used in APPC/VM. These are:

- IUCV ACCEPT
- IUCV CONNECT
- IUCV DCLBFR (Declare Buffer)
- IUCV IPOLL (Interrupt Poll)
- IUCV QUERY
- IUCV RTRVBFR (Retrieve Buffer)
- IUCV SETCMASK (Set Control Mask)
- IUCV SETMASK
- IUCV SEVER
- IUCV TESTCMPL (Test Completion)
- IUCV TESTMSG (Test Message)

If you are unfamiliar with reading syntax diagrams, see "Syntax, Message, and Response Conventions" on page xxxv.

The set of IUCV macro functions that can be used with APPC/VM is split into two sections. Those that:

- Can be used in CMS without special considerations
- Should be avoided in CMS.

Shared Functions That Can Be Used in CMS

The following table summarizes each IUCV macro function that can be used in CMS, then points to the section that describes each function in detail.

Function	Description	Page
ACCEPT	Accepts the connection from a requesting program to complete a path.	"IUCV ACCEPT" on page 524
CONNECT	Establishes and reserves a path for resource manager programs to communicate with *IDENT.	"IUCV CONNECT" on page 529
QUERY	Gets information about the external interrupt buffer and finds out how many paths can be established.	"IUCV QUERY" on page 543
SEVER	Ends communications with another program when APPCVM SEVER is not appropriate.	"IUCV SEVER" on page 556

For information on CMS in a virtual MP environment, see "Virtual MP Considerations for APPC/VM Applications" on page 398.

Shared Functions That Should Be Avoided in CMS

Other functions are also shared for both APPC/VM and IUCV. These shared functions should not be used in a CMS environment because they could affect other programs in the same virtual machine; however, they can be used safely in a non-CMS environment.

© Copyright IBM Corp. 1991, 2020

Each of these functions is briefly described in the following list, then described in detail in the remainder of this chapter.

The following table summarizes each IUCV macro function that should be avoided in a CMS environment, then points to the section that describes each function in detail.

Function	Description	Page
DCLBFR	Declares an interrupt buffer. (This buffer can be used for both APPC/VM and IUCV interrupts.)	"IUCV DCLBFR (Declare Buffer)" on page 533
DESCRIBE	Gives the following information: • The next message pending on non-APPC paths • The next message pending on an APPC path that is in the Receive state • A SENDREQ on an APPC path.	"IUCV DESCRIBE" on page 538
IPOLL	Determines if any messages or replies are pending.	"IUCV IPOLL (Interrupt Poll)" on page 540
RTRVBFR	Releases an interrupt buffer. (This buffer can be used for both APPC/VM and IUCV interrupts.)	"IUCV RTRVBFR (Retrieve Buffer)" on page 548
SETMASK	Disables and enables APPC and non-APPC interrupts.	"IUCV SETCMASK (Set Control Mask)" on page 550
SETCMASK	Disables and enables APPC and non-APPC control interrupts.	"IUCV SETMASK" on page 553
TESTCMPL	Determines the next APPC or non-APPC function that has completed.	"IUCV TESTCMPL (Test Completion)" on page 562
TESTMSG	Waits for the following: • A message pending or message complete interrupt on non-APPC paths • A message pending interrupt on an APPC path that is in the Receive state • A request-to-send interrupt on an APPC path • A function complete interrupt on an APPC path.	"IUCV TESTMSG (Test Message)" on page 566

For more information on how to use the IUCV macro functions when writing programs, see *z/VM: CMS Macros and Functions Reference*.

Condition Codes and Return Codes for IUCV Macro Functions

This section summarizes the condition codes and return codes that IUCV macro functions generate. See the individual IUCV function details for specific condition code and return code information.

Condition Codes

The condition code (CC) is stored in the program status word (PSW). There are four possible values for condition codes: 0, 1, 2, and 3. Here is the general meaning for each CC value:

Code
Meaning

CC=0
The function successfully completed with no errors.

CC=1
An error occurred when the function was initiated. In this case, the error code is stored in the IPRCODE field of the output parameter list. (See a description of IPRCODE below.)

CC=2
The function completed immediately with no errors. In most cases, the requested function was not performed. However, for IUCV ACCEPT on an APPC path, CC=2 indicates a successful completion.

CC=3
The function completed, but an error was detected.

Note: CC values for the IUCV TESTMSG have different meanings than shown here.

Return Codes

Return codes are stored in IPRCODE, 1-byte field in the output parameter list. IPRCODE reports error conditions that CP detects when the function is initiated. A nonzero value is placed into IPRCODE when CC=1. There is no corrective action for this type of error. You should sever the path when you get a nonzero value in this field.

Note: In addition, the IUCV TESTCMPL output parameter list may define a field called IPAUDIT. IPAUDIT reports error conditions that CP detects between the time that the function is initiated and the time it completes. Like IPRCODE, there is no corrective action for this type of error. You should sever the path when you get a nonzero IPAUDIT.

Using the Online HELP Facility for IUCV Macro Functions

You can receive online information about the IUCV macro functions (used with APPC/VM) by using the z/VM HELP Facility. For example, to display a menu of the IUCV macro functions, enter:

```
help iucv menu
```

To display information about a specific IUCV macro function (ACCEPT in this example), enter:

```
help iucv accept
```

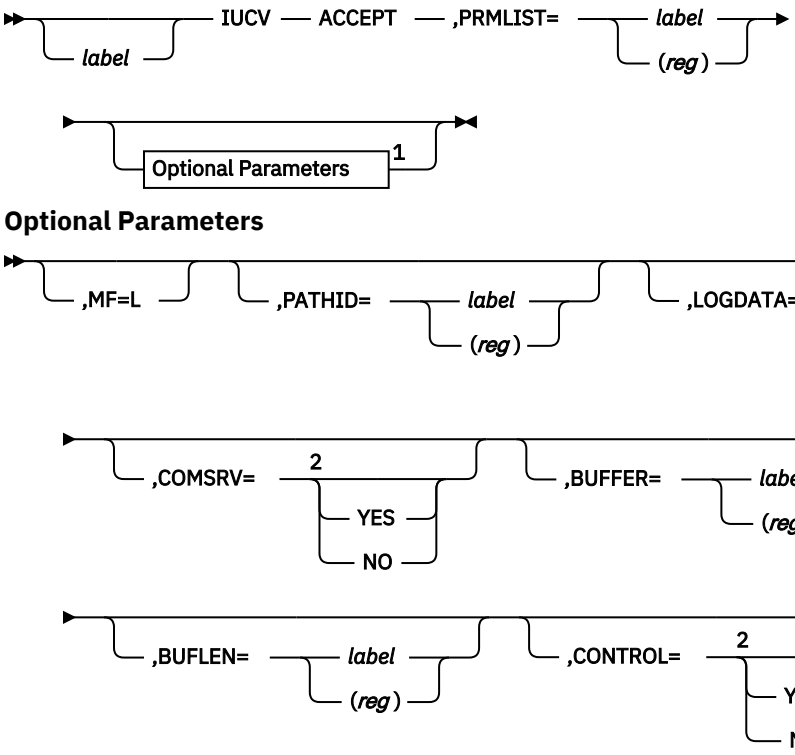
For more information about using the HELP Facility, see *z/VM: CMS User's Guide*. To display the main HELP Task Menu, enter:

```
help
```

For more information about the HELP command, see *z/VM: CMS Commands and Utilities Reference* or enter:

```
help cms help
```

IUCV ACCEPT



Notes:

[1] Optional parameters can be entered in any order.
[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

Use the ACCEPT function of the IUCV macro to accept a connection from another virtual machine or from your own virtual machine. You should use ACCEPT only after your virtual machine gets a connection pending external interrupt.

Note: The ACCEPT function is not part of the APPC architecture and is unique to VM.

In a CMS environment, you should use IUCV ACCEPT to set up an ACCEPT parameter list, then call CMSIUCV ACCEPT to invoke the function. For more information, see *z/VM: CMS Application Development Guide for Assembler*.

Parameters

Required Parameter

PRMLIST=
specifies the address of the IUCV ACCEPT parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters

MF=L
generates the instructions necessary to initialize the IUCV parameter list as specified, but does not invoke the IUCV ACCEPT.

PATHID=
identifies the path on which to accept the connection. If you specify the path ID, it must be the same value presented in the connection pending interrupt; if you do not specify the path ID, it defaults to the value contained in the first two-bytes of your parameter list.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

LOGDATA=
indicates whether the accepting program is willing to receive log data on the path being established.

YES
indicates that the accepting program receives log data.

NO
indicates that the accepting program does not receive log data. CP discards any log data sent to this program.

Communication Server Parameters

COMSRV=
indicates whether the connection is being accepted for another user.

YES
indicates that the connection is being accepted for another virtual machine. Only communication server virtual machines (OPTION COMSRV specified in the directory) can specify this parameter.

NO
indicates that the connection is not being accepted for another virtual machine.

BUFFER=
specifies the address of the connection complete extended data. (See Connection Complete Extended Data for more information.) Only communication servers can specify this parameter. If a noncommunication server specifies this parameter, it is ignored. There is not an error indication, and CP builds the extended data.

label
is the relocatable label of connection complete extended data address.

(*reg*)
is the register number that contains the address of the connection complete extended data.

BUFLEN=
specifies the length of the connection complete extended data. (See Connection Complete Extended Data for more information.) Only communication servers can specify this parameter. If a noncommunication server specifies this parameter, it is ignored and there is no error indication.

label
is the relocatable label of the fullword storage location containing the length.

(*reg*)
is the register number that contains the length. If BUFLEN is 0, BUFFER and BUFLEN are considered not specified; if BUFLEN is nonzero, length must be 80 (X'50') bytes, the current length of the connection complete extended data.

CONTROL=
lets you specify whether a control path is being established. Control paths allow interrupt information for your half of the path to be placed in the control buffer.

YES
sends the APPC/VM interrupt information on this path to the control buffer.

Note: Do not specify CONTROL=YES in application programs running in CMS; CMS uses control paths.

NO
sends APPC/VM interrupt information on this path the to application buffer.

Input Parameter List: The IUCV ACCEPT parameter list has the input format shown in the following figure when accepting a connection on an APPC path:

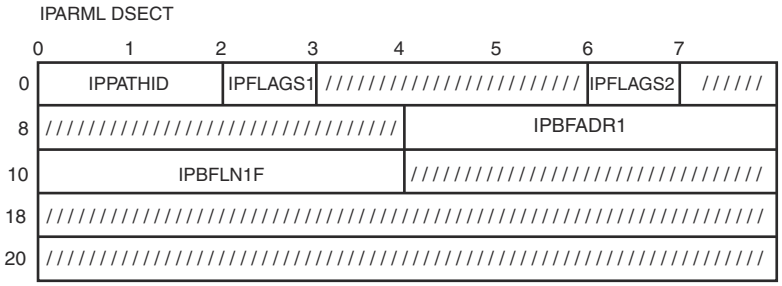


Figure 59. IUCV ACCEPT Input Parameter List

IPPATHID
is the path ID on which to accept the connection.

IPFLAGS1
contains the following input bit flag:

IPCNTRL (X'04')
a control path is being established.

X'80'
this value is reserved for IBM use only.

IPFLAGS2
contains one of the following input bit flags:

IPCOMSRV (X'20')
the connection is being accepted for another user.

IPLOGDOK (X'04')
the accepting program is specifying LOGDATA=YES.

IPBFADR1
specifies the address of the connection complete extended data.

IPBFLN1F
specifies the length of the connection complete extended data.

Usage Notes

Considerations for SNA Communication Servers: An SNA communication server can use the BUFFER and BUFLEN parameters to send connection-complete extended data to the source (connecting) program. The connection-complete extended data, which is sent to the source program along with connection-complete data, contains the session ID, the conversation correlator, the local and remote network fully qualified LU names, and the access user ID. CP passes the connection-complete extended data information unchanged from the SNA communication server to the source program.

It is the communication server's responsibility to ensure that the conversation correlator provided matches the conversation correlator received in the FMH5.

If an I/O error is encountered while reading in the connection-complete extended data from BUFFER, CP builds the connection-complete extended data from the allocate data as it would for an ACCEPT by a noncommunication server.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
Not Possible	X	X	Not Possible

CC=1
an error occurred. The output parameter list is the same as the ACCEPT input parameter list, except one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	IUCV ACCEPT cannot be issued on an APPC path until the connection pending interrupt is presented. You specified a path ID that is not yet established, or you tried issuing IUCV ACCEPT before getting a connection pending interrupt. You can also get this code if you issue a second IUCV ACCEPT on a path.
X'0A'	10	Invalid length for the connection complete extended data buffer
X'14'	20	Connection cannot be completed—originator has severed the path. **Note:** If your path gets severed, you must still issue IUCV SEVER to clean up your side of the path.
X'1D'	29	You are not authorized to act for another user.
X'44'	68	IUCV ACCEPT is invalid from the Reset state.
X'56'	86	You must have a control buffer declared to accept a SYNCLVL=SYNCPT connection.

CC=2
ACCEPT is complete. The format of the output parameter list is as shown in the following figure.

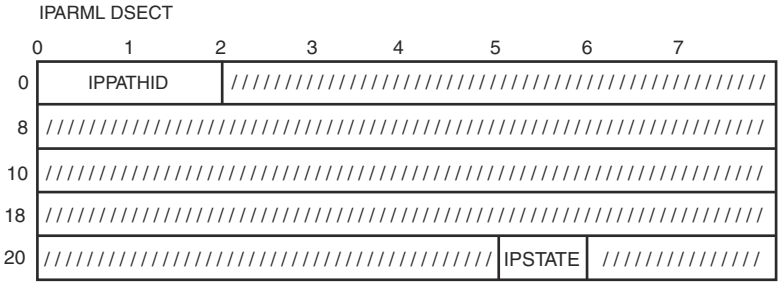


Figure 60. IUCV ACCEPT Output Parameter List

IPPATHID
is the path ID on which the connection was accepted.

IPSTATE
contains the current state for this path, which may have one of the following values:

IPRECVST (X'04')
the path is in the Receive state.

IPBKREQ (X'0E')
the path is in the Backout_Required state.

Program Exceptions

The program exceptions for ACCEPT are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine, the CCED buffer is outside the virtual machine, or the CCED buffer address is 0.
Operation	Either an external interrupt buffer has not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

If the ACCEPT completes successfully, your virtual machine enters the Receive state.

If the connection cannot be completed because the path was severed (IPRCODE 20), your virtual machine remains in the Connect state.

When IUCV ACCEPT completes for a conversation established with SYNCLVL=SYNCPT, the current state (Receive) is saved as the initial state value.

Completion Conditions

Because you can issue ACCEPT only when there is a connection pending, the ACCEPT function completes immediately. If you have not yet received the allocation data pending for this path, that data is purged when ACCEPT completes.

If the connection is for a private resource, the connection pending interrupt is presented in the control buffer; after the ACCEPT is issued, all subsequent interrupts are placed in the application buffer.

When IUCV ACCEPT is issued before the PIP variable has been completely received, CP purges any remaining data for the PIP variable that has not been received.

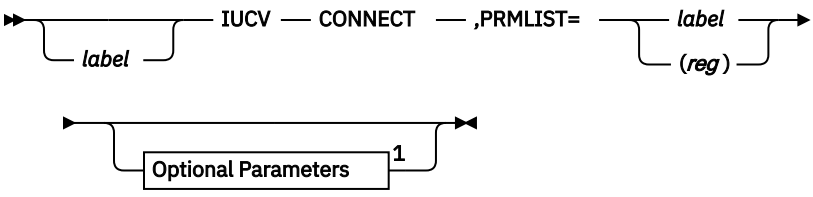
What Happens to Your VM Communication Partner

When you issue the ACCEPT function, the receiver of the connection-complete indication depends on the system your communication partner (the virtual machine that issued the original CONNECT) is on. If it is:

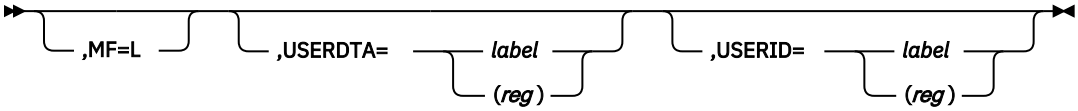
- The same system, your partner gets the indication.
- A different system, the intermediate communication server (for example, TSAF or AVS) gets it.

The ACCEPT function can be issued by an intermediate communication server that is not the target of the CONNECT. In this case, accepting a connection does not necessarily mean that the APPC/VM path has been completed between two communicating programs. See “APPCVM CONNECT” on page 412 for more details on the connection-complete indication.

IUCV CONNECT



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the CONNECT function of the IUCV macro to establish a path to the Identify system service (*IDENT). A program in a virtual machine with proper directory authorization can manage or revoke a resource by establishing a path to *IDENT.

Although the CONNECT may complete successfully, you are not able to use the path until you receive a connection-complete external interrupt (meaning that *IDENT has accepted your connection) for this path.

If you get an connection-severed external interrupt for this path (meaning that *IDENT has severed your connection), you may not use the path. For more information on *IDENT, see Chapter 16, “Identify System Service (*IDENT),” on page 731.

Notes:

1. The IUCV CONNECT function has more options when being used in an IUCV environment. However, only the IUCV CONNECT parameters shown here should be used when connecting to *IDENT.
2. If an external security manager is installed on your system, you may not be authorized to use this function. For additional information, contact your security administrator.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the IUCV parameter list. The IUCV instruction is generated to reference the address specified.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
lets you build an IUCV CONNECT parameter list without initializing any registers or invoking the connection. After using MF=L, you can use CMSIUCV CONNECT to actually issue the connect request to *IDENT. (For more information, see *z/VM: CMS Application Development Guide for Assembler*.)

USERDTA=
specifies the data area containing the 16 bytes of user data that you are supplying to *IDENT. The user data is presented to *IDENT as part of the connection pending external interrupt.

label
is the relocatable label of the user data.

(*reg*)
is the register number that contains the address of the user data.

On this IUCV CONNECT, the user data field must have the format shown in the following figure.

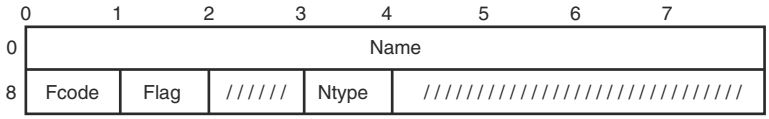


Figure 61. User Data Field for CONNECT

Name
contains the name of the resource or gateway that you are managing or revoking. The first byte of this name must be alphanumeric; IBM reserves names beginning with the remaining characters for its own use. In addition, this resource/gateway name cannot be blanks (X'40's), nulls (X'00's), ANY, ALLOW, or SYSTEM.

Fcode
is the function code. FCODE=1 indicates a request to identify (manage) a resource or gateway. FCODE=2 indicates a request to revoke a resource or gateway.

Flag
is a flag byte.

For **manage** requests (FCODE=1):

Bit 0 on
The resource is accessible from outside the local system:

- If bit 0 is on and bit 2 is off, this is a global resource, unique in the TSAF or CS collection, and accessible throughout the TSAF or CS collection and by AVS.
- If bit 0 is on and bit 2 is on, this is a system resource, accessible throughout the TSAF or CS collection through the system gateway and by AVS.

This bit must be on for a gateway.

Bit 0 off
The resource is accessible only from the local system (a local resource).

Bit 1 on
The resource manager program accepts connections with SECURITY(NONE).

Bit 1 off
The resource manager program does not accept connections with SECURITY(NONE).

Bit 2 on
The resource is a system resource. If this bit is on, bit 0 must also be on.

Bit 2 off
This resource is not a system resource.

For **revoke** requests (FCODE=2):

Bit 0 on
CP revokes the global or system resource or gateway, known to the TSAF or CS collection. It must also be on for revoking a system resource (see bit 2).

Bit 0 off
CP revokes the local resource, known only to the local system.

Bit 2 on
This resource is a system resource. If this bit is on, bit 0 must also be on.

Bit 2 off
This resource is not a system resource.

Ntype
indicates the type for name. If Ntype is 0 indicates a resource ID; if Ntype is 1 indicates a gateway name.

USERID=
specifies *IDENT, the 8-character user ID for the Identify system service.

Input Parameter List: The IUCV CONNECT parameter list has the input format shown in the following figure.

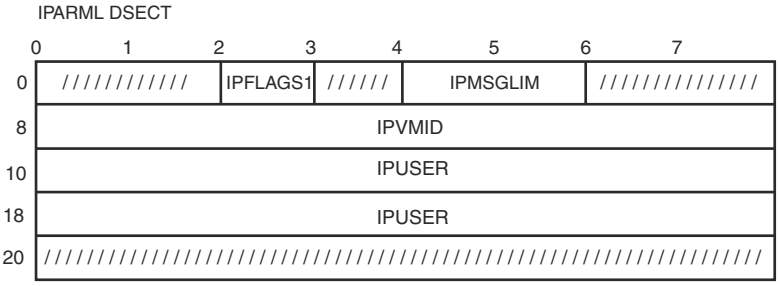


Figure 62. IUCV CONNECT Input Parameter List

IPFLAGS1
contains all zeros.

IPMSGLIM
contains all zeros.

IPVMID
contains *IDENT, the user ID of the system service to which you want to establish this path.

IPUSER
contains the user data that is reflected to the *IDENT system service.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	Not Possible	Not Possible

CC=0
the IUCV CONNECT completed successfully. The output parameter field, IPPATHID, identifies the path ID that IUCV assigns this new path.

CC=1
the IUCV CONNECT encountered an error, and stored one of the following return codes in IPRCODE (byte 3) of the output parameter list:

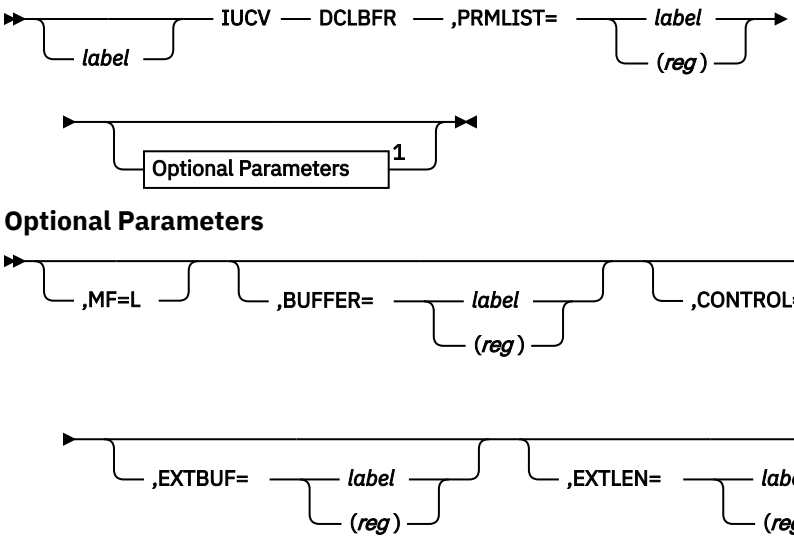
Hex Code	Decimal Code	Why the Error Occurred
X'00'	0	Normal return
X'0D'	13	Your virtual machine already has the maximum number of connections.
X'0F'	15	Your virtual machine is not authorized to connect to *IDENT.
X'10'	16	Your program specified an invalid IUCV system service name.

Program Exceptions

The program exceptions for IUCV CONNECT are:

Type	Description
Specification	The parameter list is not on a doubleword boundary.
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Addressing	The parameter list address that you specified is outside the virtual machine's storage.
Protection	The storage key of the specified parameter list address does not match the key of the user.

IUCV DCLBFR (Declare Buffer)



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

Use the DCLBFR (Declare Buffer) function before you use any other APPC/VM functions (except IUCV QUERY) to set the address of a buffer that APPC/VM and IUCV can use to store external interrupt data. After you receive an external interrupt, this buffer contains information about the message, reply, or control function that caused the interrupt.

When you issue DCLBFR, the virtual machine is enabled for all types of APPC/VM and IUCV external interrupts. Use the SETMASK and the SETCMASK functions to change these initial settings.

In addition, programs using control paths can set up an interrupt buffer extension.

Note: The IUCV interrupt mask in control register 0 is not affected by DCLBFR. See "IUCV SETMASK" on page 553 for more details. If your program is running in a CMS environment, you can use HNDIUCV SET to do the same job as DCLBFR.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Parameters

Required Parameters:

PRMLIST=

specifies the address of the IUCV DCLBFR parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label

is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
expands the IUCV macro to generate the instructions necessary to initialize the parameter list as specified, but not to invoke the DCLBFR function.

BUFFER=
identifies the external interrupt buffer. When an external interrupt is sent to the virtual machine, APPC/VM stores information about the message or the control interrupt in this buffer.

label
is the relocatable label of the storage area used as the external interrupt buffer.

(*reg*)
is the register number that contains the address of the storage area used as the external interrupt buffer.

Note: You can reduce the overhead involved in reflecting APPC/VM external interrupts to the virtual machine if the buffer you declare is entirely within one page. You can further reduce overhead if the buffer is entirely within page 0 of the virtual machine.

CONTROL=
lets you declare an application buffer or a control buffer.

YES
declares a control buffer.

NO
declares an application buffer.

User applications running CMS and GCS should not use a control buffer and control paths. CMS and GCS declare control buffers during their initialization process. CMS and GCS do not allow applications that use the control buffer to establish paths.

However, applications **not** running in CMS or GCS can use control and application buffers. If you declare a control buffer with DCLBFR, you can establish control paths with CONNECT CONTROL=YES. When you specify CONTROL=YES, only you view the path as a control path. Your communications partner views it as an application path.

When an interrupt for a control path is presented to your virtual machine, it goes to the control buffer. When an interrupt for an application path comes in, it goes to the application buffer, and the path ID is stored in the control buffer. The rest of the control buffer contains zeros.

When a private resource manager connection pending interrupt is presented to a virtual machine, the interrupt is presented in the control buffer. If the conversation is accepted with CONTROL=YES, then all subsequent interrupts for the conversation are placed in the control buffer. If the conversation is accepted with CONTROL=NO, then all subsequent interrupts for the conversation are placed in the application buffer. For more information, see “IUCV ACCEPT” on page 524.

Restricted Parameters:

EXTBUF=
specifies the address of the interrupt buffer extension. This parameter should not be used by applications running in CMS or GCS. (See Interrupt Buffer Extension.)

label
is the relocatable label in storage that contains the interrupt buffer extension.

(*reg*)
is the register number that contains the address of the interrupt buffer extension.

Note: EXTBUF is not valid when CONTROL=NO.

EXTLEN=
specifies the length of the interrupt buffer extension. This length must have a value from 0 to 4096 (inclusive). This parameter should not be used by applications running in CMS or GCS. (See Interrupt Buffer Extension.)

You can use the IUCV QUERY function to determine the maximum length of data that can be returned in the external interrupt buffer.

label
Is the relocatable label of the fullword storage location containing the length.

(*reg*)
Is the register number that contains the length.

Note: EXTLEN is not valid when CONTROL=NO.

Input Parameter List: The IUCV DCLBFR parameter list has the input format shown in the following figure.

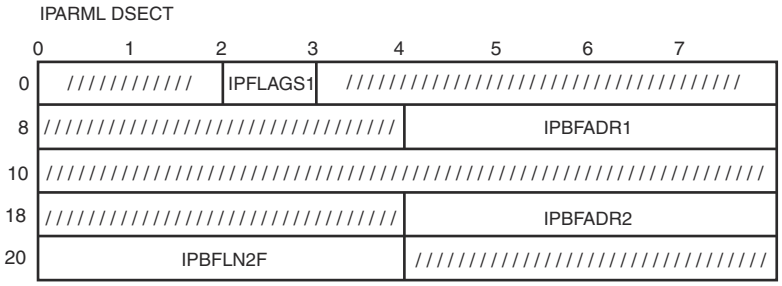


Figure 63. IUCV DCLBFR Input Parameter List

IPFLAGS1
may contain the following input bit flag:

IPCNTRL (X'04')
a control buffer is being established.

IPBFADR1
identifies the area where information about an APPC/VM or IUCV external interrupt is stored.

IPBFADR2
specifies the address of the interrupt buffer extension.

IPBFLN2F
specifies the length of the interrupt buffer extension.

Interrupt Buffer Extension: When an interrupt for which extended information is defined (that is, connection pending and connection complete interrupts on APPC/VM paths.) is reflected to the virtual machine, the extended information is placed in the interrupt buffer extension declared by the virtual machine. When a function (for which extended information is defined) completes with CC=2 instead of a completion interrupt, the extended information is still placed in the interrupt buffer extension (declared by the virtual machine), even though the other information is placed in the parameter list.

Note: If an I/O error is encountered while paging in the user's interrupt buffer extension, the extended information is not valid.

To ensure that you get all of the extended information, declare your extended interrupt buffer using the maximum allowable size. Use the IUCV QUERY,PRMLIST= function to determine the maximum size of the buffer. (See “IUCV QUERY” on page 543 for more information.)

If the length of the interrupt buffer extension is greater than the length of the extended information, the remaining area is undefined and may be set to zero by CP or may contain data left over from a previous interrupt. All extended interrupt data that can be variable in length contains length fields so that the application can determine what data is valid. The data is valid only until the next IUCV or APPC/VM interrupt is reflected to the virtual machine regardless of whether the next interrupt has extended data defined.

The interrupt buffer extension should not be used directly by applications running in CMS or GCS. CMS and GCS should use the interrupt buffer extension to obtain information necessary for coordinated resource recovery, such as the conversation correlator, session ID, and the network fully qualified logical unit (LU) names.

Note: Issuing IUCV RTRVBFR has no effect on an interrupt buffer extension.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	Not Possible	X

CC=0
DCLBFR completed with no errors. The output parameter list is the same as the input shown in the DCLBFR Input Parameter List.

CC=1
an error occurred before the DCLBFR was initiated. The output parameter list is the same as the input shown in the DCLBFR Input Parameter List, except that the return code is stored in IPRCODE (byte 3) You can get the following return codes:

Hex Code	Decimal Code	Why the Error Occurred
X'0A'	10	Invalid length for the interrupt buffer extension.
X'13'	19	Depends on the value for the CONTROL parameter: • If CONTROL=YES, a control buffer has already been defined. • If CONTROL=NO, an application buffer has already been defined.
X'3E'	62	Two of the following buffers overlap: • Control buffer • External interrupt buffer • Interrupt buffer extension.
X'5C'	92	A paging or storage error was detected.

CC=3
IUCV found errors while reading your directory.

Program Exceptions

The program exceptions for DCLBFR are:

Type	Description
Addressing	The parameter list address or specified buffer address is outside the virtual machine or at virtual address zero; or, the interrupt buffer extension is outside the virtual machine storage or at virtual address zero.
Operation	The invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

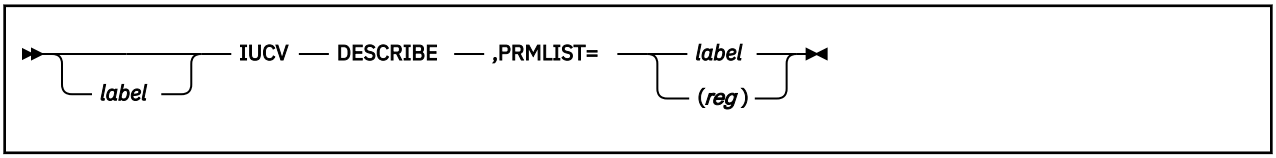
State Checks and State Changes

DCLBFR does not act on any one path and does not cause any state changes.

Completion Conditions

The DCLBFR function completes immediately.

IUCV DESCRIBE



Purpose

Use the DESCRIBE function to get the following:

- A description of a pending APPC/VM or IUCV message without receiving it
- A request-to-send indication.

DESCRIBE returns information about a message only once—the next time you invoke DESCRIBE, you get a description of the next *undescribed* message. You can issue a RECEIVE on a message after you have described it. However, it is not necessary to describe a message before receiving it.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the IUCV DESCRIBE parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Usage Notes

1. For APPC/VM messages, the message is only described if the corresponding path is in the Receive state. APPC/VM presents request-to-send indications regardless of the state of the corresponding path.
2. DESCRIBE does not describe messages that are pending on control paths.
3. If there is a function outstanding on a path, APPC/VM may report the message on the completion of that function (instead of on DESCRIBE).
4. CP considers a message described if you do one of the following:
   - Completely or partially receive a message
   - Get a message pending interrupt
   - Get a request-to-send interrupt.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	Not Possible	X	Not Possible

CC=0
DESCRIBE completed with no errors. The output parameter list of an APPC/VM message is shown in the following figure.

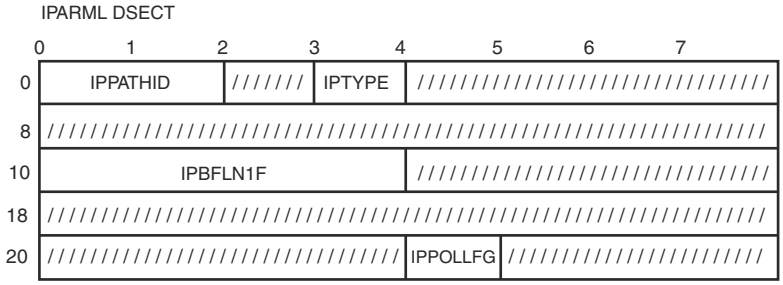


Figure 64. IUCV DESCRIBE Output Parameter List

IPPATHID
contains the path ID on which the message is pending.

IPTYPE
contains the interrupt type for a message pending (IPTYPMPA, X'89') or request-to-send (IPTYPSRA, X'88').

IPBFLN1F
contains the length of the message that is pending. This length has no relationship to the length of the APPC data stream being sent; it is only the length of the data that has arrived and is ready to be received. If the interrupt type is not X'89', this has no meaning.

IPPOLLFG
contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that another iteration of this function will probably not find a message waiting at this time.

CC=2
IUCV did not find any APPC/VM or IUCV *undescribed* messages.

Program Exceptions

The program exceptions for DESCRIBE are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

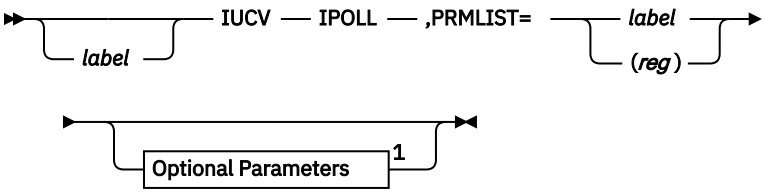
State Checks and State Changes

No state changes occur.

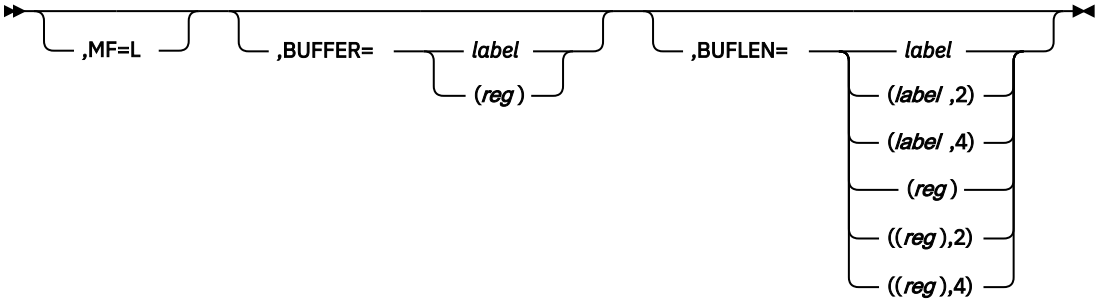
Completion Conditions

The DESCRIBE function completes immediately with CC=0 or CC=2.

IUCV IPOLL (Interrupt Poll)



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the IPOLL (Interrupt Poll) function of the IUCV macro to determine if you have any replies or incoming messages pending. If IUCV finds any replies or incoming messages pending, it returns the information about them in the buffer provided. The maximum number of pending interrupts that can be retrieved on a single request is the number of IUCV external interrupt buffers that can fit on one 4K page.

Notes:

1. Unless you disable your virtual machine for IUCV message-complete and message-pending interrupts, you should not use the IPOLL function. When the virtual machine is enabled for these interrupts, IUCV automatically informs you of message completion or arrival of an incoming message by reflecting an external interrupt to your virtual machine.
2. No external interrupt occurs for a reply represented by a message-complete returned by IPOLL.
3. No external interrupt occurs for a message represented by a message-pending returned by IPOLL. It is your responsibility to use the RECEIVE or REJECT function to process a message obtained using IPOLL.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Parameters

Required Parameters:

PRMLIST=
identifies to IUCV the address of your parameter list. The address of the parameter list must be a guest real address (an address that is real to your virtual machine), and it must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction.

BUFFER=
specifies the address of the IPOLL buffer for interrupt data.

label
is the relocatable label of the buffer.

(*reg*)
is the register number that contains the address of the buffer.

BUFLEN=
specifies the length of the IPOLL buffer for interrupt data. This length must be at least the size of an IUCV interrupt buffer and not more than 4096 bytes, and the buffer may not cross a 4K page boundary. The length need not be an exact multiple of the length of an IPARML.

label
is the relocatable label of the halfword storage location containing the length.

(*reg*)
is the register number that contains the length of the buffer.

Input Parameter List: The IUCV IPOLL parameter list has the input format shown in the following figure.

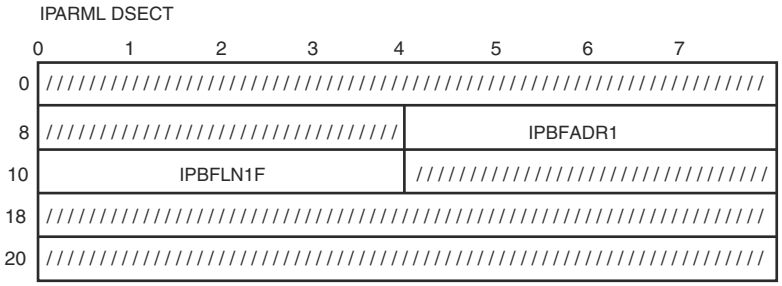


Figure 65. IUCV IPOLL Output Parameter List

IPBFADR1
contains the address of the input buffer.

IPBFLN1F
contains the length of the input buffer. For output data returned by the IPOLL function, it contains the length of the output data returned by the IPOLL function. This value will always be a multiple of the length of an IUCV external interrupt buffer (IPARML).

Output Buffer Format When the condition code is zero, the buffer contains one or more interrupt data areas for replies or messages. See the Message Complete External Interrupt, the Message Pending External Interrupt, the Message Pending External Interrupt, SENDREQ Interrupt, and APPCVM RECEIVE Output Parameter List. The remainder of the buffer not occupied by the external interrupt data remains unchanged.

If no more replies or messages are pending for the invoker, the last interrupt placed in the output buffer will have the IPNOPOLL flag set.

If IPOLL is issued in an APPC/VM environment you may receive interrupt information both for IUCV and APPC/VM paths.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	Not possible

CC=0
IUCV returned one or more messages and/or replies

CC=1
An error occurred. The output parameter list is the same as the IPOLL input parameter list, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'5C'	92	A paging or storage error was detected.

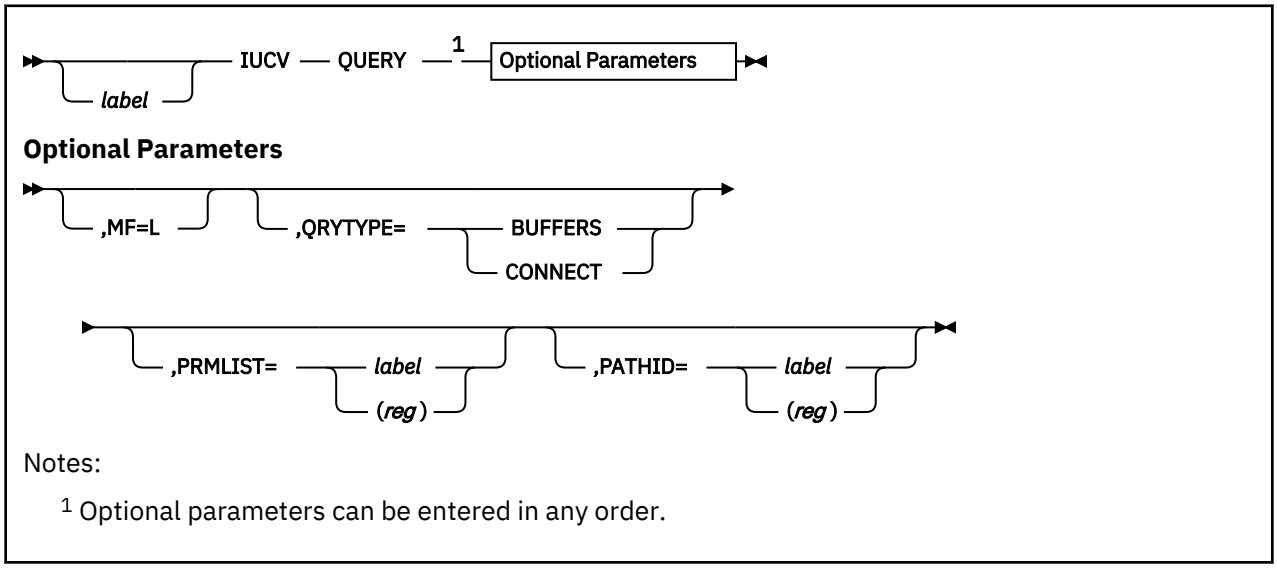
CC=2
IUCV did not find any replies or messages

Program Exceptions

The program exceptions for IPOLL are:

Type	Description
Addressing	Your parameter list address or buffer address is invalid; the address you specified is outside of your virtual machine's storage.
Operation	IUCV is inactive; you have not used the DCLBFR function to declare an external interrupt buffer.
Specification	• You did not define your parameter list on a doubleword boundary. • You specified a buffer length less than the size of an IUCV interrupt buffer. • You specified a buffer which spans a 4K page boundary.

IUCV QUERY



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the QUERY function to obtain information about the IUCV environment in your virtual machine. An application program can use the QUERY function to determine the maximum number of communication paths that can be established for your virtual machine. The value is returned in register 1.

In addition, CMS and GCS can use IUCV QUERY during initialization to determine the following:

1. The size of the external interrupt buffer. This value is returned in register 0.
2. The maximum length of data that can be returned in the external interrupt buffer extension, if you specify the PRMLIST=parameter. This value is returned in the output parameter list.
3. The maximum length Input Parameter List Extension (IPARMLX) that will be accepted on an APPC/VM connect. This value is only available if the PRMLIST=parameter was specified. This value is returned in the output parameter list.

An application program can use IUCV QUERY to obtain information about a specific connection.

Note: The IUCV QUERY function:

- Causes no state changes
- Completes immediately
- Does not involve a communication partner.

Parameters

Optional Parameter:

MF=L
generates the instructions necessary to initialize the IUCV parameter list as specified, but does not invoke the IUCV QUERY function.

QRYTYPE=
specifies the IUCV QUERY subtype. The value must represent one of the codes defined for IPQTYPE. The input QRYTYPE refines the IUCV QUERY request as follows:

BUFFERS
CP returns the length of the APPC/VM parameter/buffer extensions.

CONNECT
CP returns information about the connection on a specified PATHID.

PRMLIST=
specifies the address of the IUCV QUERY parameter list. The address must be a guest real address; the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary. The PRMLIST is optional. If it is omitted, the IUCV QUERY will only return information through the Output Registers. (This is the basic IUCV QUERY format.) If PRMLIST is specified without QRYTYPE, the IUCV QUERY will return extended buffer length information. (This is equivalent to QRYTYPE=BUFFERS.)

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

PATHID=
specifies the IUCV path ID to be interrogated by this operation. The specified path ID is assigned to the field IPPATHID in the Input Parameter List.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

Input Parameter List: IUCV QUERY may be issued without a parameter list. When PRMLIST is specified, the parameter list passed to the IUCV instruction has the format shown in Figure 66 on page 544:

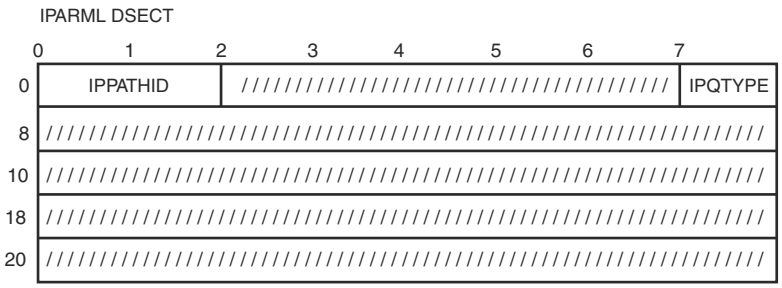


Figure 66. IUCV QUERY Input Parameter List

The parameters are as follows:

IPPATHID
contains the path ID to be interrogated for path-specific information. This field is set from the PATHID parameter, and is only relevant when IPQTYPE=IPQCONN.

IPQTYPE
IPQTYPE contains one of the following codes which represent the specified QRYTYPE:

IPQBFLN (X'00')
Return general information about the parameter and interrupt extension lengths for the APPC/VM interface. This is set as the default value for IPQTYPE when IUCV QUERY is specified with a PRMLIST value.

IPQCONN (X'01')
Return information about the connection that exists on a specific path (the one specified in IPPATHID).

Note: All other fields in the input parameter list are reserved for IBM use, and should be initialized to zero by the invoker. A nonzero value in one of these fields is not reported as an error, but may cause undesirable results at some point in the future.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	X

Note: QUERY always completes immediately.

CC=0

QUERY completed with no errors. The requested information is returned through the output parameter list. The format of the output parameter list is determined by the input IPQTYPE field value.

Output Parameter List: When IPQTYPE=IPQBFLN (Query Buffer Lengths) the IUCV QUERY output parameter list has the output format shown in Figure 67 on page 545:

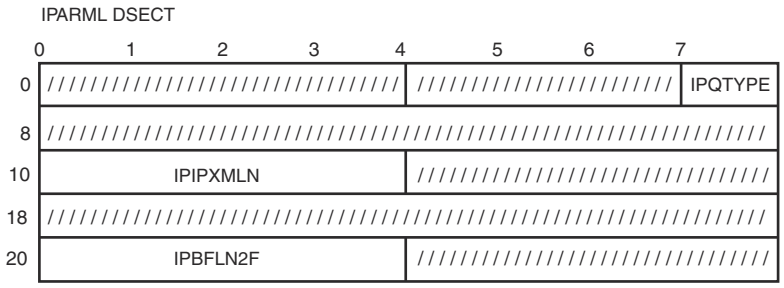


Figure 67. IUCV QUERY Output Parameter List (QRYTYPE=BUFFERS)

IPQTYPE
Contains the input QRYTYPE value

IPIPXMLN
Contains the length (in bytes) of the maximum valid APPC/VM connect input parameter list extension (IPARMLX)

IPBFLN2F
Contains the maximum length (in bytes) of data that can be reflected in the interrupt buffer extension

When IPQTYPE=IPQCONN (Query Connection) the IUCV QUERY output parameter list has the output format shown in Figure 68 on page 545 :

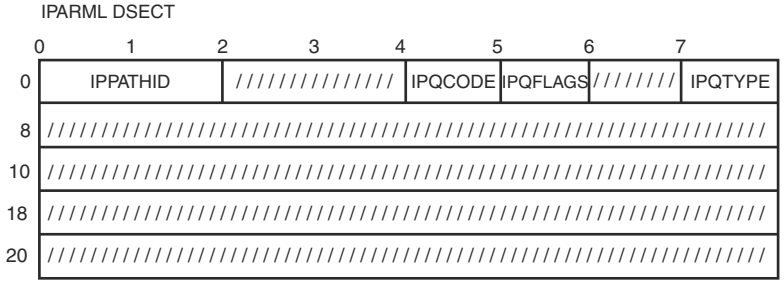


Figure 68. IUCV QUERY Output Parameter List (QRYTYPE=CONNECT)

IPPATHID
Contains the original input value (the specified path ID)

IPQCODE
Contains one of the following codes:

IPQCNNOT (0)
The specified PATHID is NOT connected at all.

IPQCNPND (1)
The specified PATHID is Connection Pending.

IPQCNCTD (2)
The specified PATHID is Connected.

IPQCNSEV (3)
The specified PATHID is Severed.

IPQFLAGS
These flags provide additional information about an IUCV path. Note that some characteristics (such as IPQAPPC) can be established for any valid path. Other characteristics (like IPQCMSRV) can only be established if IPQCODE=IPQCNCTD (indicating that the path is already connected). IPQFLAGS may contain a combination of the following flags:

IPQAPPC (X'80')
The path specified on input represents an APPC/VM connection (instead of an IUCV connection).

IPQCMSRV (X'40')
The specified path is connected to a Virtual Machine operating as a Communication Server. This normally means the path is being extended to a remote partner.

IPQISFC (X'20')
The specified path is connected to CP ISFC (Inter-System Facility for Communication) services. This normally means the path is being extended to a remote partner.

IPQCPSVC (X'10')
The specified path is connected to a CP System Service (for example, *IDENT). None of the CP System Services would extend a connection beyond the local system.

IPQTYPE
Contains the input QRYTYPE value.

CC=1
QUERY did not complete. The output parameter list is the same as the QUERY input parameter list, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that does not exist.
X'11'	17	The input IPQTYPE is not one of the supported QUERY subfunctions (refer to “Parameters” on page 543 for more information).

CC=2
QUERY did not complete. The user ID of the invoking virtual machine was not found in the CP directory, or the QUERY function was initiated while an IUCV RTRVBFR was in progress. The output parameter list is the same as the QUERY input parameter list.

CC=3
QUERY did not complete. CP encountered an error while trying to read the CP user directory entry for the invoking virtual machine. The output parameter list is the same as the QUERY input parameter list.

Program Exceptions

The program exceptions for QUERY are:

Type	Description
Operation	The invoking virtual machine is not in the supervisor state.
Specification	The parameter list is not doubleword aligned.
Addressing	The parameter list is outside the virtual machine storage.
Protection	The parameter list storage area key does not match the user's key.

Output Registers: When the IUCV QUERY code is executed, the following information is returned in the invoker's general purpose registers R0 and R1:

R0
The size (in bytes) of the external interrupt buffer for IUCV

R1
The maximum number of communication paths that can be established for your virtual machine

Usage Notes

1. IUCV QUERY does not require an IUCV interrupt buffer. So it can be issued before the IUCV DCLBFR (Declare Buffer) function.
2. IUCV QUERY completes immediately and does not cause a state change for any APPC/VM conversation.
3. An IUCV application can use a simple IUCV QUERY function (with no PRMLIST) to determine how much storage to allocate for interrupt buffers and path information. An APPC/VM application can use IUCV QUERY with QRYTYPE=BUFFERS to determine how much storage to allocate for the Input Parameter List Extension and the Interrupt Buffer Extension.
4. An IUCV or APPC/VM application can use IUCV QUERY with QRYTYPE=CONNECT to obtain information about a specific connection. For example, if the output IPQCODE=IPQCNCTD and either IPQCMSRV or IPQISFC is ON, it would be reasonable to assume that the specified PATHID is extended to a remote partner.

IUCV RTRVBFR (Retrieve Buffer)

Purpose

Use the RTRVBFR (Retrieve Buffer) function to do the following:

- Stop all IUCV and APPC/VM outstanding messages
- Sever all IUCV and APPC/VM communication paths
- End IUCV and APPC/VM communications.

When issued by a virtual machine, RTRVBFR causes all paths except control paths to be severed. For example, if a program using CMSIUCV support issues HNDIUCV CLR, CMS issues RTRVBFR and all paths are severed **except control paths**.

When issued by CP, the RETRIEVE BUFFER function severs **all** paths.

Note: Be aware that CP issues RTRVBFR for the following commands:

- SYSTEM RESET
- IPL, which issues SYSTEM RESET
- LOGOFF.

This severs **all** paths, including control paths.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	Not Possible	Not Possible	Not Possible

CC=0
normal completion.

Program Exceptions

The program exception for RTRVBFR is:

Type	Description
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.

State Checks and State Changes

RTRVBFR does not act on any one path; therefore, no state checks occur. When RTRVBFR is done executing, all paths for your virtual machine are destroyed (put in the Reset state) except your control paths, which do not change states.

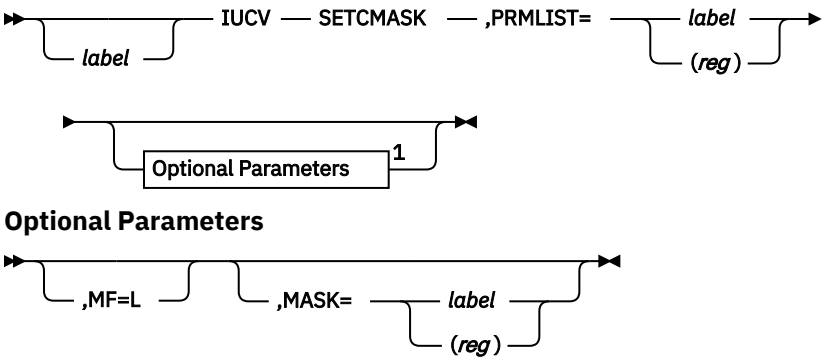
Completion Conditions

The RTRVBFR function completes immediately.

What Happens to Your VM Communication Partner

When you invoke RTRVBFR, all your noncontrol APPC/VM and IUCV communication paths are severed. APPC/VM informs your communication partners as if you issued a SEVER TYPE=ABEND. See What Happens to Your VM Communications Partner for a description of possible sever codes.

IUCV SETCMASK (Set Control Mask)



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the SETCMASK (Set Control Mask) function to enable or disable external interrupts for the following APPC/VM and IUCV control functions:

- Connection pending
- Connection complete
- Path severed
- Path quiesced (non-APPC only)
- Path resumed (non-APPC only).

You cannot use the SETCMASK function to disable interrupts on control paths.

To use this function, you must enable your virtual machine for external interrupts by setting the following bits to 1:

- Bit 7 in the virtual PSW
- Submask bit 30 in the control register 0.

You must also enable external interrupts with the SETMASK function. Otherwise, APPC/VM ignores the SETCMASK settings.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Parameters

Required Parameters:

PRMLIST=
specifies the address of the IUCV SETCMASK parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
expands the IUCV macro to generate the instructions necessary to initialize the parameter list as specified, but not to invoke the SETCMASK function.

MASK=
specifies the mask byte to determine for which, if any, of the APPC/VM and IUCV external interrupts a virtual machine is enabled for.

label
is the relocatable label of a byte containing the mask.

(*reg*)
is the register number that contains the mask in its low-order byte.

Input Parameter List: The IUCV SETCMASK parameter list has the input format shown in the following figure.

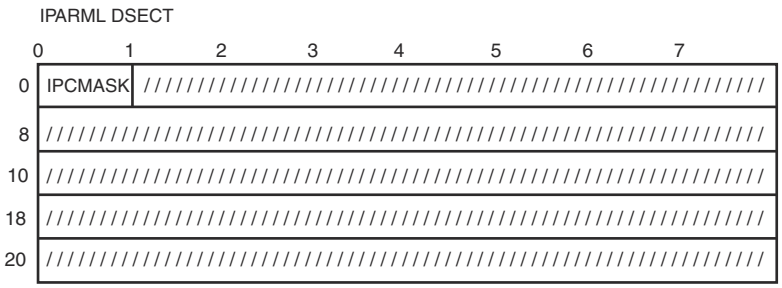


Figure 69. IUCV SETCMASK Input Parameter List

IPCMASK
specifies the mask byte to determine for which of the control interrupts your virtual machine is enabled for.

When a bit is turned off, the virtual machine is disabled for that interrupt. For example, if IPCMASK contains X'C0', this means that your virtual machine is enabled for connection pending and connection complete interrupts, but disabled for all other interrupts.

IPCLPC (X'80')
you are enabled for connection pending interrupts. This is type X'81' for APPC and type X'01' for non-APPC.

IPCLCC (X'40')
you are enabled for connection complete interrupts. This is type X'82' for APPC and type X'02' for non-APPC.

IPCLPS (X'20')
you are enabled for sever interrupts. This is type X'83' for APPC and type X'03' for non-APPC.

IPCLPQ (X'10')
you are enabled for path-quiesced interrupts. This is type X'04'; it applies to non-APPC only.

IPCLPR (X'08')
you are enabled for path-resumed interrupts. This is type X'05'; it applies to non-APPC only.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	Not Possible	Not Possible	Not Possible

CC=0
normal completion.

Program Exceptions

The program exceptions for SETCMASK are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

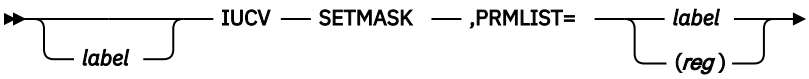
State Checks and State Changes

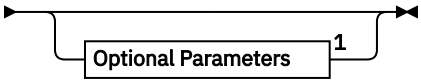
There are no states associated with the SETCMASK function.

Completion Conditions

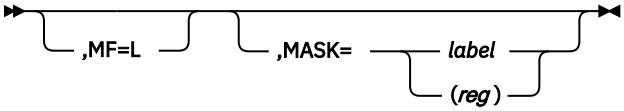
The SETCMASK function completes immediately.

IUCV SETMASK





Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the SETMASK (Set Mask) function to enable or disable external interrupts for the following APPC/VM and IUCV functions:

- Message-pending interrupts
- SENDREQ (request-to-send) interrupts
- Function-complete interrupts
- APPC/VM and IUCV control interrupts.

You cannot use the SETMASK function to disable interrupts on control paths.

To use this function, you must enable your virtual machine for external interrupts by setting the following bits to 1:

- Bit 7 in the virtual PSW
- Submask bit 30 in control register 0.

The IUCV SETMASK function specifies a byte of selective masks. This lets you mask APPC/VM and IUCV external interrupts selectively.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the IUCV SETMASK parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
expands the IUCV macro to generate the instructions necessary to initialize the parameter list as specified, but not to invoke the SETMASK function.

MASK=
lets you specify the mask byte to determine which, if any, of the APPC/VM and IUCV external interrupts a virtual machine is enabled for.

label
is the relocatable label of a byte containing the mask.

(*reg*)
is the register number that contains the mask in its low-order byte.

Input Parameter List: The IUCV SETMASK parameter list has the input format shown in the following figure.

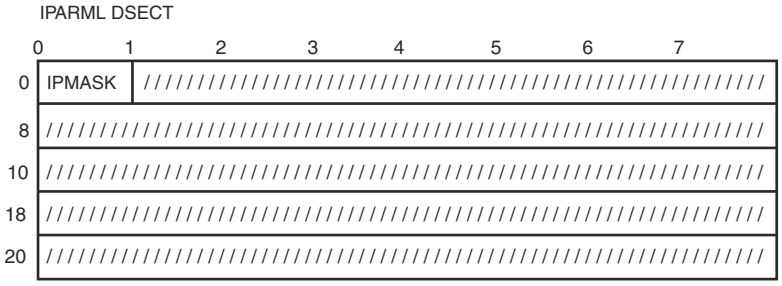


Figure 70. IUCV SETMASK Input Parameter List

IPMASK
specifies the mask byte to determine which types of interrupts a virtual machine is enabled for.

When a bit is turned off, the virtual machine is disabled for that interrupt. For example, if IPMASK contains X'C0', the virtual machine is enabled for IUCV and APPC message interrupts and APPC SENDREQ interrupts, but disabled for all other interrupts.

IPSNDN (X'80')
you are enabled for nonpriority message interrupts (type X'09', for non-APPC) and message-pending interrupts (type X'89', for APPC).

IPSNDP (X'40')
you are enabled for priority message interrupts (type X'08', for non-APPC) and SENDREQ interrupts (type X'88', for APPC).

IPRPYN (X'20')
you are enabled for nonpriority reply interrupts (type X'07', for non-APPC) and function-complete interrupts (type X'87', for APPC).

IPRPYP (X'10')
you are enabled for priority reply interrupts (type X'06', for non-APPC only).

IPCTRL (X'08')
you are enabled for control interrupts (non-APPC and APPC).

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	Not possible	Not possible	Not possible

CC=0
normal completion.

Program Exceptions

The program exceptions for SETMASK are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

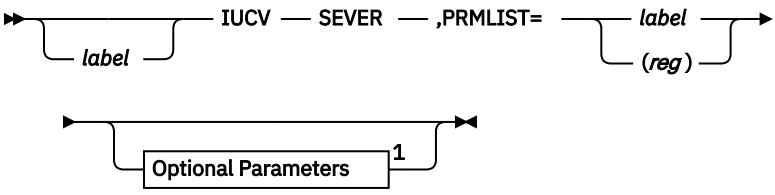
State Checks and State Changes

No states are associated with the SETMASK function.

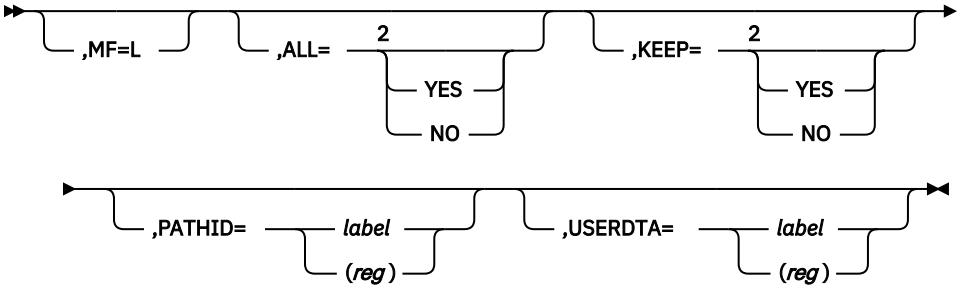
Completion Conditions

The SETMASK function completes immediately.

IUCV SEVER



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

[2] There is no default value. If a value is not specified, the parameter retains the value currently set in the parameter list.

Purpose

Use the SEVER function of the IUCV macro for the following reasons:

- To revoke ownership of a resource that you manage by ending an established path with the Identify system service (*IDENT)
- To terminate your side of the path when a connecting program severs its path or logs off
- When your program wants to terminate a path that could be either an IUCV (non-APPC) path or an APPC path.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the IUCV SEVER parameter list. The IUCV instruction is generated to reference the address specified.

label
Is the relocatable label of the parameter list.

(*reg*)
Is the register number that contains the address of the parameter list.

Optional Parameters: If you do not specify these parameters, the macro assumes that you have stored the desired values into the parameter list before invoking the IUCV macro.

MF=L
lets you build an IUCV parameter list without initializing any registers or executing the IUCV instruction. Using this format, you can then use CMSIUCV SEVER to actually issue the sever. For more information on using CMSIUCV SEVER, see *z/VM: CMS Macros and Functions Reference*.

ALL=
specifies whether all paths for this virtual machine are to be severed. When ALL is specified by a virtual machine, all paths **except control paths** are severed.

YES
indicates that all of your paths are to be severed.

NO
indicates that you do not want all of your paths severed, only the one specified by PATHID.

KEEP=
indicates whether the path ID may be reassigned by CP for another conversation immediately after the IUCV SEVER. This applies only to APPC/VM paths.

YES
indicates that the path ID is not to be freed for reuse by CP for another conversation.

Note: This value is invalid when ALL=YES.

NO
indicates that the path ID is to be freed for reuse by CP for another conversation.

PATHID=
specifies the path ID to be severed.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

USERDTA=
specifies the data area containing the 16 bytes of user data that IUCV is to reflect across the path. The user data is reflected as part of the IUCV connection-severed external interrupt. USERDTA is ignored on APPC/VM paths.

label
is the relocatable label of the user data.

(*reg*)
is the register number that contains the address of the user data.

Input Parameter List: The IUCV SEVER parameter list has the input format shown in the following figure.

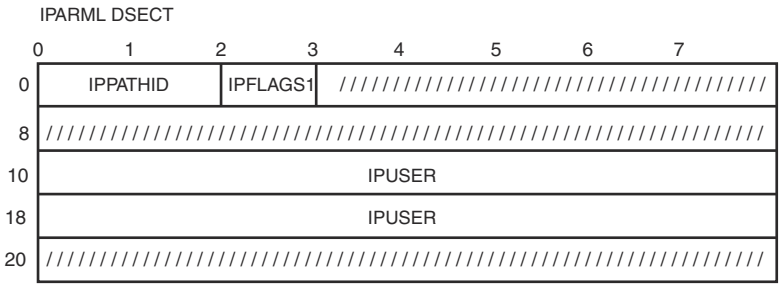


Figure 71. IUCV SEVER Input Parameter List

IPPATHID
contains the path ID of the path you want to sever.

IPFLAGS1
contains options for the SEVER function.

IPAPPC (X'08')
indicates the protocol to be used on this path. This bit must be set to 0.

IPKEEP (X'10')
indicates that the path ID is not to be freed for reuse after the IUCV SEVER completes.

Note: This flag is ignored on a non-APPC/VM path, or if the IPALL flag is set on.

IPALL (X'80')
indicates that you want to sever all paths for this virtual machine.

IPUSER
contains the user data that IUCV reflects across the path.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	Not possible	Not possible

CC=0
The SEVER completed normally. The output parameter list is shown in the following figure.

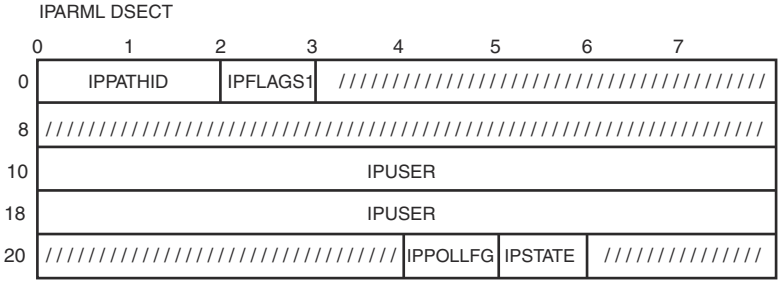


Figure 72. IUCV SEVER Output Parameter List (Sever Complete Interrupt)

IPPATHID
contains the path ID of the path you want to sever.

Note: IPPATHID is undefined when ALL=YES.

IPFLAGS1
may contain one or more of the following output bit flags:

Note: IPFLAGS1 is undefined when ALL=YES.

X'20'
reserved for IBM use only.

X'02'
reserved for IBM use only.

X'01'
reserved for IBM use only.

IPUSER
contains the user data that IUCV reflected across the path. IPUSER is ignored on APPC/VM paths.

IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

IPSTATE
contains the current state of the path:

IPRESET (X'01')
the path is in Reset state.

IPSENDST (X'03')
the path is in Send state.

IPRECVST (X'04')
the path is in Receive state.

Note: IPSTATE is undefined when ALL=YES and for non-APPC paths.

CC=1
An error occurred. The output parameter list is the same as the SEVER input parameter list, except that one of the following return codes is stored in IPRCODE (byte 3):

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that does not exist.
X'1E'	30	The IPAPPC flag in IPFLAGS1 is not 0.
X'44'	68	IUCV SEVER with KEEP=YES is an invalid function from the Reset state.
X'57'	87	IUCV SEVER with KEEP=YES is invalid when a previous APPCVM SEVER was issued with log data on this path.

Program Exceptions

The program exceptions for SEVER are:

Type	Description
Addressing	The parameter list address that you specified is outside the virtual machine's storage.
Operation	Either an application interrupt buffer has not been declared, or the invoking virtual machine is not in supervisor state.
Protection	The storage key of the specified parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

A state check occurs (IPRCODE=X'44') if KEEP=YES is specified for an APPC/VM path that is in Reset state.

When IUCV SEVER completes (CC=0), the state changes to Reset.

Completion Conditions

IUCV SEVER always completes immediately. When IUCV SEVER with KEEP=YES specified completes, the path ID is still valid, and IUCV SEVER (without KEEP=YES) may be issued from RESET state to free the path ID for reuse.

If KEEP=YES is not specified for IUCV SEVER, the path ID is no longer valid when the SEVER completes. If another function is then issued for that path ID, the function will complete with CC=1 and IPRCODE=X'01' (specified path ID not established).

What Happens to Your VM Communication Partner

Your communications partner may be affected different ways, depending on the sequence of functions that you issue. Any of the following conditions can occur when you issue:

- CONNECT, and then issue IUCV SEVER before your partner gets the connection pending interrupt. In this case, your partner does not get a connection pending interrupt or a sever interrupt.
- CONNECT, and then issue IUCV SEVER after your partner gets the connection pending interrupt, but before your partner issues ACCEPT. In this case, your partner gets a sever interrupt (assuming it is enabled for sever interrupts).
- An IUCV SEVER after receiving a connection pending interrupt, instead of issuing an ACCEPT. If your partner issued:
  - CONNECT with WAIT=NO, your partner gets a sever interrupt (assuming it is enabled for sever interrupts).
  - CONNECT with WAIT=YES, your partner's CONNECT completes with a sever indication.

If IUCV SEVER is issued on an APPC/VM path established with SYNCLVL=CONFIRM or SYNCLVL=NONE, CP reflects the SEVER to the partner as an APPCVM SEVER TYPE=ABEND with a sever code X'0610' (RESOURCE_FAILURE_NO_RETRY).

Note the following when IUCV SEVER is issued on an APPC/VM path established with APPCVM CONNECT, SYNCLVL=SYNCPT. If you issue IUCV SEVER:

- Immediately after issuing IUCV ACCEPT, CP reflects the SEVER to the partner as an APPCVM SEVER TYPE=ABEND with a sever code X'0610' (RESOURCE_FAILURE_NO_RETRY).
- From Connect state or Reset state, CP reflects the SEVER to the partner as an APPCVM SEVER TYPE=ABEND with a sever code X'0610' (RESOURCE_FAILURE_NO_RETRY).
- After the path is established, CP reflects the SEVER to the partner as an APPCVM SEVER TYPE=ABEND with sever code X'0220' (DEALLOCATE_ABEND_SVC).

Because IUCV SEVER may be issued twice on the path, once with KEEP=YES then again with KEEP=NO, the second SEVER is not reflected to the communication partner.

IUCV SEVER External Interrupt: When IUCV SEVER is issued on an IUCV path (for *IDENT connection), the sever interrupt that your communications partner gets has the format shown in the following figure.

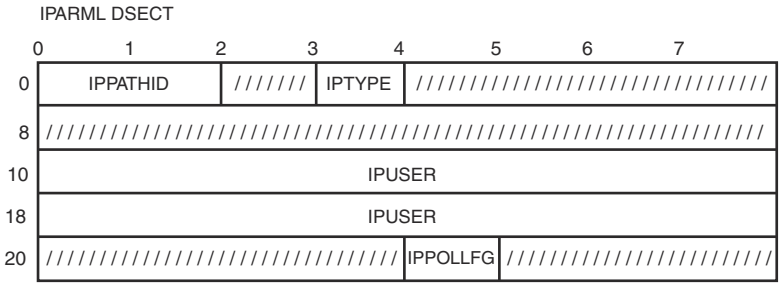


Figure 73. IUCV SEVER External Interrupt

IPPATHID
contains the path ID being severed.

IPTYPE
contains the interrupt type for IUCV SEVER (IPTYPSVC, X'03').

IPUSER
contains the user data specified by the program that severed this path.

If the sever interrupt came from *IDENT, byte 10 of this IPUSER field contains a reason code. See "*IDENT Sever Reason Codes" on page 736.

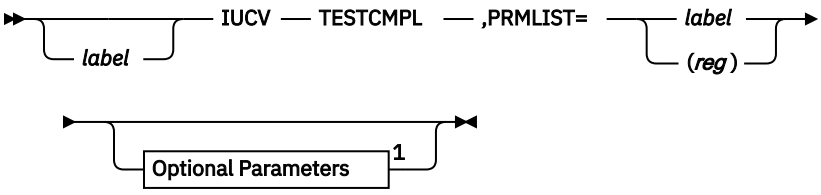
IPPOLLFG
Contains a flag returned by IUCV.

IPNOPOLL (X'80')
Indicates that an IPOLL function would not be productive for the user.

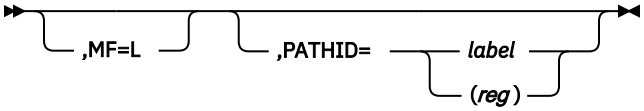
Note: When an IPNOPOLL flag is set in an interrupt, this indicates that a brief check by CP of the user's pending replies and messages reveals that an IPOLL request at this time may not be productive. If a user enables for a reply interrupt or for a message interrupt, or issues an IUCV DESCRIBE, an IUCV TESTCMPL, or an IUCV IPOLL function immediately, the user may still see a reply or message even though IPNOPOLL was set on the previous function's completion.

When IUCV SEVER is issued on an APPC/VM path, CP ignores the user data field and issues an APPCVM SEVER TYPE=ABEND to your communications partner. Your partner then gets an APPC/VM SEVER interrupt. Refer to SEVER External Interrupt for information on the sever interrupt.

IUCV TESTCMPL (Test Completion)



Optional Parameters



Notes:

[1] Optional parameters can be entered in any order.

Purpose

Use the TESTCMPL (Test Completion) function to determine if any messages or functions have been completed. You can identify a specific path when you invoke this function. If you do not specify a path, the next function on the queue of completed functions (if such a function exists) is displayed. TESTCMPL does not present functions completed on control paths.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Parameters

Required Parameter:

PRMLIST=
specifies the address of the IUCV TESTCMPL parameter list. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

(*reg*)
is the register number that contains the address of the parameter list.

Optional Parameters:

MF=L
expands the IUCV macro to generate the instructions necessary to initialize the parameter list as specified, but not to invoke the TESTCMPL function.

PATHID=
identifies the path ID to do the test completion.

label
is the relocatable label of a halfword that contains the path ID.

(*reg*)
is the register number that contains the path ID in the low-order halfword.

Input Parameter List: The IUCV TESTCMPL parameter list has the format shown in the following figure.

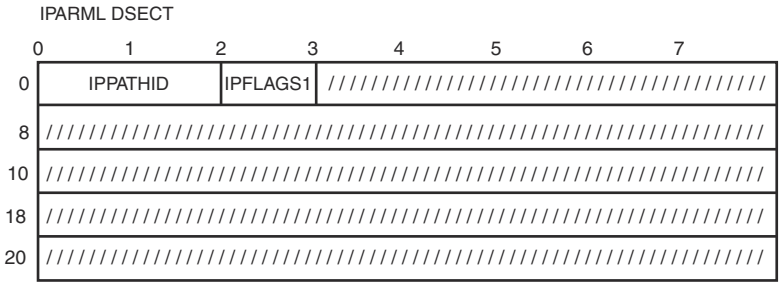


Figure 74. IUCV TESTCMPL Input Parameter List

IPPATHID
contains the path ID on which you want to complete the function. This parameter is only valid when the IPFGPID flag is set.

IPFLAGS1
contains the following input bit flag:

IPFGPID (X'02')
You specified a path ID.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	X

CC=0
indicates normal completion. The output parameter list is shown in the following figure.

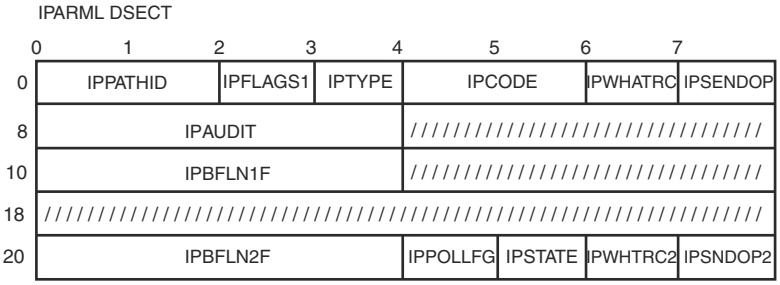


Figure 75. IUCV TESTCMPL Output Parameter List

Note: Only the contents of the IPSENDOP and IPBFLN1F fields are described here. The contents of the other fields in this parameter list depend on what function has completed. The function that has completed is indicated in IPSENDOP. Refer to the CC=2 description under the specific function being completed.

IPSENDOP
contains one of the following SEND option codes:

IPDATA (X'01')
SENDDATA RECEIVE=NO is being completed.

IPSNDRCV (X'02')
SENDDATA RECEIVE=YES is being completed.

IPERROR (X'03')
SENDERR is being completed.

IPCNFRM (X'04')
SENDCNF TYPE=NORMAL is being completed.

IPCNFSEV (X'05')
SENDCNF TYPE=SEVER is being completed.

IPSABEND (X'09')
SEVER TYPE=ABEND is being completed.

IPRECV (X'0A')
RECEIVE is being completed.

IPPREPRC (X'0C')
SENDCNF TYPE=PREPRECV is being completed.

X'0F'
reserved for IBM use only.

X'10'
reserved for IBM use only.

X'11'
reserved for IBM use only.

X'14'
is reserved for IBM use only.

X'16'
reserved for IBM use only.

IPBFLN1F
contains the length of pending log data for you to receive. This field is only meaningful when IPWHATRC is equal to IPSABEND or IPERROR.

CC=1
means an error occurred. The parameter list format is the same as the input shown in the TESTCMPL input parameter list except that the following return code is stored in IPRCODE:

Hex Code	Decimal Code	Why the Error Occurred
X'01'	1	You specified a path ID that is not yet established.

CC=2
no APPC/VM function completes, or IUCV message completes were found.

CC=3
a nonzero IPAUDIT value was stored.

Program Exceptions

The program exceptions for the IUCV TESTCMPL are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Checks and State Changes

TESTCMPL sets the state of the path on which the function is being completed. The state that is set depends on which function is being completed; refer to the individual function description for details.

Completion Conditions

The TESTCMPL function completes immediately.

IUCV TESTMSG (Test Message)

Purpose

Use the TESTMSG (Test Message) function to avoid using external interrupt handling. When you invoke the TESTMSG function, your virtual machine enters the WAIT state if none of the following are pending:

- SENDREQ interrupts (APPC/VM)
- Function-complete interrupts (APPC/VM)
- Message-pending interrupts (IUCV and APPC/VM)
- Message-complete interrupts (IUCV).

If any of these interrupts are pending while your virtual machine is in the WAIT state, the virtual machine reexecutes the TESTMSG function. TESTMSG then returns a condition code. TESTMSG ignores APPC/VM message pending interrupts unless the path corresponding to the message pending is in the Receive state. TESTMSG does not receive or describe the interrupt. You must use RECEIVE, DESCRIBE, or TESTCMPL, or enable for interrupts to clear the interrupt.

Note that CMS does not currently support IUCV or APPC/VM virtual MP functions.

Condition Codes and Return Codes

CC=0	CC=1	CC=2	CC=3
Not possible	X	X	X

CC=1
means a message or SENDREQ indication is pending.

CC=2
means a message completion or function completion is pending.

CC=3
means one or more conditions causing a condition code 1 and one or more conditions causing a condition code 2 are pending.

Program Exceptions

The program exception for TESTMSG is:

Type	Description
Operation	Either an application interrupt buffer is not declared, or the invoking virtual machine is not in the supervisor state.

State Checks and State Changes

No states are associated with the TESTMSG function.

Completion Conditions

The TESTMSG function completes when control is returned to your virtual machine.

Chapter 9. Migrating Programs from IUCV to APPC/VM

The Inter-User Communications Vehicle (IUCV) provides a way for program-to-program communications within one z/VM system. A program using IUCV can communicate with itself, with a CP system service, or with another program on the same system. IUCV is not part of the APPC (SNA LU 6.2) architecture. Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521 has more details on IUCV.

APPC/VM, which is z/VM's implementation of APPC (SNA LU 6.2) protocol, includes some IUCV support. However, because IUCV is not part of the APPC architecture, it is important to know the differences between APPC/VM and IUCV.

To start, APPC/VM depends on a half-duplex protocol, while IUCV communication uses a full-duplex protocol. In support of half duplex protocol, APPC/VM defines and enforces states on each path.

In addition, in APPC/VM the high-order bit of the IPTYPE field is set to designate APPC/VM from IUCV interrupts. (The IPTYPE field is part of an interrupt.)

APPC/VM and IUCV each provide a set of communication functions. This chapter outlines the differences and similarities between APPC/VM functions and IUCV functions.

APPC/VM and IUCV Functions That Work Differently

The following functions are supported in both APPC/VM and IUCV, but work differently. For more information about the APPC/VM function discussed in this section, see Chapter 7, “APPCVM Macro Functions,” on page 411. For more information about the IUCV functions discussed in this section, see Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

- **CONNECT**

  The RESID on the APPCVM CONNECT defines the target of the connection. The target could be on the same z/VM system, a different system within the same TSAF collection, or in an SNA network. For IUCV CONNECT, the USERID defines the target. The target must be on the same z/VM system.[4]

  Other APPC/VM differences are:

  – APPCVM CONNECT does not have a message limit parameter, whereas IUCV CONNECT does. This is because the message limit is always one on APPC/VM paths.
  – APPCVM CONNECT does not have a user data field.
  – APPCVM CONNECT has a WAIT=YES option.
  – You cannot use APPCVM CONNECT to connect to a CP system service.
  – A connection parameter list extension is defined for APPCVM CONNECT that lets you specify additional information needed to complete the connect.

  Note that resource manager programs in APPC/VM must use IUCV CONNECT to connect to the Identify system service. This function is described in “IUCV CONNECT” on page 529.

- **RECEIVE**

  APPC/VM differences are:

  – You must provide a path ID on APPCVM RECEIVE.

[4] This chapter focuses on the migration of traditional, single-system IUCV communication. In general, migration from a distributed IUCV environment would have the same considerations. However, the target of a CONNECT does not need to be on the same z/VM system with distributed IUCV. Distributed IUCV might also be considered as an alternative to migrating to APPC. Refer to “IUCV in a Distributed Environment” on page 309 for more information.

© Copyright IBM Corp. 1991, 2020

- You can issue APPCVM RECEIVE before data arrives on a path.
- APPCVM RECEIVE has a WAIT=YES option.
- APPCVM RECEIVE has a PIP=YES option.

- **SEND** (for APPC/VM, **SENDDATA**)

  APPCVM SENDDATA and IUCV SEND differences are:

  - APPCVM SENDDATA does not have a parameter list data option.
  - APPCVM SENDDATA does not have a priority message option.
  - APPCVM SENDDATA does not have any special message identifiers (a message class or a message tag).
  - APPCVM SENDDATA has a WAIT=YES option.
  - APPCVM SENDDATA has a RECEIVE=YES option that lets you define an answer area. IUCV SEND has a TYPE=2WAY option that lets you define an answer area.

    The APPC/VM user responds to a SENDDATA RECEIVE=YES with a SENDDATA. The length of the response does not depend on the size of the answer area. The IUCV user, on the other hand, responds to a SEND TYPE=2WAY with a REPLY. The length of the response cannot be bigger than the size of the answer area.
  - With APPC/VM, the data sent is in logical record format. With IUCV, the data can be in any format.

- **SEVER**

  APPC/VM differences are:

  - APPCVM SEVER does not have a user data field.
  - APPCVM SEVER has a WAIT=YES option.
  - There are two APPCVM SEVER types, TYPE=NORMAL and TYPE=ABEND. There is only one IUCV SEVER type.
  - APPCVM SEVER has a function complete interrupt for log data.

  Resource manager programs must use IUCV SEVER to:

  - Sever a connection to the Identify system service
  - Sever an APPC/VM connection before a path is established.

For more information about IUCV SEVER, refer to on page “IUCV SEVER” on page 556.

IUCV Functions Not Supported on APPC/VM Paths

The following IUCV functions are not supported on APPC/VM paths:

- **PURGE**
- **QUIESCE** and **RESUME**

  (However, because the message limit on APPC/VM paths is one, an application can quiesce a path by not receiving a message pending on that path.)
- **REJECT**

  (However, APPCVM SENDERR is similar to IUCV REJECT.)
- **REPLY**

  (However, APPCVM SENDDATA can be used in place of IUCV REPLY. Refer to “APPCVM SENDDATA” on page 475 for more information.)

APPC/VM Functions Not Supported on IUCV Paths

The following APPC/VM functions are not supported on IUCV paths:

- **QRYSTATE**

  (IUCV does not have any equivalent functions.)
- **SENDCNF** and **SENDCNFD**

  (IUCV does not have any equivalent functions.)
- **SENDERR**

  (However, IUCV REJECT is similar to APPCVM SENDERR.)
- **SENDREQ**

  (However, APPCVM SENDREQ is similar to an IUCV priority 1WAY parameter data SEND, when that SEND is used as a signal and does not contain any data.)
- **SETMODFY**

  (IUCV does not have any equivalent functions.)

Shared APPC/VM and IUCV Functions

Several functions are shared for both APPC/VM and IUCV. This section describes these functions as they relate to an APPC/VM environment. Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521 describes the IUCV versions of these functions unrelated to APPC/VM. Parameters other than those described in this document are available, but have no meaning on APPC/VM paths and are ignored.

Shared Functions That Can Be Used in CMS

The following two functions are documented in Chapter 7, “APPCVM Macro Functions,” on page 411 and can be safely used in a CMS environment:

- **ACCEPT**

  When ACCEPT is issued to establish an APPC path,

  – ACCEPT does not have a message limit parameter, whereas non-APPC ACCEPT does. This is because the message limit is always one on APPC/VM paths.

  – ACCEPT does not have a user data field.
- **QUERY**

  QUERY gives you the following information about a virtual machine:

  – The size of the external interrupt buffer

  – The maximum number of communication paths that can be established for the virtual machine.

  – The maximum length of the interrupt buffer extension used in CMS and GCS.

Shared Functions That Should Be Avoided in CMS

Other functions are also shared for both APPC/VM and IUCV. These shared functions should not be used in a CMS environment because they could affect other programs in the same virtual machine; however, they can be used safely in a non-CMS environment.

Each of these functions is briefly described in the following list, then described in detail in Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

- **DCLBFR (Declare Buffer)**

DCLBFR declares an interrupt buffer. (Both APPC/VM and IUCV interrupts are presented in the same buffers.)

DCLBFR should not be directly issued by a program in CMS; HNDIUCV uses DCLBFR to initialize the virtual machine's APPC/VM environment.

- **DESCRIBE**

  DESCRIBE gives the following information:

  – The next message pending on non-APPC paths
  – The next message pending on an APPC path that is in Receive state
  – A SENDREQ on an APPC path.

  DESCRIBE should not be used in CMS because this function clears the pending-message external interrupt for the described message. This interrupt may not belong to the issuer of the DESCRIBE function; thus, other programs running in the same virtual machine can be affected because the message is lost and never reflected to the true target.

- **IPOLL (Interrupt Poll)**

  IPOLL determines if there are any replies or incoming messages pending. If IUCV finds any replies or incoming messages pending, information about them is stored in the buffer provided. The maximum number of interrupts that can be retrieved on a single request is the number of interrupt buffers which can fit on one page. INTERRUPT POLL can be used instead of DESCRIBE and TEST COMPLETION to avoid some of the processing overhead involved in large numbers of messages and replies.

  IPOLL can be directly issued by a program in CMS. If it is not, APPC/VM completes the first message on the REPLY queue for the entire virtual machine, and that interrupt may not belong to the application that issued the IPOLL. Thus, other programs running in the same virtual machine can be affected because the message or reply is lost and never reflected to the true target.

- **RTRVBFR (Retrieve Buffer)**

  RTRVBFR releases an interrupt buffer. (Both APPC/VM and IUCV interrupts are presented in the same buffers.)

  RTRVBFR should not be directly issued by a program in CMS; HNDIUCV and CMS abend processing use RTRVBFR to terminate a virtual machine's APPC/VM environment.

- **SETMASK** and **SETCMASK**

  SETMASK and SETCMASK disable and enable APPC and non-APPC interrupts.

  These functions should not be used by a program in CMS because they disable certain APPC/VM external interrupts for the entire virtual machine. Thus, other programs running in the same virtual machine may be affected.

- **TESTCMPL (Test Completion)**

  TESTCMPL determines the next APPC or non-APPC function that has completed.

  TESTCMPL can be directly issued by a program in CMS; however, the issuer must be careful that a message ID or path ID is specified in the IUCV parameter list. If it is not, APPC/VM completes the first message on the REPLY queue for the entire virtual machine, and that message may not belong to the application that issued the TESTCMPL.

- **TESTMSG (Test Message)**

  TESTMSG waits for the following:

  – A message pending or message complete interrupt on non-APPC paths
  – A message pending interrupt on an APPC path that is in Receive state
  – A request-to-send interrupt on an APPC path
  – A function complete interrupt on an APPC path.

TESTMSG should not be used by a program in CMS because it places the entire virtual machine in a wait state if no incoming messages or replies are pending. Thus, other programs running in the same virtual machine may be affected.

Chapter 10. APPC Mapped with APPC/VM

The APPC/VM assembler interface implements the base set and various options sets of APPC (SNA LU 6.2) communication functions. This chapter details the relationship between APPC and APPC/VM, and then maps APPC/VM functions with the APPC functions provided with the SNA LU 6.2 protocol.

For more information on the LU 6.2 protocol, see these documents:

- *SNA Format and Protocol Reference Manual: Architectural Logic for LU Type 6.2*
- *SNA Transaction Programmer's Reference Manual for LU 6.2*.

APPC Conversations

User programs and resource manager programs in z/VM must follow the rules of an APPC conversation. APPC/VM supports and enforces these rules, as described in the following sections.

Note: An APPC conversation is represented within a TSAF or CS collection as an APPC/VM path. SNA sessions have no representation in a TSAF or CS collection. VTAM allocates and ends the SNA session on which APPC/VM conversations are established.

Establishing a Conversation

If your virtual machine manages a resource, and another virtual machine is trying to establish a path to the resource, you receive a connection pending interrupt. Check to be sure that the connection pending interrupt is for an APPC connection. Do not assume that the program trying to connect is on the local TSAF or CS collection, or that the program is a VM program.

The server virtual machine is responsible for invoking the transaction program (resource manager program) and verifying the contents of the FMH5. CP recognizes nothing smaller than the server virtual machine. CMS and GCS recognize nothing smaller than a program. In general, in the CMS and GCS environments, each inbound connection does not cause the resource manager to create another instance of the transaction program. Instead, the program is notified that another path is being established.

It is the program's responsibility to receive the Attach FMH5 (optionally) and save its relevant contents. In addition, the program must issue an IUCV ACCEPT before communicating on the APPC/VM path. ACCEPT is not part of the APPC architecture. If there is something wrong in the Attach FMH5 data (for example, the program does not support the synchronization level specified), then it is the program's responsibility to sever the connection with the appropriate sever code.

After the CONNECT/ACCEPT sequence has been successfully completed on both sides, the two programs can exchange data using the half-duplex protocol of an APPC conversation. APPC/VM fully supports the base set of APPC communication and enables programs to support basic and mapped conversations.

APPC/VM Interrupts

APPC/VM uses external interrupts to signal certain events and can be categorized as follows:

- Those interrupts that let applications process other paths while waiting for input or a function to complete on other paths:
  - Message pending interrupts
  - Connection pending interrupts
  - Function complete interrupts
  - Connection complete interrupts.
- Those interrupts that asynchronously indicate your partner has issued a SENDREQ or a SEVER:
  - Request-to-send interrupts

© Copyright IBM Corp. 1991, 2020

– Sever interrupts.

This interrupt-oriented signalling of your partner's activity is unique to APPC/VM; it does not map to any APPC architected function. In situations where your APPC/VM conversation is carried on a VTAM link, interrupts describing your partner's issuing of SEVER or SENDREQ may not be delivered to you until you perform some other operation on the conversation. This is because VTAM provides no asynchronous means for delivering the SENDREQ or SEVER indication to APPC/VM VTAM Support (AVS), and therefore the notification cannot be passed along to your program.

APPC/VM supports some IUCV macro functions that do not correspond to any APPC function. A program that wants to avoid non-APPC functions should be enabled only for connection pending interrupts.

APPC/VM Conversation States

APPC/VM defines several states that a program can be in during a conversation. Most of these are based on states defined by APPC. The following table lists the conversation states for APPC and the APPC/VM counterparts. (Note that several APPC/VM states do not map to an APPC state.)

Table 76. APPC Conversation States and Corresponding APPC/VM Implementation

APPC State	**APPC/VM Implementation**
Backout Required	Backout_Received Backout_Required
Confirm Confirm Send Confirm Deallocate	Confirm Confirm Confirm
Deallocate	Sever
Defer Receive Defer Deallocate	Defer_Receive Defer_Sever
Receive	Receive
Reset	Reset
Send	Send
Sync-Point	Prepare_Received or Unsolicited_Request_Commit_Received (if the sync-point control modifier is request send.)
Sync-Point Send	Prepare_Received or Unsolicited_Request_Commit_Received (if the sync-point control modifier is request receive.)
Sync-Point Deallocate	Prepare_Received or Unsolicited_Request_Commit_Received (if the sync-point control modifier is request sever.)
N/A	Committed_Received
N/A	Connect
N/A	Solicited_Request_Commit_Received

APPC/VM Return Codes

CP reports errors that it finds in the IPRCODE or IPAUDIT field. The target application or communications servers report errors that they find in the IPCODE field on SEVER or SENDERR functions; this same condition may be reported in IPRCODE, IPAUDIT, or IPCODE depending on the following:

- Whether the path goes through a communications server (TSAF and AVS), and
- Where CP detected the error along the path.

CP does not report the error in both IPRCODE/IPAUDIT and IPCODE at the same time.

An application should sever if it gets an error in IPRCODE. The application can sever the path using the APPCVM SEVER function with a SEVER code X'0210' to indicate DEALLOCATE_ABEND_PROG.

The return code mapping tables in this chapter have entries for the return code (or condition code), and the corresponding IPRCODE and/or IPCODE. **APPC/VM return codes that do not correspond to defined APPC return codes are not discussed in this chapter**.

Notes:

1. All APPC return codes are mapped to a return code and/or sever code in APPC/VM.
2. The LU 6.2 Architecture for SYNCPT conversations has secondary return codes (reason codes) for each of the following APPC return codes:
   - DEALLOCATE_ABEND
   - DEALLOCATE_ABEND_PROG
   - DEALLOCATE_ABEND_SVC
   - DEALLOCATE_ABEND_TIMER
   - DEALLOCATE_NORMAL
   - RESOURCE_FAILURE_NO_RETRY
   - RESOURCE_FAILURE_RETRY

   VM does not support these. An application should always initiate a roll-back process when it gets one of the above return codes on a SYNCLVL=SYNCPT conversation.

APPC Verb Names Mapped to APPC/VM Macro Functions

APPC/VM supports the base set of APPC functions. The following table lists the base set of APPC verbs for basic conversations, and their APPC/VM counterparts:

Table 77. Base Set of APPC Verbs and APPC/VM Functions

APPC Verb	APPC/VM Implementation
ALLOCATE	CONNECT
CONFIRM	SENDCNF TYPE=NORMAL
CONFIRMED	SENDCNFD
DEALLOCATE	SEVER or SENDCNF TYPE=SEVER
FLUSH	SENDDATA FLUSH=YES, RECEIVE=NO, BUFLEN=0
GET_ATTRIBUTES	*No specific functions, but indirect support*
RECEIVE_AND_WAIT	RECEIVE
REQUEST_TO_SEND	SENDREQ
SEND_DATA	SENDDATA RECEIVE=NO

Table 77. Base Set of APPC Verbs and APPC/VM Functions (continued)

APPC Verb	APPC/VM Implementation
SEND_DATA followed by RECEIVE_AND_WAIT	SENDDATA RECEIVE=YES
SEND_ERROR	SENDERR

In addition, the APPC PREPARE_TO_RECEIVE verb (from the *Prepare to Receive* option set) is implemented in APPC/VM with the RECEIVE or SENDCNF TYPE=PREPRECV functions.

Here's how other types of APPC base set verbs map to APPC/VM:

- Mapped conversation verbs

  APPC/VM lets programs support mapped conversations. You must specify TYPE=MAPPED on the APPCVM CONNECT to do this. For mapped conversations, the APPC/VM implementation is roughly the same as in the preceding table.
- Operator control verbs

  Some operator control commands are provided for APPC/VM VTAM Support (AVS). The following table shows the APPC verbs that have equivalents for AVS. APPC operator control verbs not shown here do not have AVS equivalents.

  Table 78. APPC Operator Control Verbs Mapped to AVS Commands

APPC Verb	AVS Equivalent
CHANGE_SESSION_LIMIT	AGW CNOS command
INITIALIZE_SESSION_LIMIT	AGW CNOS command
RESET_SESSION_LIMIT	AGW CNOS command

  For more information on the AGW CNOS command, see *z/VM: Connectivity*.

In addition, APPC/VM implements APPC type-independent conversation verbs as follows:

- You can use the GET_TYPE function by providing a general-purpose application running on top of APPC/VM.
- The SYNCPT and BACKOUT verbs are implemented on VM for protected conversations through two interfaces:
  - In CMS you can use several callable services library routines to initiate a sync-point or roll back process. For more information on these routines, see *z/VM: CMS Callable Services Reference*.
  - The APPC/VM assembler interface contains several macros that initiate a sync-point or roll back process.

APPC ALLOCATE

The APPC ALLOCATE verb maps to the APPCVM CONNECT function.

Note: Do not make assumptions about the target of the APPCVM CONNECT when the CONNECT completes—your CONNECT may complete before the target program is even invoked.

Parameters: The following list maps APPC ALLOCATE parameters (in bold) to APPC/VM equivalents (in italics):

LU_NAME - ***locally known LU name in the connection parameter list extension***
In APPC/VM, locally known LU names are 16 bytes.

MODE_NAME - ***mode name in the connection parameter list extension***
In APPC/VM, mode names are 8 bytes.

TPN - ***transaction program name in the connection parameter list extension***

TYPE - ***TYPE= parameter on APPCVM CONNECT***
APPC/VM supports the APPC options:

- TYPE(BASIC_CONVERSATION) as TYPE=BASIC
- TYPE(MAPPED_CONVERSATION) as TYPE=MAPPED.

RETURN_CONTROL - ***RETURN= parameter on APPCVM CONNECT***
APPC/VM supports the APPC options:

- RETURN_CONTROL(WHEN_SESSION_ALLOCATED) as RETURN=ALLOCD
- RETURN_CONTROL(IMMEDIATE) as RETURN=IMMED.

APPC/VM does not support the APPC option RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED).

SYNC_LEVEL - ***SYNCLVL= parameter on APPCVM CONNECT***
APPC/VM supports the APPC options:

- SYNC_LEVEL(NONE) as SYNCLVL=NONE
- SYNC_LEVEL(CONFIRM) as SYNCLVL=CONFIRM
- SYNC_LEVEL(SYNCPT) as SYNCLVL=SYNCPT.

SECURITY - ***security fields in the connection parameter list extension***
APPC/VM supports the APPC options:

- SECURITY(NONE) as a X'01' value in the security type field of the connection parameter list extension.
- SECURITY(SAME) as a X'00' value in the security type field of the connection parameter list extension, and the user ID in the connection pending interrupt.
- SECURITY(PGM(USER_ID PASSWORD)) as a X'02' value in the security type field of the connection parameter list extension, and the user ID and password in the security fields of this extension.

SECURITY(PGM(PROFILE)) is not supported.

PIP - ***address of PIP data in the connection parameter list extension***
APPC/VM supports the PIP option set on the source and target program.

A source program specifies information about PIP data in the connection parameter list extension. This consists of the address of the PIP data, length of the PIP data, and a flag to indicate whether the data is presented in a single buffer or multiple buffers.

The target program must make a special indication when receiving PIP data. It does this by specifying PIP=YES on the APPCVM RECEIVE.

RESOURCE - ***IPPATHID in connect complete interrupt***
In APPC/VM, the path ID is a halfword number.

RETURN_CODE - ***IPRCODE*** **and** ***IPCODE***
The APPC RETURN_CODE variable corresponds to the following in APPC/VM:

- IPRCODE in the APPCVM CONNECT output parameter list
- IPCODE in the APPCVM SEVER external interrupt.

The connecting program must look at the IPRCODE when it receives a CC=1 on CONNECT. Also, if your partner rejects the connection with SEVER (CC=2), then the connecting program must look at IPCODE to determine the allocation error.

RETURN_CODE	IPRCODE	IPCODE
OK	0	Not applicable
ALLOCATION_ERROR		
ALLOCATION_FAILURE_RETRY	X'0B',X'0C',X'0D',X'0E'	X'0111'
ALLOCATION_FAILURE_NO_RETRY	X'0B'	X'0110'
UNSUCCESSFUL	X'0F'	X'0112'
SECURITY_NOT_VALID	X'31'	X'0160'
TRANS_PGM_NOT_RECOGNIZED	X'0B'	X'0142'
TRANS_PGM_NOT_AVAILABLE_RETRY	X'0C',X'0D',X'0E'	X'0141'
TRANS_PGM_NOT_AVAILABLE_NO_RETRY	X'0F'	X'0140'
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	X'89'	X'0131'
PARAMETER_ERROR		
Invalid LU name	X'28'	X'0301'
Invalid mode name	X'29'	X'0302'

State Changes: For both APPC ALLOCATE and APPCVM CONNECT, you, the invoker, are in Send state when the function successfully completes.

Abend Conditions: The parameter check condition is as follows:

Parameter Check Condition	IPRCODE
The program is not allowed to specify MODE_NAME(SNASVCMG), or MODE_NAME(SNASVCMG) is not supported.	X'29'

APPC CONFIRM

The APPC CONFIRM verb maps to the APPC/VM function, SENDCNF TYPE=NORMAL.

Parameters: The following list maps APPC CONFIRM parameters (in bold) to APPC/VM equivalents (in italics):

RESOURCE - *PATHID parameter of APPCVM SENDCNF*
The resource ID returned in APPC/VM is a path ID. The path ID is a halfword number.

REQUEST_TO_SEND_RECEIVED - *SENDREQ interrupt*
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

RETURN_CODE - *IPCODE*
The APPC RETURN_CODE variable corresponds to the following in APPC/VM:

- IPCODE from APPCVM SENDERR or APPCVM SEVER.

If the SENDCNF completes with a SENDERR or SEVER, then the virtual machine that invoked SENDCNF should look at the IPCODE field to determine the error.

RETURN_CODE	IPCODE
OK	X'0000'
ALLOCATION_ERROR	Any allocation error code (X'0110' through X'0160')
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'

RETURN_CODE	IPCODE
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes: No state changes occur.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
SYNC_LEVEL(NONE) was specified	X'25'
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send state	X'20', X'22', X'23', X'24', X'44'-X'4C'
Conversation started but did not finish sending a logical record	X'2C'

APPC CONFIRMED

The APPC CONFIRMED verb maps to the APPCVM SENDCNFD function.

Parameters: The following list maps the APPC CONFIRMED parameter (in bold) to the APPC/VM equivalent (in italics):

RESOURCE - *PATHID parameter on APPCVM SENDCNFD*
In APPC/VM, the path ID is a halfword number.

State Changes: Your program can be in any of the following states:

- RECEIVE, if the SENDCNFD is in response to a SENDCNF TYPE=NORMAL.
- SEVER, if the SENDCNFD is in response to a SENDCNF TYPE=SEVER. Sever state is the APPC/VM equivalent of Deallocate state.
- SEND, if the SENDCNFD is in response to a SENDCNF TYPE=PREPRECV.
- SEND or RECEIVE, if the SENDCNFD is in response to the partner's backout.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Confirm state	X'20', X'21', X'22', X'24', X'44', X'4A'-X'4C'

APPC DEALLOCATE

The APPC DEALLOCATE verb maps to the APPC/VM functions SEVER and SENDCNF TYPE=SEVER.

Parameters: The following list maps APPC DEALLOCATE parameters (in bold) to APPC/VM equivalents (in italics):

RESOURCE - ***PATHID parameter of APPCVM SENDCNF and APPCVM SEVER***
In APPC/VM, the path ID is a halfword number.

TYPE - ***TYPE and CODE parameters of APPCVM SEVER***

TYPE(SYNC_LEVEL)
Use the following APPC/VM functions:

- APPCVM SEVER TYPE=NORMAL, to do a DEALLOCATE TYPE(SYNC_LEVEL) when SYNCLVL=NONE.
- APPCVM SENDCNF TYPE=SEVER, followed by APPCVM SEVER TYPE=NORMAL, to do a DEALLOCATE TYPE(CONFIRM) when SYNCLVL=CONFIRM.
- APPCVM SETMODFY TYPE=SEVER, followed by a function to initiate a sync-point, to do a DEALLOCATE TYPE(SYNC_LEVEL) when SYNCLVL=SYNCPT.
- APPCVM SETMODFY TYPE=SEVER, followed by APPCVM CONFIRM to do a DEALLOCATE TYPE(SYNC_LEVEL) when SYNCLVL=CONFIRM.

TYPE(FLUSH)
Use APPCVM SEVER TYPE=NORMAL.

TYPE(CONFIRM)
Use the APPCVM SENDCNF TYPE=SEVER, followed by APPCVM SEVER TYPE=NORMAL, to do a TYPE(CONFIRM).

TYPE(ABEND_PROG)
Use APPCVM SEVER TYPE=ABEND with the appropriate sever code (CODE=X'210').

TYPE(ABEND_SVC)
Use APPCVM SEVER TYPE=ABEND with the appropriate sever code (CODE=X'220').

TYPE(ABEND_TIMER)
Use APPCVM SEVER TYPE=ABEND with the appropriate sever code (CODE=X'230').

TYPE(LOCAL)
Use APPCVM SEVER TYPE=NORMAL after receiving a sever indication from your partner.

LOG_DATA - ***BUFFER and BUFLEN parameters of APPCVM SEVER***
The LOG_DATA can be from 8 to 600 bytes in length. APPC/VM supports the APPC option:

- LOG_DATA(NO) as BUFLEN=0
- LOG_DATA(YES) as BUFLEN > 0.

RETURN_CODE - ***IPRCODE*** **and** ***IPCODE***
For all types of DEALLOCATE except SYNC_LEVEL(CONFIRM), the only possible return code is OK. For SYNC_LEVEL(CONFIRM), the same mapping exists as for the return codes from CONFIRM. See the return code table under “APPC CONFIRM” on page 578 for details.

State Changes: After the Sever completes, your program is in Reset state.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Issued TYPE(FLUSH), TYPE(CONFIRM), or TYPE(SYNC_LEVEL) from the wrong state.	X'20', X'22', X'23', X'24'
Issued TYPE(LOCAL) from the wrong state.	X'20', X'21', X'22', X'23'
Issued TYPE(FLUSH) or TYPE(SYNC_LEVEL), and the conversation started but did not finish sending a logical record.	X'2C'

APPC FLUSH

The APPC FLUSH verb maps to APPCVM SENDDATA FLUSH=YES, RECEIVE=NO, BUFLEN=0.

Parameters: The following list maps APPC FLUSH parameters (in bold) to APPC/VM equivalents (in italics):

RESOURCE - *PATHID parameter of APPCVM SENDDATA*
In APPC/VM, the path ID is a halfword number.

State Changes: Your program will be in Receive state if the SENDDATA FLUSH=YES is issued from Defer_Receive state. Otherwise, no state change occurs.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send state	X'20', X'22', X'23', X'24', X'44'-X'4C'

APPC GET_ATTRIBUTES

APPC/VM provides the function of the GET_ATTRIBUTES verb, but does not provide a specific APPC/VM function.

The transaction program is responsible for obtaining the following:

- The PARTNER_LU_NAME from the VM area, after initially receiving the allocate data.
- The MODE_NAME from the VM area, after initially receiving the allocate data.
- The SYNC_LEVEL in the connection pending interrupt and/or Attach FMH5.
- The SECURITY_USER_ID from the connection pending interrupt and/or the Attach FMH5.

APPC/VM does not provide SECURITY_PROFILE.

APPC PREPARE_TO_RECEIVE

The APPC PREPARE_TO_RECEIVE verb maps to the APPC/VM function SENDCNF TYPE=PREPRECV.

Parameters: The following list maps APPC PREPARE_TO_RECEIVE parameters (in bold) to APPC/VM equivalents (in italics):

RESOURCE - *PATHID parameter on APPCVM SENDCNF*
In APPC/VM, the path ID is a halfword number.

TYPE(FLUSH) - *APPCVM RECEIVE with BUFLEN=0 from Send state*
In APPC/VM, if you are in Send state and issue RECEIVE with a receive area of 0 length, this is equivalent to APPC PREPARE_TO_RECEIVE TYPE(FLUSH).

LOCKS - *no parameter*
APPC/VM does not support the *LOCKS* option set.

RETURN_CODE - *IPCODE*
The APPC RETURN_CODE variable corresponds to the following in APPC/VM:

- The IPCODE of APPCVM SENDERR or APPCVM SEVER TYPE=ABEND.

If the SENDCNF completes with a SENDERR or SEVER, then the virtual machine that invoked SENDCNF should look at the IPCODE field to determine the error.

The same mapping exists as for the return codes from CONFIRM. See the return code table under "APPC CONFIRM" on page 578 for details.

State Changes: After this function completes, your program is in Receive state.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send state	X'20', X'22', X'23', X'24', X'44'-X'4C'

APPC RECEIVE_AND_WAIT

The APPC RECEIVE_AND_WAIT verb maps to the APPC/VM function, RECEIVE.

Parameters: The following list maps APPC RECEIVE_AND_WAIT parameters (in bold) to APPC/VM equivalents (in italics):

RESOURCE - *PATHID parameter on APPCVM RECEIVE*
In APPC/VM, the path ID is a halfword number.

FILL - *no parameter*
APPC/VM supports FILL(BUFFER), but does not support FILL(LL). Support for FILL(LL) can be provided by a program.

DATA - *BUFFER parameter of APPCVM RECEIVE*
The APPC parameter, DATA(variable), and the APPC/VM parameter, BUFFER=, specifies the address of the buffer to place the data being received.

LENGTH - *BUFLEN parameter of APPCVM RECEIVE*
The APPC parameter, LENGTH(variable), and the APPC/VM parameter, BUFLEN=, define the RECEIVE area length.

When control is returned to the program at the completion of RECEIVE_AND_WAIT, the LENGTH variable contains the length of data received. For APPC/VM, the length variable, IPBFLN2F, contains one of the following:

- The amount of space left in the buffer
- A count of how much data is pending that did not fit into the buffer.

When you are in Send state and specify a 0 length, the Receive completes before the target responds. This maps to PREPARE_TO_RECEIVE TYPE (FLUSH). When you issue RECEIVE with a 0 length from Receive state, it completes immediately even if nothing is pending on the path. In APPC/VM, you can

use RECEIVE and then wait for a message pending or sever interrupt on the same path to do the equivalent of an APPC RECEIVE_AND_WAIT with a 0 length.

REQUEST_TO_SEND_RECEIVED - *SENDREQ interrupt*
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

WHAT_RECEIVED - *IPWHATRC in function complete interrupt*
You can receive data along with other indicators.

WHAT_RECEIVED(DATA)
In APPC/VM, this indication is presented by either IPWHATRC=IPDATA, or IPWHATRC does not equal IPDATA with the length field (IPBFLN2F) less than IPBFLN1F when you issued RECEIVE.

WHAT_RECEIVED(DATA_COMPLETE,DATA_INCOMPLETE, LL_TRUNCATED)
Does not occur in APPC/VM, because APPC/VM does not support FILL(LL).

WHAT_RECEIVED(SEND)
In APPC/VM, this indication is IPWHATRC=IPSEND.

WHAT_RECEIVED(CONFIRM)
In APPC/VM, this indication is IPWHATRC=IPCNFRM.

WHAT_RECEIVED(CONFIRM_SEND)
In APPC/VM, this indication is IPWHATRC = IPSNDCNF.

WHAT_RECEIVED(CONFIRM_DEALLOCATE)
In APPC/VM, this function is IPWHATRC=IPCNFSEV.

WHAT_RECEIVED(TAKE_SYNCPT)
In APPC/VM, this indication can be either IPWHATRC = IPPREPAR or, IPWHATRC = IPREQCOM and IPWHATRC2=IPTPRECV.

WHAT_RECEIVED(TAKE_SYNCPT_SEND)
In APPC/VM, this indication can be either IPWHATRC = IPPREPAR or, IPWHATRC = IPREQCOM and IPWHATRC2=IPTPSEND.

WHAT_RECEIVED(TAKE_SYNCPT_DEALLOCATE)
In APPC/VM, this indication can be either IPWHATRC = IPPREPAR or, IPWHATRC = IPREQCOM and IPWHATRC2=IPTPSEVR.

RETURN_CODE - *IPCODE in function complete interrupt*
The APPC RETURN_CODE variable corresponds to the following in APPC/VM:

- The IPCODE of APPCVM SENDERR or APPCVM SEVER TYPE=ABEND
- The IPWHATRC field of the APPCVM RECEIVE output parameter list.

If the RECEIVE completes with a SENDERR or SEVER TYPE=ABEND, then the virtual machine that issued RECEIVE should look at the IPCODE field to determine the error. IPWHATRC could also contain IPSNORM to indicate DEALLOCATE_NORMAL.

RETURN_CODE	IPCODE
OK	X'0000'
ALLOCATION_ERROR	Any allocation error code (X'0110' through X'0160')
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_NO_TRUNC	X'0410'
PROG_ERROR_TRUNC	X'0420'
PROG_ERROR_PURGING	X'0430'

RETURN_CODE	IPCODE
SVC_ERROR_NO_TRUNC	X'0510'
SVC_ERROR_TRUNC	X'0520'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes: When RECEIVE completes, your program may be in any of the following states:

- Receive state, when WHAT_RECEIVED is DATA
- Send state, when WHAT_RECEIVED is SEND
- Confirm state, when WHAT_RECEIVED is CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.
- Prepare_Received state, when IPWHATRC=IPPREPAR
- Unsolicited_Request_Commit_Received state, when IPWHATRC=IPREQCOM.

No state change occurs when the verb is issued in Receive state and WHAT_RECEIVED is DATA.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send or Receive state	X'20', X'23', X'24', X'44'-X'4C'
Conversation started but did not finish sending a logical record	X'2C' (see note)

Note: This condition could also be reported in IPAUDIT by the IPADITRN flag.

APPC REQUEST_TO_SEND

The APPC REQUEST_TO_SEND verb maps to the APPC/VM function SENDREQ.

Parameters: The following maps the APPC REQUEST_TO_SEND parameter (in bold) to the APPC/VM equivalent (in italics):

RESOURCE - ***PATHID parameter of APPCVM SENDREQ***
In APPC/VM, the path ID is a halfword number.

State Changes: No state changes occur.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send, Receive, or Confirm state	X'20', X'24', X'44'-X'46', X'48', X'4A'-X'4C'

APPC SEND_DATA

The APPC SEND_DATA verb maps to the APPC/VM function SENDDATA RECEIVE=NO.

Parameters: The following list maps the APPC SEND_DATA parameters (in bold) to the APPC/VM equivalents (in italics):

RESOURCE - ***PATHID parameter of APPCVM SENDDATA***
In APPC/VM, the path ID is a halfword number.

DATA - ***BUFFER parameter of APPCVM SENDDATA***
The APPC parameter, DATA(variable), and the APPC/VM parameter, BUFFER=, specify the address of the data to send.

LENGTH - ***BUFLEN parameter of APPCVM SENDDATA***
The APPC parameter LENGTH(variable) and the APPC/VM parameter BUFLEN= specify the length of the data to send.

REQUEST_TO_SEND_RECEIVED - ***SENDREQ interrupt***
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

RETURN_CODE - ***IPCODE in function complete interrupt***
The APPC RETURN_CODE variable corresponds to the following in APPC/VM:

- The IPCODE of APPCVM SENDERR or APPCVM SEVER.

If the SENDDATA completes with a SENDERR or SEVER, the virtual machine that issued the SENDDATA should look at the IPCODE field to determine the error.

RETURN_CODE	IPCODE
OK	X'0000'
ALLOCATION_ERROR	Any allocation error code (X'0110' through X'0160')
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes: No state changes occur.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource ID	X'01'
Invalid logical record length	X'2A' (see note)

Note: This condition could also be reported in IPAUDIT by the IPADIINV flag.

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send state	X'20', X'22', X'23', X'24', X'44'-X'4C'

APPC SEND_ERROR

The APPC SEND_ERROR verb maps to the APPC/VM function SENDERR.

Parameters: The following list maps the APPC SEND_ERROR parameters (in bold) to the APPC/VM equivalents (in italics):

RESOURCE - ***PATHID parameter of APPCVM SENDERR***
In APPC/VM, the path ID is a halfword number.

TYPE - ***TYPE parameter of APPCVM SENDERR***
APPC/VM supports APPC options:

- TYPE(PROG) as TYPE=PROG
- TYPE(SVC) as TYPE=SVC.

LOG_DATA - ***BUFFER and BUFLEN parameters of APPCVM SENDERR***
The APPC LOG_DATA variable corresponds to the APPC/VM BUFFER and BUFLEN. The LOG_DATA can be from 8 to 600 bytes in length. APPC/VM supports the APPC option:

- LOG_DATA(NO) as BUFLEN=0
- LOG_DATA(YES) as BUFLEN with a value > 0.

REQUEST_TO_SEND_RECEIVED - ***SENDREQ interrupt***
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

RETURN_CODE - ***IPCODE in function complete interrupt***
The APPC RETURN_CODE variable corresponds to the following in APPC/VM:

- The IPCODE of APPCVM SENDERR or APPCVM SEVER TYPE=ABEND
- The IPWHATRC field of the APPCVM SENDERR output parameter list.

If the SENDERR completes with an indication that the communication partner issued a SENDERR or SEVER, the virtual machine should look at the IPCODE field to determine the error.

If you issue SEND_ERROR from Send state, the following return codes are possible. IPWHATRC may also contain IPSNORM to indicate DEALLOCATE_NORMAL.

RETURN_CODE	IPCODE
OK	X'0000'
ALLOCATION_ERROR	Any allocation error code (X'0110' through X'0160')
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

If you issue SEND_ERROR from Receive state, the following return codes are possible. IPWHATRC may also contain IPSNORM to indicate DEALLOCATE_NORMAL.

RETURN_CODE	IPCODE
OK	X'0000'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

If you issue SEND_ERROR from Confirm state, the following return codes are possible:

RETURN_CODE	IPCODE
OK	X'0000'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes: If you issue the SENDERR, you remain in or are put into Send state.

Abend Conditions: The parameter check conditions follow:

Parameter Check Condition	IPRCODE
LOG_DATA not supported	Not supported
Invalid resource ID	X'01'

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in Send, Receive, or Confirm state	X'20', X'24', X'44'-X'46', X'48', X'4A'- X'4C'

Part 4. CP System Services

This part contains the following chapters:

- Chapter 11, “Access Verification System Service (*RPI),” on page 591
- Chapter 12, “Account System Service (*ACCOUNT),” on page 699
- Chapter 13, “Asynchronous CP Command Response System Service (*ASYNCMD),” on page 719
- Chapter 14, “DASD Block I/O System Service (*BLOCKIO),” on page 721
- Chapter 15, “Error Logging System Service (*LOGREC),” on page 729
- Chapter 16, “Identify System Service (*IDENT),” on page 731
- Chapter 17, “Message System Service (*MSG),” on page 739
- Chapter 18, “Message All System Service (*MSGALL),” on page 741
- Chapter 19, “SCLP System Service (*SCLP),” on page 743
- Chapter 20, “Signal System Service (*SIGNAL),” on page 747
- Chapter 21, “Spool System Service (*SPL),” on page 751
- Chapter 22, “Symptom System Service (*SYMPTOM),” on page 773
- Chapter 23, “VM Event System Service (*VMEVENT),” on page 775

Note: For information on Monitor System Service (*MONITOR), see Appendix A, in *z/VM: Performance*.

© Copyright IBM Corp. 1991, 2020

Chapter 11. Access Verification System Service (*RPI)

The access verification system service (*RPI) handles IUCV communications between the CP access control interface (ACI) and an external security manager (ESM) service virtual machine, such as the Resource Access Control Facility (RACF). The ACI is a group of CP modules that mediate between CP and the ESM to handle authorization checking. z/VM supplies stub modules that are replaced by the ESM when it is installed. Part of this ESM-supplied code becomes the *RPI system service. Therefore, the *RPI system service is available only if an ESM is installed and the interface to *RPI is as defined by the ESM.

Using the CP Access Control Interface

The CP *access control interface* (ACI) is a group of modules that mediate between CP and an *external security manager* (ESM). An ESM is a service virtual machine that runs outside the primary operating system to help maintain the latter's security and integrity. The RACF Security Server for z/VM is an example of an ESM.

The access control interface consists of the CP modules: HCPRPD, HCPRPF, HCPRPG, HCPRPI, HCPRPW, HCPRPP, and HCPRWA, and the ACIPARMS control block. These modules are supplied in CP as stub modules. When an ESM is installed, these modules can be replaced with ESM versions that do not exceed one page in size. These modules are installation-wide exits. HCPRPD, HCPRPI, HCPRPP, and HCPRPW comprise the portion of the ACI in which CP takes an active part. CP routes control to these modules and expects certain return conditions. HCPRPF and HCPRPG are provided solely for ESM use. HCPRWA and HCPRPF are nonexecutable modules which can be used as data areas.

HCPPWAPF is an entry point that can be called by the ESM to notify CP of the level of support provided for optional features, such as password phrases. HCPPWA is not part of the ACI and it is not intended to be modified by the ESM.

The CP module, HCPDA0, may be called by an ESM to query or to update the security bit settings for each command, DIAGNOSE, and security relevant system function. The *security bit settings* control the calls that CP makes to the ACI.

The following topics are discussed in this chapter:

- “Overview” on page 592
- “HCPRPI Module” on page 593
- “HCPRPW Module” on page 596
- “HCPRPD Module” on page 599
- “HCPRPE Module for handling DIAGNOSE X'A0'” on page 602
- “HCPRPF Module” on page 606
- “HCPRPG Module” on page 606
- “HCPRPP Module” on page 606
- “HCPRWA Module” on page 607
- “CP Callable Services for the ACI” on page 607
- “Summary of CP Modules and Entry Points” on page 608
- “ACI Security Bits” on page 609
- “HCPDA0 Module for Updating ACI Security Bits” on page 612
- “ACIPARMS Control Block” on page 622
- “CP Calls to the ACI” on page 639

© Copyright IBM Corp. 1991, 2020

- "Generic Command and DIAGNOSE Audit Calls" on page 639
- "ACIPARMS Parameter Lists for CP Commands" on page 641
- "ACIPARMS Parameter Lists for DIAGNOSE Codes" on page 670
- "ACIPARMS Parameter Lists for System Functions" on page 678.

Overview

CP's SSI Configuration Manager defines a "security" service that calls entry points in the module HCPRPP to handle the following SSI related events: Worthiness checks; Enablement function; Join processing; State change notification; STABLE preparation; SSI level change. See "HCPRPP Module" on page 606 for more details. CP and the ESM use a control block called ACIPARMS to communicate with each other about security-relevant events. An event may be a CP command, DIAGNOSE code, or system function.

Figure 76 on page 592 provides an overview of the CP access control interface and the suggested method for an ESM to use it. The general security steps are as follows:

1. The ESM replaces the CP stub modules HCPRPW, HCPRPI, HCPRPD, HCPRPF, HCPRPG, and HCPRWA with its own tailored versions.
2. When a command, DIAGNOSE, or security-relevant system function is issued, CP checks the ACI security bit settings for that event. If the ACI needs to be called, CP creates an ACIPARMS parameter list specifically for that request and then calls the ACI through either HCPRPI or HCPRPW, sending ACIPARMS as input.
3. The request is then passed to the ESM through IUCV.
4. The ESM performs the requested function, records its response in the appropriate ACIPARMS field (ACICODE), and returns the ACIPARMS control block to CP.
5. CP then carries out the ESM's security decision. The process continues at step "2" on page 592 for the next event.

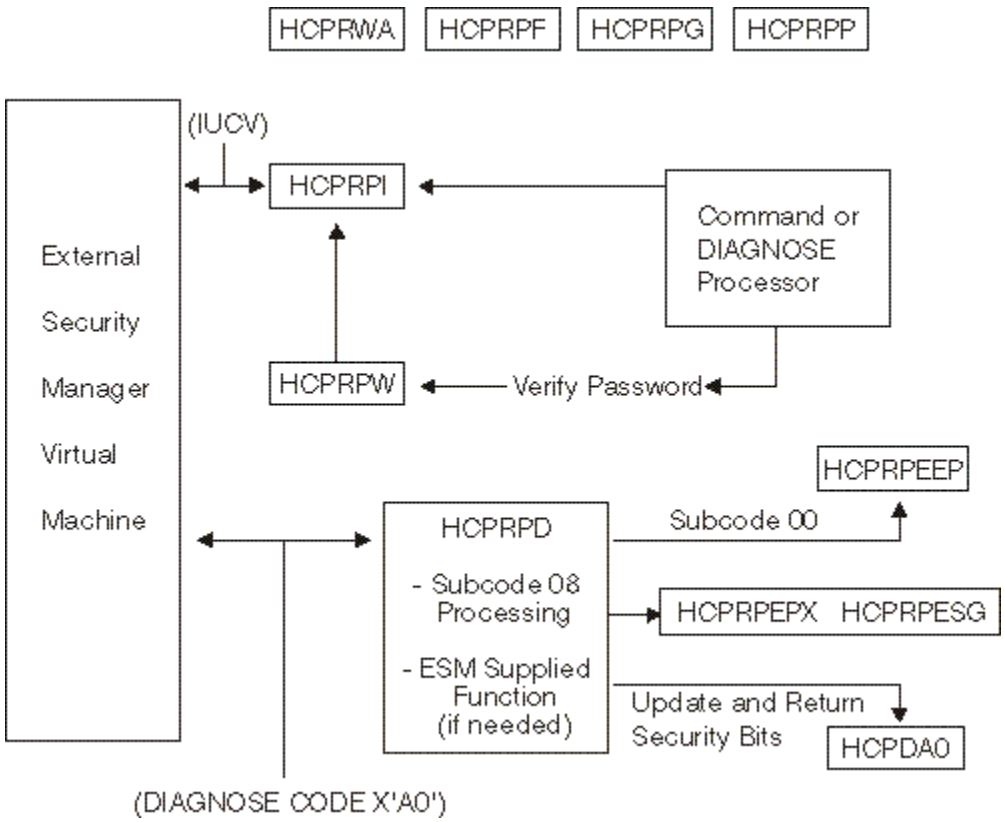


Figure 76. Overview of the CP Access Control Interface to an ESM

HCPRPI Module

The HCPRPI module contains six entry points: HCPRPICN, HCPRPIIL, HCPRPISV, HCPRPIQS, HCPRPIRM, and HCPRPIRA.

Entry Points HCPRPICN, HCPRPIIL, HCPRPISV, HCPRPIQS, and HCPRPIRM - IUCV Interface

The HCPRPICN, HCPRPIIL, HCPRPISV, HCPRPIQS, and HCPRPIRM entry points are defined to support an IUCV interface. These entry points comprise the skeleton for the *RPI system service through which the Security Manager service virtual machine can communicate with the ACI modules. These entry points, along with their respective assigned functions, are defined to CP's IUCV processing as the interrupt handlers for the *RPI system service:

Entry Point
Function

HCPRPICN
IUCV Connect interrupt handler

HCPRPIIL
IUCV message pending interrupt handler

HCPRPISV
IUCV Sever interrupt handler

HCPRPIQS
IUCV Quiesce interrupt handler

HCPRPIRM
IUCV Resume interrupt handler

For example, if a virtual machine issues an IUCV connect to *RPI, CP's IUCV processing routes control to HCPRPICN.

The ESM may replace these skeleton entry points with code to perform the IUCV services required to communicate with the ESM virtual machine. This ESM-supplied code becomes the *RPI system service. In conjunction with this, the ESM virtual machine must contain the IUCV code needed to communicate with the *RPI system service. The ESM virtual machine must use the same protocol for IUCV communications as the ESM-supplied interrupt handlers.

For information about IUCV, see Part 2, “The Inter-User Communications Vehicle,” on page 295.

CP's stub version of the HCPRPICN entry point issues an IUCV sever for the path that the connect was on.

CP's stub versions of the HCPRPIIL, HCPRPISV, HCPRPIQS, and HCPRPIRM entry points issue a defer (ACIDEFR) return code in the ACICODE field of ACIPARMS, and return to CP. However, there is no CP handling of this return code, and the ESM replacement module does not need to reproduce it.

Interface specifications for the HCPRPICN, HCPRPIIL, HCPRPISV, HCPRPIQS, and HCPRPIRM entry points follow.

Input Registers:

R1
Address of the external interrupt.

The interrupt buffer is mapped by IPARML, and the contents differ depending on the type of interrupt.

R11
Address of the dispatched VMDBK

Output: None

Attributes: MP, dynamic, resident

Linkage: Call with a dynamic save area

Register Usage:

R0
Scratch

R1
Address of external interrupt

R2
Scratch

R3
Scratch

R4
Scratch

R5
Scratch

R6
Scratch

R7
Scratch

R8
Scratch

R9
Scratch

R10
Scratch

R11
Address of dispatched VMDBK

R12
Base register

R13
Address of save area

R14
Scratch

R15
Scratch

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

CP's stub module expects register 1 to contain the address of ACIPARMS on input and output. However, the ESM's replacement module should expect R1 to contain the address of the external interrupt buffer on input; there are no output requirements.

Entry Point HCPRPIRA - Request Services from the ESM

The HCPRPIRA entry point is defined to support authorization requests (calls) from CP to the ESM. For a subset of commands and DIAGNOSE codes, CP calls this entry point to check whether the user is authorized *before* it performs the requested function. CP sends the address of ACIPARMS in Register 1. ACIPARMS contains information about the command or DIAGNOSE code that has been issued. The calls to this entry point and the format of ACIPARMS for each call are described in detail in "CP Calls to the ACI" on page 639.

CP's stub module returns a code of ACIDEFR in the ACICODE field in ACIPARMS. This indicates to CP that the ESM has made no authorization check.

The ESM replacement module is expected to receive the input parameters in ACIPARMS, and to send the ESM's response back to the caller in the ACICODE field.

Figure 77 on page 595 lists the interface specifications for this entry point.

Input Registers:

R1
Address of ACIPARMS parameter list

R11
Address of the dispatched VMDBK

Output:

R1
Address of ACIPARMS parameter list

ACICODE field contains the return code

Attributes: MP, dynamic, resident

Linkage: Call with a dynamic save area

Register Usage:

R0
Scratch

R1
Address of ACIPARMS

R2
Scratch

R3
Scratch

R4
Scratch

R5
Scratch

R6
Scratch

R7
Scratch

R8
Scratch

R9
Scratch

R10
Scratch

R11
Address of dispatched VMDBK

R12
Base register

R13
Address of save area

R14
Scratch

R15
Scratch

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Figure 77. Interface Specifications for the HCPRPIRA Entry Point

Exit from HCPRPI

Return from all the executable entry points in HCPRPI is made to the calling module by the HCPEXIT macro, located in the HCPPSI macro library.

The HCPEXIT macro is coded in HCPRPI as:

```
HCPEXIT EP=(HCPRPIRA,HCPRPICN,HCPRPIIL,HCPRPIQS,HCPRPIRM,HCPRPISV)
```

Each entry point branches to this instruction to exit.

HCPRPW Module

The HCPRPW module contains two entry points, HCPRPWEP and HCPRPWPR.

Entry Point HCPRPWEP - Logon Password Verification Routine

This entry point is defined to support calls from CP to the ESM for password checking for the following commands:

- LOGON
- AUTOLOG
- XAUTOLOG.

For LOGON, AUTOLOG, and XAUTOLOG, CP calls the HCPRPWEP entry point to check whether the user is authorized *before* it processes the command. CP sends the address of the ACIPARMS parameter list in Register 1. ACIPARMS contains information about the command that has been issued. The format of ACIPARMS for each command is described in “CP Calls to the ACI” on page 639.

CP's stub module returns a code of ACIDEFR in the ACICODE field. This indicates to CP that the ESM has made no authorization check.

The ESM replacement module is expected to receive the input parameters in ACIPARMS, and to send the ESM's response back to the caller in the ACICODE field.

Figure 78 on page 597 lists the interface specifications for this entry point.

Input Registers:

R1
Address of ACIPARMS parameter list

R11
Address of the dispatched VMDBK

Output:

R1
Address of ACIPARMS parameter list

ACICODE field contains the return code

Attributes: Non-MP, dynamic, resident

Linkage: Call with a dynamic save area

Register Usage:

R0
Scratch

R1
Address of ACIPARMS

R2
Scratch

R3
Scratch

R4
Scratch

R5
Scratch

R6
Scratch

R7
Scratch

R8
Scratch

R9
Scratch

R10
Scratch

R11
Address of dispatched VMDBK

R12
Base register

R13
Address of save area

R14
Scratch

R15
Scratch

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Figure 78. Interface Specifications for the HCPRPWEP Entry Point

Exit from HCPRPWEP

Return is made to the calling module by the HCPEXIT macro, located in the HCPPSI macro library. The HCPEXIT macro is coded in HCPRPWEP as:

```
HCPEXIT EP=(HCPRPWEP)     Return to Caller
```

Entry Point HCPRPWPR - Logon Password Prompting Routine

This entry point prompts for password/phrase data and passes the entered data back in ACIPARMS. CP calls the HCPRPWPR entry point to prompt for and collect password/phrase data for later verification by entry point HCPRPWEP. CP sends the address of the ACIPARMS parameter list in Register 1.

CP's stub module returns a code of ACIDEFR in the ACICODE field. This indicates to CP that no ESM is installed.

The ESM replacement module is expected to receive the input parameters in ACIPARMS, and to send the response back to the caller in the ACICODE, ACIOPPLN and ACIOPP fields.

Interface specifications for the HCPRPWPR entry point follow.

Input Registers:

R1
Address of ACIPARMS parameter list

Output:

R1
Address of ACIPARMS parameter list

ACICODE field contains the return code
X'08' - the password/phrase string has been successfully collected
X'10' - a console error occurred

ACIOPPLN = length of entered data
ACIOPP = entered data

Attributes: Non-MP, dynamic, resident

Linkage: Call with a dynamic save area

Register Usage:

R0
Scratch

R1
Address of ACIPARMS

R2
Scratch

R3
Scratch

R4
Scratch

R5
Scratch

R6
Scratch

R7
Scratch

R8
Scratch

R9
Scratch

R10
Scratch

R11
Address of dispatched VMDBK

R12
Base register

R13
Address of save area

R14
Scratch

R15
Scratch

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Exit from HCPRPWPR

Return is made to the calling module by the HCPEXIT macro, located in the HCPPSI macro library. The HCPEXIT macro is coded in HCPRPWPR as:

```
HCPEXIT EP=(HCPRPWPR)     Return to Caller
```

HCPRPD Module

The HCPRPD module contains one entry point, HCPRPDEP, which functions as the DIAGNOSE code X'A0' processor.

Entry Point HCPRPDEP - DIAGNOSE Code X'A0' Processor

CP's stub version of HCPRPDEP contains a call to module HCPRPE (entry point HCPRPEEP), which handles subcodes X'00' and X'08' of DIAGNOSE code X'A0'. Subcode X'00' retrieves the ACI group name, ACIGROUP, for a given user ID from the user's directory. Subcode X'08' sets the condition code to indicate whether an ESM is installed. Subcode X'48' retrieves the ESM product information from HCPRWAPP and copies it into the buffer pointed to by the RX register. These subcodes are documented in “DIAGNOSE Code X'A0' – Obtain ACI Information” on page 134. Upon return from HCPRPE, HCPRPD stores registers R0 and R15 in the caller's R0 and R15 registers, and returns to the caller.

For subcode X'00', the ESM's replacement module should duplicate the call to HCPRPE as it appears in the stub module so that this subcode continues to be supported in the same way.

For subcode X'08' the ESM should set the guest (virtual machine) condition code to 0 indicating that an ESM is installed. The condition code is set using the HCPCALL macro, located in the HCPGPI macro library. The call is coded as:

```
HCPCALL HCPGSVC0        Set guest condition code to zero
```

For subcode X'48', the ESM does not need to modify the stub code in HCPRPD and HCPRPE, but should update the ESM product information in the HCPRWAPP table. This table is mapped by the RWAESM DSECT in the ACIPARMS macro.

An ESM may also provide additional functions in this entry point by defining its own subcodes for DIAGNOSE code X'A0' and by handling those subcodes in HCPRPD. The subcodes should be specified in the Ry register. If the ESM is to contain calls to HCPDA0, it is suggested that these calls be implemented as DIAGNOSE code X'A0' subcodes, and made from HCPRPD. See “HCPDA0 Module for Updating ACI Security Bits” on page 612 for more information.

The interface specifications for the HCPRPDEP entry point follow.

Input Registers:

R5
Address of the Rx register

R6
Address of the Ry register

R11
Address of the dispatched VMDBK

See DIAGNOSE code X'A0' for a description of Rx and Ry register specifications for subcodes X'00', X'08', and X'48'.

Exit Values:

Subcode X'00':
Normal:

R15 =
0 (Successful request)

Rx =
First doubleword is unchanged. Second doubleword contains the ACI group name.

Ry =
Unchanged

Guest condition code = 0

Error:

- If the input user ID is invalid:

  R15 = 0
  Virtual machine condition code = 1

- Otherwise:

  R15 = 8 (Unsuccessful request)
  R0 = X'04' Protection exception indicator
  R0 = X'05' Addressing exception indicator
  R0 = X'06' Specification exception indicator
  Virtual machine condition code = 1

Subcode X'08':

R15 =
0 (Successful request)

Rx =
Unchanged

Ry =
Unchanged

Guest condition code is set to 0 = ESM is installed

Guest condition code is set to 1 = ESM is not installed

Subcode X'48':

R15 =
0 (Successful request)

Rx =
Points to area updated with ESM product information

Ry =
Unchanged

Guest condition code is set to 0

Invalid Subcode:

R15 =
8 (Unsuccessful request)

R0 =
X'06' Specification exception indicator

Note: R15 in these cases refers to the caller's R15 as opposed to the guest's R15.

Attributes: Non-MP, dynamic

Linkage: Call with a dynamic save area

Register Usage:

R0
Scratch

R1
Scratch

R2
Scratch

R3
Scratch

R4
Scratch

R5
Address of Rx register

R6
Address of Ry register

R7
Scratch

R8
Scratch

R9
Scratch

R10
Scratch

R11
Address of dispatched VMDBK

R12
Base register

R13
Address of save area

R14
Scratch

R15
Scratch

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Return Codes

When HCPRPD returns to its caller, the return code in Register 15 is handled as shown in Table 79 on page 602.

Table 79. Supported HCPRPD Return Codes

RC	Meaning
0	Processing was successful. Complete the guest instruction.
4	Processing failed due to a condition that would cause a guest program check. Simulate the guest program interruption passed in Register 0.
8	Nullify the guest instruction.
12	Generate a machine check for a storage error, then nullify the instruction. The guest real address of failing storage is in Register 1.
16	Generate a machine check for processing damage, then terminate the instruction.
20	Generate a machine check for storage error, then terminate the instruction. The guest real address of failing storage is in Register 1.
24	Issue an error message or soft abend for a paging I/O error, then nullify the instruction. Register 1 has the message or abend number.

Exit from HCPRPD

Return is made to the calling module by the HCPEXIT macro, located in the HCPPSI macro library. The HCPEXIT macro is coded in HCPRPDEP as:

```
HCPEXIT EP=(HCPRPDEP)          Return to Caller
```

HCPRPE Module for handling DIAGNOSE X'A0'

HCPRPE is a CP module that can be called by an ESM to handle certain DIAGNOSE code X'A0' subcodes and other functions. The following entry points are provided:

Entry Point
Function

HCPRPEEP
To handle DIAGNOSE code X'A0' subcodes X'00', X'08', and X'48'.

HCPRPEPX
To inform CP of the ESM's POSIX capabilities.

HCPRPESG
To inform CP of a change to a user's POSIX Supplementary GID list.

The suggested way for an ESM to call HCPRPE is through HCPRPD, the DIAGNOSE code X'A0' processor (see "HCPRPD Module" on page 599). After the appropriate input parameters are set up, the HCPRPE call can be coded as follows:

```
HCPCALL HCPRPExx
```

where *xx* are the last two letters of the entry point name.

Entry Point HCPRPEEP - Handle DIAGNOSE Code X'A0' subcodes X'00', X'08', and X'48'

If no ESM is installed, HCPRPDEP will call HCPRPEEP to handle subcodes X'00', X'08', and X'48'. If an ESM is installed, it continues to call HCPRPEEP for subcodes X'00' and X'48', but it should handle subcode X'08' in HCPRPDEP. See "HCPRPD Module" on page 599 for additional information.

Entry Point HCPRPEPX - Notify CP of POSIX capabilities

If the ESM contains POSIX support, it should inform CP of this by calling HCPRPEPX; otherwise, CP will not attempt to acquire POSIX database information from the ESM or invoke the ESM to authorize the various POSIX functions. If HCPRPEPX is not invoked, the system will behave as if the ESM returned ACIDEFR on all POSIX-related requests:

- For AUTOLOG, XAUTOLOG and LOGON requests, the POSIX-related information will be obtained from the CP directory. See “LOGON Command” on page 652 and “AUTOLOG and XAUTOLOG Commands” on page 641 for more information about these fields and their use in these requests.
- For POSIX set IDs requests, the entire request is considered to have been deferred to CP. See “POSIX Set ID Functions” on page 685 for more about this type of request.
- For POSIX group database queries, the entire request is considered to have been deferred to CP. See “POSIX Group Database Query Function” on page 686 for more about this type of request.
- For POSIX user database queries, the entire request is considered to have been deferred to CP. See “POSIX User Database Query Function” on page 688 for more about this type of request.

{NGROUPS_MAX} is the maximum number of POSIX Supplementary Group IDs (SGIDs) associated with a single POSIX process or POSIX database entry. If the ESM is providing the POSIX database information for the system, it is permitted to support a different number of SGIDs than CP does when an ESM is not providing the POSIX database information. If this is the case, the ESM must inform CP of its {NGROUPS_MAX} value. The value must be within the valid range supported by CP. The minimum value can be determined from the QPXFCONF function of DIAGNOSE code X'2A0'; the maximum value is 125. The ESM is only permitted to supply the value to CP a single time. It becomes the system's {NGROUPS_MAX} value. for the duration of the CP system IPL. It is recommended that the ESM inform CP of its {NGROUPS_MAX} value during ESM initialization, so that all users have the same capabilities and the system behaves in a POSIX-compliant manner.

For performance and storage utilization reasons, it is recommended that this value be no larger than necessary. The number of SGIDs in the largest SGID list in the database, plus a small margin for growth, would be a reasonable value.

Descriptive Name: Notify CP of POSIX capabilities

Function: Registers that there is an ESM installed that provides support for POSIX functions and the POSIX system databases. Accepts the value of {NGROUPS_MAX} that the ESM supports and makes it the system value.

Input Registers:

R0
{NGROUPS_MAX} value supported by the ESM.

R1
Must contain zero (X'00000000')

R11
Address of the dispatched VMDBK

Output: See Exit Values (Normal and Error)

Note: R15 will be changed for each exit. See each particular exit case, for the registers which contain meaningful information for that case.

Exit Values:

Normal:

R15 = 0
Function Completed

Error:

R15 = 4
The ESM has already informed CP of its capabilities and its {NGROUPS_MAX} value

R15 = 8
The {NGROUPS_MAX} value supplied in R0 is invalid

Attributes: Resident, MP, Reentrant

Linkage: Call with a dynamic save area

Abend Codes: None

Messages None

Responses: None

Wait States: None

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Figure 79. Interface Specifications for the HCPRPEPX Entry Point

Entry Point HCPRPESG - Refresh a user's POSIX SGID list

If a user's POSIX database information is updated such that the POSIX Supplementary GID (SGID) list is affected, the user normally has to LOGOFF and log back on to have the new SGIDs take effect. However, if an ESM wishes to have the change take effect sooner, it may use the following interface to notify CP of the change. The ESM passes the SGIDs to CP, and CP causes them to be in effect for subsequent POSIX programs. Existing POSIX programs are not affected by this action; they continue to execute with their own SGIDs.

There is no need to inform CP of such a change if the affected user is not logged on. For performance and efficiency reasons, the ESM should only inform CP of changes to logged on or disconnected users' SGID lists.

Descriptive Name: Refresh a user's POSIX Supplementary GIDs (SGIDs)

Function: Accepts a notification from the ESM that a user's Supplementary GIDs in the POSIX database have changed. CP makes them take effect for certain new POSIX processes created by the user.

Input Registers:

R0
z/VM user ID of the user whose SGIDs have changed.

R1
This user ID must be specified in upper case and be left-justified and padded with blanks.

R2
Address of a buffer containing the new SGID list. The buffer consists of contiguous four-byte entries, each containing a GID.

R3
Number of SGIDs contained in the buffer pointed to by R2.

R11
Address of dispatched VMDBK.

Output: See Exit Values (Normal and Error)

Note: R15 will be changed for each exit. See each particular exit case, for the registers which contain meaningful information for that case.

Exit Values:

Normal:

R15 = 0
Function Completed

Error:

R15 = 4
The new SGID list contains an invalid number of SGIDs:

- The list contains 0 SGIDs
- The list contains more than {NGROUPS_MAX} SGIDs.

R15 = 8
The input user ID is not logged on.

R15 = 12
A CP error condition was detected that prevented the replacement of the user's SGID list. A soft abend or SNAP dump may have occurred.

Attributes: Resident, MP, Reentrant

Figure 80. Interface Specifications for the HCPRPESG Entry Point (Part 1 of 2)

Linkage: Call with a dynamic save area

Abend Codes: None

Messages None

Responses: None

Wait States: None

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Figure 81. Interface Specifications for the HCPRPESG Entry Point (Part 2 of 2)

HCPRPF Module

CP's version of HCPRPF is a data area, not an executable module. It is empty, nonexecutable, static, and resident. An ESM may replace CP's stub module with its own copy of HCPRPF, which it may use as a data area. CP makes no reference to this module.

HCPRPG Module

CP's version of HCPRPG contains an entry point, HCPRPGPH, and serves only as a place holder. An ESM may replace CP's stub module with its own copy of HCPRPG. CP makes no reference to this module. This module is resident, MP, and dynamic.

HCPRPL Module

CP's version of HCPRPL contains an entry point, HCPRPLPH, and serves only as a place holder. An ESM may replace CP's stub module with its own copy of HCPRPL. CP makes no reference to this module. This module is resident, MP, and dynamic.

HCPRPP Module

CP's version of HCPRPP contains six entry points for ESM single system image (SSI) support: HCPRPPEN, HCPRPPJN, HCPRPPPC, HCPRPPPS, HCPRPPSC, and HCPRPPWC. It also contains the following local entry points: HCPRPPHR, HCPRPPJD, HCPRPPJR, and HCPRPPHS.

Entry Point HCPRPPEN

This entry point is used for single system image enablement processing. HCPRPPEN issues:

```
HCPSOCK CREATE_PORT,PORT='*RPI',RETSID=(R),EP=HCPRPPHR
```

This establishes the socket and stacks a call to the local entry point for the code that handles incoming messages, HCPRPPHR. The returned socket id is stored for subsequent use in the table HCPRWAPP mapped by the RWAESM DSECT.

The portname *RPI should be used by all HCPSOCK macros in HCPRPP.

Entry Point HCPRPPJN

This entry point is used for single system image join processing. HCPRPPJN stacks a deferred call to HCPRPPJD and returns with RC=4.

Entry Point HCPRPPPC

CP calls this entry point to handle an upgrade to the single system image level. The CP stub entry point is not used.

Entry Point HCPRPPPS

CP calls this entry point when a single system image mode change to STABLE is about to occur. The CP stub entry point is not used.

Entry Point HCPRPPWC

CP calls this entry point to perform local worthiness checks. When omitted, the service is assumed to be single system image-worthy. The CP stub entry point is not used.

Entry Point HCPRPPSC

CP calls this entry point when a single system image state change occurs. The CP stub entry point is not used.

SSI join time processing

A deferred call to HCPRPPJD is stacked by HCPRPPJN after it is called by the single system image Configuration Manager at SSI join time. HCPRPPJD compares this system's installed ESM/level (if any) with that of the already joined nodes. If any compare is incompatible the result is PXMREJCT. This code works in conjunction with the table at EP HCPRWAPP, mapped by DSECT RWAESM in ACIPARMS COPY. An ESM should provide updates to HCPRWA to ensure that the SSI join time checks use the correct product and version information. The comparison data from the already joined nodes in the SSI cluster is provided by local entry point HCPRPPJR. Communication between nodes is by the HCPSOCK service, and local entry point HCPRPPHS provides a HCPSOCK SEND_REPLY service.

HCPRWA Module

CP's version of HCPRWA contains an entry point, HCPRWACP, which is empty, nonexecutable, static, and resident. An ESM may replace CP's stub module with its own copy of HCPRWA, which it may use as a work area. CP makes no reference to this module.

CP Callable Services for the ACI

The following CP services are intended to be called by any of the ACI modules. The use of such services is optional.

Entry Point HCPPWAPF

If the ESM supports optional features, such as the use of password phrases or mixed-case passwords, it should inform CP by calling HCPPWAPF. By default, there is no ESM support for password phrases or mixed-case passwords. When calling HCPPWAPF the ESM must indicate the level of support for optional features in the low order byte of register 1.

Figure 82 on page 608 lists the interface specifications for this entry point.

Input Registers:

R1
A four-byte parameter containing:

X'00000001' - The ESM supports the extended form of the LOGON ACI call. For more information, refer to the notes for the "LOGON Command" on page 652.

R11
Address of the dispatched VMDBK

Output:

R15
See Exit Values (Normal and Error)

Exit Values:

Normal:

R15 = 0
Optional features support flag updated

Error:

R15 = 4
Optional features support flag not updated

Attributes: MP, dynamic, resident

Linkage: Call with a dynamic save area

Figure 82. Interface Specifications for the HCPPWAPF Entry Point

Summary of CP Modules and Entry Points

The following table summarizes the module and entry point function information given on pages "HCPRPI Module" on page 593 through "HCPRPG Module" on page 606:

Module	Entry Point	Function/Description
HCPRPI	HCPRPICN	IUCV Connect interrupt handler
	HCPRPIIL	IUCV message pending interrupt handler
	HCPRPISV	IUCV Sever interrupt handler
	HCPRPIQS	IUCV Quiesce interrupt handler
	HCPRPIRM	IUCV Resume interrupt handler
	HCPRPIRA	Defined to support authorization requests (calls) from CP to the ESM. For commands, DIAGNOSE codes, and a subset of system functions, it is called to check whether a user is authorized prior to performing the requested function.
HCPRPW	HCPRPWEP	Defined to support password authorization checking on the LOGON, AUTOLOG, and XAUTOLOG commands (checked prior to processing the command).
HCPRPD	HCPRPDEP	Functions as a DIAGNOSE code X'A0' processor. This entry point calls HCPRPE at entry point HCPRPEEP, which processes subcodes X'00', X'08', and X'48'.

Module	Entry Point	Function/Description
HCPRPE	HCPRPEEP	Handle DIAGNOSE code X'A0' subcodes X'00', X'08', and X'48'
	HCPRPEPX	Notify CP of POSIX capabilities
	HCPRPESG	Refresh a user's POSIX SGID list.
HCPRPF	(none)	Functions as a data area only, not an executable module.
HCPRPG	HCPRPGPH	Serves only as a place holder.
HCPRPL	HCPRPLPH	Serves only as a place holder.
HCPRWA	HCPRWA	This module is nonexecutable, static, and resident.
	HCPRWAPP	The entry point of the ESM product information table, used by HCPRPEEP, HCPRPPEN, HCPRPPJD, HCPRPPJR and HCPRPPHS.
HCPPWA	HCPPWAPF	Called by the ESM to indicate to CP the level of support provided for optional features.

ACI Security Bits

The term, *event*, will be used to refer to all CP commands, all CP DIAGNOSE codes, and selected 'system functions' which have associated ACI Security bits. These selected system functions are listed below:

APPCCON
APPC connect

APPCPWVL
APPC connect with password

APPCSEV
APPC sever

DIRECTORY_CMD
CP command issued from user directory

IUCVCON
IUCV connect

IUCVSEV
IUCV sever

MAINTCCW
Maintenance CCW

MDISK
MDISK and LINK-to-self

RSTDSEG
Load/find a restricted NSS or DCSS

SDF_CREATE
SDF file create

SDF_DELETE
SDF file delete

SDF_OPEN
SDF file open

SNIFFER_MODE
Guest LAN Sniffing ON/OFF

SPF_CREATE
Spool file create

SPF_DELETE
Spool file delete

SPF_OPEN
Spool file open

UTLPRINT
CP PRINT function

There are three different types of security bits:

- AUDIT
- PROTECT(DAC)
- MAC (Mandatory Access Control).

These bits control the calls that CP makes to the ESM. PROTECT and MAC checks are types of authorization control. AUDITing consists of logging a record to show that the event was issued.

Setting the ACI Security Bits

CP provides the initial settings of the security bits; all AUDIT and MAC bits are initialized to off. The following is the list of those events for which the PROTECT bit is initialized to ON.

AUTOLOG
DIAGNOSE code X'D4'
DIAGNOSE code X'E4'
DIAGNOSE code X'280'
LINK
LOGOFF
LOGON
TAG
TRANSFER (see the note below)
TRSOURCE
XAUTOLOG

Note: If the PROTECT Bit is on for TRANSFER, then the ACI is called for CHANGE, CLOSE, SPOOL, TRSAVE, VMDUMP, and DIAGNOSE code X'94', if that event is used to change the ownership of a spool file, such as when the TO option is specified. The format of ACIPARMS on these calls is similar to the format used on the call for the TRANSFER command.

If an ESM is installed, it may set the security bits by invoking HCPDA0 (see “HCPDA0 Module for Updating ACI Security Bits” on page 612). CP enforces restrictions on setting the ACI Security bits. The AUDIT bit may be set for all events; the PROTECT and MAC bits can be set only for a predefined subset of events. The AUDIT and PROTECT bits can be set independently on an individual event basis. The MAC bits, however can only be set either all on or all off.

The following is the list of those events for which the PROTECT bit can be enabled or disabled.

APPC connect with password validation
ATTACH
COUPLE
DEDICATE directory statement processing
DIAGNOSE code X'A0'
DIAGNOSE code X'E4'
DIAGNOSE code X'D4'
DIAGNOSE code X'88'
DIAGNOSE code X'280'
DIAGNOSE code X'290'
FOR
GIVE
LINK
MDISK

RSTDSEG
STORE version C
TAG
TRANSFER
TRSOURCE

The following is the list of events for which the MAC bit can be enabled or disabled.

APPC Connect
AUTOLOG
CHANGE
COUPLE
DIAGNOSE code X'14'
DIAGNOSE code X'68'
DIAGNOSE code X'BC'
DIAGNOSE code X'D4'
DIAGNOSE code X'23C'
DIAGNOSE code X'290'
IUCV Connect
LINK
LOGON
MDISK
MESSAGE ('ANY' version), MSGNOH, SMSG, WARNING
QUERY RDR/PRT/PUN
QUERY TAG
QUERY TRFILES
RSTDSEG
SDF_OPEN
SPF_OPEN
START
TAG
TRSOURCE
UTLPRINT
XAUTOLOG

Checking the Security Bits and Calling the ESM

Each time CP processes an event, it checks the associated ACI Security bits. If any are on, CP sets up an ACIPARMS parameter list and calls the ESM, passing the address of ACIPARMS in General Register 1. See “CP Calls to the ACI” on page 639 for the parameter lists on the CP calls to the ACI. The ACIBMAPA, ACIBMAPP, and ACIBMAPM fields indicate which security bits were enabled. These three fields contain respectively the AUDIT, PROTECT, and MAC setting for the event executing. For DIAGNOSEs, system functions, and 'ANY' class commands, the high order (X'80') bit in the ACIBMAPx fields contains the security setting. For privileged commands, the ACIBMAPx fields contain settings for privilege classes A through G of the command (X'80' - X'20').

Upon return from the ESM, the return code in the ACICODE field is checked. The supported ESM return codes for each call are documented in “CP Calls to the ACI” on page 639. In general, the return code handling is as follows:

There are four main ESM return codes defined in the ACIPARMS control block:

ACIAUTH 0
Authorization is granted

ACIDEFR 4
ESM is not there or defers

ACINOAC 8
Authorization is denied

ACIUNAV 20
ESM is not available (could not complete function)

Notes:

1. Audit-only calls to the ESM support the ACIAUTH, ACIDEFR and ACIUNAV return codes.
2. ACIAUTH and ACIDEFR return codes are handled the same: processing continues.
3. If ACINOAC is received, the event will fail with an error message, or condition applicable to the event which was issued.
4. If ACIUNAV is returned then the event will not be allowed. Most commands fail with message:

   ```
   6525E    The ESM is unavailable
   ```

   Diagnose codes and system functions end with various failing return conditions.

LOGON, AUTOLOG, XAUTOLOG POSIX SET ID, POSIX GROUP DATABASE QUERY, and POSIX USER DATABASE QUERY ESM calls recognize additional return codes. These are documented in "CP Calls to the ACI" on page 639.

HCPDA0 Module for Updating ACI Security Bits

HCPDA0 is a CP module which can be called by an ESM to control the ACI Security bit settings for CP commands, diagnose codes and selected system functions. Three entry points are provided for ACI Security bit manipulation:

Entry Point
Function

HCPDA0RL
To return the general ACI Security bit settings

HCPDA0UL
To update general ACI AUDIT and PROTECT bits

HCPDA0MC
To update the ACI MAC bits

The suggested way for an ESM to call HCPDA0 is through HCPRPD, the DIAGNOSE code X'A0' processor (see "HCPRPD Module" on page 599). After the appropriate input parameters are set up, the HCPDA0 call can be coded as follow:

```
HCPCALL HCPDA0xx
```

where *xx* are the last two letters of the entry point name.

Entry Point HCPDA0RL - Return ACI Security Bit Settings

Figure 83 on page 613 lists the interface specifications for the HCPDA0RL entry point. To determine the buffer size needed for the return data, this entry point can be called with a buffer size of zero. Upon return, general register zero will contain the number of entries to be returned by HCPDA0RL. Each returned entry will be mapped by an HCPA0LBK of size: A0LSIZE. (See "HCPA0LBK Control Block" on page 617 for format of HCPA0LBK. The required buffer size can then be calculated, and HCPDA0RL can be called again with the correct buffer size specified.

Descriptive Name: Return General ACI Security bits

Function: Returns to caller a list of all CP commands, diagnose codes, and some system functions, with the associated system security data.

Input Registers:

R1
Guest real address of buffer in which to return data.

The data is returned as a list of entries mapped by HCPA0LBK, one for each command, diagnose codes, and system function.

R2
Size of Buffer

R11
Address of the dispatched VMDBK

Output: See Exit Values (Normal and Error)

Note: R0, R1, R2, and R15 will be changed for each exit. See each particular exit case, for the registers which contain meaningful information for that case.

Exit Values:

Normal:

R15 = 0
Function Completed

R1 =
The number of entries that were processed.

Error:

R15 = 4
The size of the buffer was not large enough to hold all the entries.

R0 =
The number of entries that remain (i.e. could not fit into the buffer).

R1 =
The number of entries that were processed.

R15 = 8
Processing failed due to a condition which would cause a guest program check, or guest program exception.

R0 =
Program exception code representing the error. A negative value indicates instruction nullification is required.

R15 = 12
Processing failed due to a host paging or storage error.

Attributes: MP, Reentrant

Figure 83. Interface Specifications for the HCPDA0RL Entry Point (Part 1 of 2)

Linkage: Call with a dynamic save area

Abend Codes: None

Messages None

Responses: None

Wait States: None

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Figure 84. Interface Specifications for the HCPDA0RL Entry Point (Part 2 of 2)

Entry Point HCPDA0UL - Update ACI AUDIT and PROTECT Bits

Following are the interface specifications for the HCPDA0UL entry point.

Descriptive Name: Update AUDIT and PROTECT Security bits

Function: Update general audit and protect bits for all CP commands, DIAGNOSE codes, and security-relevant system functions.

Input Registers:

R1
Guest real address which contains the input entries, mapped by HCPA0UBK.

R2
Number of entries in the input buffer

R11
Address of the dispatched VMDBK

Output: See Exit Values (Normal and Error)

Note: R0, R1, R2, and R15 will be changed for all Normal and Error exits. See each particular exit case for the registers which contain meaningful information for that case.

Exit Values:

Normal:

R15 = 0
Function Completed

Error:

R15 = 4
Security data was not updated because of an error with one or more of the input A0UBK entries. 'A0UECODE' has been set for all entries. If A0UECODE equals A0UNOERR then there was no error with that entry. Otherwise, the appropriate error code (as defined in the HCPA0UBK) has been stored in A0UECODE.

R1 =
The number of entries that were incorrect.

R15 = 8
Processing failed due to a condition which would cause a guest program check, or guest program exception.

R0 =
Program exception code representing the error. A negative value indicates instruction nullification is required.

R15 = 12
Processing failed due to a host paging or storage error.

Attributes: MP, Reentrant

Figure 85. Interface Specifications for the HCPDA0UL Entry Point (Part 1 of 2)

Linkage: Call with a dynamic save area

Abend Codes: None

Messages None

Responses: None

Wait States: None

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

For commands whose protection setting (CMDPROT in the HCPCMDBK control block) can not be changed (A0LCPROT=OFF in the HCPA0LBK or CMDVPROT=OFF in the HCPCMDBK control block), the AOUBK entry must have A0UPROT set to off. For commands whose "valid before logon" setting (A0LCLOGN=OFF in the HCPA0LBK or CMDXLOG=OFF in the HCPCMDBK control block), the A0UBK entry must have A0ULOGON set to off.

Figure 86. Interface Specifications for the HCPDA0UL Entry Point (Part 2 of 2)

Entry Point HCPDA0MC - Update ACI MAC Bits

Following are the interface specifications for the HCPDA0MC entry point.

Descriptive Name: Update ACI Security MAC bits

Function: Update MAC security bits for CP commands, diagnose codes, and system functions. MAC bits will be updated only for those events which are defined to have MAC protection, and they will be updated to be either all on or all off.

Input Registers:

R3
Address of 1 byte parameter list in the following format:

X'80' - Enable MAC
X'40' - Disable MAC

Either Enable_MAC or Disable_MAC must be set, and both may not be set.

R11
Address of the dispatched VMDBK

Output: See Exit Values (Normal and Error)

Exit Values:

Normal:

R15 = 0
Function Completed

Error:

R15 = 8
Request-type parameter was not set correctly (see Input Registers), no processing was done.

Attributes: MP, Reentrant

Linkage: Call with a dynamic save area

Abend Codes: None

Messages None

Responses: None

Wait States: None

General Notes: Registers R11 and R13 should not be changed by the installation-supplied code.

Figure 87. Interface Specifications for the HCPDA0MC Entry Point

HCPA0LBK Control Block

The following table gives the format of information in the HCPA0LBK control block.

Table 80. Format of information in the HCPA0LBK control block

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
0	(00)	STRUCTURE		A0LBK	Describes each entry
0	(00)	CHARACTER	12	A0LNAME	Command, diagnose, or system function name

Table 80. Format of information in the HCPA0LBK control block (continued)

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
12	(0C)	CHARACTER	12	A0LCMDOP	Command operand. Valid only if entry type (i.e. A0LENTYP) is A0LCPCMD, A0LSTCMD, or A0LQYCMD and A0LOPRND is on
24	(18)	BITSTRING	1	A0LIBMCL	IBM Class. Valid only if entry type (i.e. A0LENTYP) is A0LCPCMD, A0LSTCMD, or A0LQYCMD.
CODES DEFINED IN A0LIBMCL:					
		`1... ....`		A0LIBMA	X'80' IBM Class A
		`.1.. ....`		A0LIBMB	X'40' IBM Class B
		`..1. ....`		A0LIBMC	X'20' IBM Class C
		`...1 ....`		A0LIBMD	X'10' IBM Class D
		`.... 1...`		A0LIBME	X'08' IBM Class E
		`.... .1..`		A0LIBMF	X'04' IBM Class F
		`.... ..1.`		A0LIBMG	X'02' IBM Class G
		`.... ...1`		A0LIBMH	X'01' IBM Class H
		`1111 1111`		A0LCLANY	X'FF' IBM Class 'ANY'
25	(19)	BITSTRING	1	A0LENTYP	Entry type
CODES DEFINED IN A0LENTYP:					
		`.... ...1`		A0LCPCMD	X'01' CP command
		`.... ..1.`		A0LSTCMD	X'02' CP SET command
		`.... ..11`		A0LQYCMD	X'03' CP QUERY command
		`.... .1..`		A0LDIAG	X'04' Diagnose code Constants 6-8 are reserved

Table 80. Format of information in the HCPA0LBK control block (continued)

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.... 1..1		A0LSYSFN	X'09' System function
26	(1A)	BITSTRING	1	A0LENTFL	Entry flags
BITS DEFINED IN A0LENTFL:					
		1...		A0LOPRND	X'80' Command operand field is valid
27	(1B)	BITSTRING	1		Reserved for future IBM use
28	(1C)	BITSTRING	1	A0LATTRI	Entry attributes
BITS DEFINED IN A0LATTRI:					
		1...		A0LCLOGN	X'80' Command can be disabled prior to logon. Valid only if entry type (i.e. A0LENTYP) is A0LCPCMD, A0LSTCMD, or A0LQYCMD
		.1..		A0LCPROT	X'40' Command, diagnose, or system function's protection setting can be changed by the ESM
		....		A0LCMAC	X'10' Command, diagnose, or system function has MAC capability
29	(1D)	BITSTRING	1		Reserved for future IBM use
30	(1E)	BITSTRING	1	A0LSECUR	Security flags
BITS DEFINED IN A0LSECUR:					
		1...		A0LLOGON	X'80' Command is valid prior to logon
		.1..		A0LAUDIT	X'40' Command, diagnose, or system function is being audited
		..1.		A0LPROT	X'20' Command, diagnose or system function is being protected
		...1		A0LMAC	X'10' Command, diagnose or system function is being MAC'd
31	(1F)	BITSTRING	33		Reserved for future IBM use
		.1..		A0LNEXT	"*" Next entry
		.... 1...		A0LSIZE	"(*-A0LBK+7)/8"Size of A0LBK in double words

Table 80. Format of information in the HCPA0LBK control block (continued)

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.1..		A0LBSIZE	"*-A0LBK" Size of A0LBK in bytes

HCPA0UBK Control Block

The following table gives the format of information in the HCPA0UBK control block.

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
0	(00)	STRUCTURE		A0UBK	Describes each entry
0	(00)	CHARACTER	12	A0UNAME	Command, diagnose, or system function name
12	(0C)	CHARACTER	12	A0UCMDOP	Command operand. Valid only if entry type (i.e. A0UENTYP) is A0UCPCMD, A0USTCMD, or A0UQYCMD and A0UOPRND is on
24	(18)	BITSTRING	1	A0UIBMCL	IBM Class. Valid only if entry type (i.e. A0UENTYP) is A0UCPCMD, A0USTCMD, or A0UQYCMD.
CODES DEFINED IN A0UIBMCL:					
		1...		A0UIBMA	X'80' IBM Class A
		.1..		A0UIBMB	X'40' IBM Class B
		..1.		A0UIBMC	X'20' IBM Class C
		...1		A0UIBMD	X'10' IBM Class D
		.... 1...		A0UIBME	X'08' IBM Class E
		.... .1..		A0UIBMF	X'04' IBM Class F
		.... ..1.		A0UIBMG	X'02' IBM Class G
		1		A0UIBMH	X'01' IBM Class H
		1111 1111		A0UCLANY	X'FF' IBM Class 'ANY'

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
25	(19)	BITSTRING	1	A0UENTYP	Entry type
CODES DEFINED IN A0UENTYP:					
		1		A0UCPCMD	X'01' CP command
		.... ..1.		A0USTCMD	X'02' CP SET command
		11		A0UQYCMD	X'03' CP QUERY command
		.... .1..		A0UDIAG	X'04' Diagnose code Constants 6-8 are reserved
		.... 1..1		A0USYSFN	X'09' System function
26	(1A)	BITSTRING	1	A0UENTFL	Entry flags
BITS DEFINED IN A0UENTFL:					
		1...		A0UOPRND	X'80' Command operand field is valid
27	(1B)	BITSTRING	1		Reserved for future IBM use
28	(1C)	SIGNED	2	A0UECODE	Error code
CODE DEFINITIONS FOR A0UECODE:					
		....		A0UNOERR	"000" Entry is o.k.
		1		A0UCMDNF	"001" Unknown CP command
		11		A0UINVLD	"003" Invalid entry
		.11. .1..		A0UCNOPR	"100" Command cannot be protected
		.11. .1.1		A0UNOLOG	"101" Command cannot be disabled prior to logon
		.11. .11.		A0UIBMNF	"102" Command not found for specified IBM class or 'A0UIBMCL' is invalid
		.11. .111		A0UOPRNG	"103" 'A0UOPRND' not supported for entry
		.11. 1...		A0UOPRMS	"104" 'A0UCMDOP' is missing

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.11. 1..1		A0UNOSET	"105" 'CP SET' command is not supported
		.11. 1.1.		A0UNOQUY	"106" 'CP QUERY' command is not supported
		11.. 1...		A0UDGINV	"200" Diagnose name is invalid
		11.. 1..1		A0UNOTM4	"201" Diagnose code is not a multiple of 4
		11.. 1.1.		A0UDGND	"202" Diagnose code is not available (i.e. defined)
		11.. 1.11		A0UDNOPR	"203" Diagnose cannot be protected
		SIGNED		A0USYSNS	"300" System function not supported or unknown
		SIGNED		A0USNOPR	"301" System function cannot be protected
30	(1E)	BITSTRING	1	A0USECUR	Flags -
BITS DEFINED IN A0USECUR:					
		1...		A0ULOGON	X'80' Command should be valid prior to logon
		.1..		A0UAUDIT	X'40' Command, diagnose, or system function should be audited
		..1.		A0UPROT	X'20' Command, diagnose or system function should be protected
		...1		A0UMAC	X'10' Command, diagnose or system function should be MAC'd
31	(1F)	BITSTRING	33		Reserved for future IBM use
		.1..		A0UNEXT	"*" Next entry
		.... 1...		A0USIZE	"(*-A0UBK+7)/8"Size of A0UBK in double words
		.1..		A0UBSIZE	"*-A0UBK" Size of A0UBK in bytes

ACIPARMS Control Block

The following table gives the format of information in the ACIPARMS control block.

Note: All fields, codes, and flags defined in the ACIPARMS parameter list that are not specifically defined are reserved and are not to be used by an ESM for other purposes.

Offsets

Dec	Hex	Type	Len	Name (Dim)	Description
0	(0)	STRUCTURE		ACIPARMS	
0	(0)	BITSTRING	1	ACIFCN	A 1 function request

CODES DEFINED IN ACIFCN:

The code in this field indicates class of event for the purposes of the authorization check associated with the ACIBMAPP setting. On calls where no authorization corresponding to the ACIBMAPP setting is applicable, ACIVMCMD should be used.

`.... ....`	ACILINK	X'00' Link access validation
`.... .1..`	ACISPOOL	X'04' Spool access validation
`.... 1...`	ACITAG	X'08' Node access validation
`.... 11..`	ACIDEL	X'0C' Delete use request
`...1 ....`	ACILOG	X'10' LOGON and (X)Autolog
`...1 1...`	ACIALTU	X'18' Alternate user ID
`...1 11..`	ACIVMCMD	X'1C' COMMAND function
`..1. ....`	ACINSSEG	X'20' Restricted Load/Find of an NSS
`..1. .1..`	ACIDCSEG	X'24' Restricted Load/Find of a DCSS
`..1. 1...`	ACIFESM1	X'28' Reserved for ESM internal use
`..1. 11..`	ACIFESM2	X'2C' Reserved for ESM internal use
`..11 ....`	ACISETID	X'30' POSIX set ID request
`..11 .1..`	ACIQGDB	X'34' POSIX group database query request
`..11 1...`	ACIQUDB	X'38' POSIX user database query request

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		`..11 11..`		ACIRSCHK	X'3C' Resource access check
1	(1)	BITSTRING	1	ACICODE	A 2 return code field

CODES DEFINED IN ACICODE:

Note: The values X'2C' and X'28' are reserved for ESM (External Security Manager) use in the ACIFCN field. When adding new function codes, do not use these values.

		`.... ....`		ACIAUTH	X'00' Access authorized, or successful audit
		`.... .1..`		ACIDEFR	X'04' Access deferred
		`.... 1...`		ACINOAC	X'08' Access denied

The following two return codes are defined only for LOGON (ACIFCN=ACILOG) calls.

		`.... 11..`		ACIFAIL	X'0C' Fail command and issue no message. For the LOGON command, force user. Only possible when ACIFCN=ACILOG.
		`...1 ....`		ACITERM	X'10' Fail command and issue no message. Only possible when ACIFCN=ACILOG.
		`..1. ....`		ACIUNAV	X'20' ESM was unable to process request
		`..1. .1..`		ACINFND	X'24' Input group or user does not exist
		`..1. 1...`		ACIBFSM	X'28' Access authorized, but the data buffer is too small to contain the requested data
		`..1. 11..`		ACINVAL	X'2C' ACINUID or ACINGID contains an invalid value
		`..11 ....`		ACINGMEM	X'30' ACITUSR not a member of the group identified in ACIGGNAM or ACIGGID. Only possible when ACIPXQGM=1.
2	(2)	BITSTRING	2	ACILEN	ACIPARMS length, in bytes

The ACIBMAPA, ACIBMAPP and ACIBMAPM fields contain the audit, protect and MAC settings respectively. For privileged commands, the bits in the field contain the settings for CLASSA-CLASSG of the command. For ANY class commands, diagnose codes, and system functions, the flags defined after each field are used to indicate the setting. (ACIBMAPM is located after ACIENV)

4	(4)	BITSTRING	1	ACIBMAPA	ACI audit bit

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		1...		ACIANYAU	X'80' Audit for 'ANY' command
		1...		ACIDGNAU	X'80' Audit for diagnose
		1...		ACISYSAU	X'80' Audit for system functions
5	(5)	BITSTRING	1	ACIBMAPP	ACI protect bit
		1...		ACIANYPR	X'80' Protect for 'ANY' command
		1...		ACIDGNPR	X'80' Protect for diagnose
		1...		ACISYSPR	X'80' Protect for system functions
6	(6)	BITSTRING	1	ACIENV	A 3 Type of event

BITS DEFINED IN ACIENV:

The ACIENV codes indicate the format of the ACIPARMS buffer being passed.

....1	ACIXAC	X'01' Command
.... ..11	COUPLEN	X'03' Network COUPLE
.... .1..	ACISNIF	X'04' Audit promiscuous mode
.... .11.	ACIXAD	X'06' Diagnose
.... 1.11	ACIXAT	X'0B' TRANSFER function
.... 11..	ACIXAV	X'0C' VMCF function
.... 1111	ACIXACCW	X'0F' Diagnostic CCW function
...1	ACISEG	X'10' DIAGNOSE X'64' and RSTDSEG.
...1 ...1	ACIDIAE4	X'11' DIAGNOSE X'E4'
...1 ..1.	ACIAPPW	X'12' APPC Connect with password

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		...1 ..11		ACICNCT	X'13' IUCV & APPC connect
		...1 .1..		ACISEVER	X'14' IUCV & APPC sever
		...1 .1.1		ACISPF	X'15' SPF/SDF create, open & delete and CP printing
		..11		ACIAUTO	X'30' (X)autolog
		..11 ...1		ACITAGN	X'31' Tag command
		..11 ..1.		ACIDISK	X'32' Link or MDISK
		..11 ..11		ACIGVRTN	X'33' GIVE return
		..11 .111		ACITRANS	X'37' Transfer command & 'TO' options
		..11 1...		ACISPTAP	X'38' SPXTAPE LOAD or DUMP
		.1..		ACIPQUDB	X'40' POSIX user database query
		.1.. ...1		ACIPQGDB	X'41' POSIX group database query
		.1.. ..1.		ACISETUI	X'42' POSIX setuid ( ) function
		.1.. ..11		ACISETEU	X'43' POSIX seteuid ( ) function
		.1.. .1..		ACISETGI	X'44' POSIX setgid ( ) function
		.1.. .1.1		ACISETEG	X'45' POSIX setegid ( ) function
		.1.. .11.		ACIEXEC	X'46' POSIX exec ( ) function
		.1.. .111		ACINWGRP	X'47' POSIX newgrp ( ) function
		.1.. .111		ACINWGRP	X'47' POSIX newgrp utility

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.1.. 1...		ACISETSG	X'48' POSIX setgroups () function
		1111 111.		ACICPAUD	X'FE' CP-generated audit
7	(7)	BITSTRING	1	ACIBMAPM	ACI MAC bits
		1...		ACIANYMC	X'80' MAC for 'ANY' command
		1...		ACIDGNMC	X'80' MAC for Diagnose
		1...		ACISYSMC	X'80' MAC for system functions
8	(8)	CHARACTER	8	ACIRGRP	Requester's group name
16	(10)	CHARACTER	8	ACIRUSR	Requester's user ID
24	(18)	CHARACTER	8	ACITGRP	Target group name
32	(20)	CHARACTER	8	ACITUSR	Target user ID
40	(28)	CHARACTER	2	ACIMODE	Access mode
42	(2A)	CHARACTER	4	ACIADDR	Resource address
46	(2E)	BITSTRING	1	ACILGOPT	Logon options

FLAGS DEFINED IN ACILGOPT

These flags are defined for calls with ACIFCN=ACILOG

		1...		ACINOPAS	X'80' Don't check the password
		.1..		ACIRECON	X'40' This is a reconnect
		..1.		ACINPMT	X'20' Don't prompt for password
		...1		ACIVAL	X'10' Only check the password
		.... 1...		ACILOGCL	X'08' Logical terminal
		.... .1..		ACILOCAL	X'04' Local terminal
		.... ..1.		ACISNA	X'02' SNA (VTAM) terminal

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.... ..1.		ACILOGIP	X'02' Logical terminal with IP address
		1		ACITTY	X'01' TTY terminal (ASCII)
46	(2E)	BITSTRING	1	ACILFLAG	VMLAN specific flags

FLAGS DEFINED IN ACILFLAG

These flags are defined for calls with ACIFCN=ACIVMCMD

		1...		ACILSNIF	X'80' Promiscuous mode is authorized
		.1..		ACILSON	X'40' Audit promiscuous mode on h
		..1.		ACILSOFF	X'20' Audit promiscuous mode off

ACIVERS indicates the level of the ACIPARMS interface.

ACIVERS should be set to ACIVERS1 on all ESM calls.

47	(2F)	BITSTRING	1	ACIVERS	ACIPARMS version indicator

BITS DEFINED IN ACIVERS:

		1...		ACIVERS1	X'80' Version 1 indicator
48	(30)	CHARACTER	8	ACINODE	Resource nodename
56	(38)	CHARACTER	8	ACILABL	Volume label
64	(40)	CHARACTER	4	ACITADDR	Target resource address

BITS DEFINED IN ACIFLAG:

68	(44)	BITSTRING	1	ACIFLAG	A 4 success/failure indicator
		1...		ACIGOOD	X'80' The event was a success. The definition of success is based on CP's authority checking for the event, not on the execution of the event. ACIGOOD indicates that CP has authorized the user to issue the command, diagnose or system function.

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.1..		ACIBAD	X'40' The event was a failure. ACIBAD indicates that CP denies authority for the user to issue the command, diagnose code, or system function. (CP is going to fail the request). This flag is normally used on audit- only requests.
		..1.		ACIANY	X'20' Diagnose code, system function or Class 'ANY' command
		.... .1..		ACIRO	X'04' Read Only MAC call
		.... ..1.		ACIWO	X'02' Write Only MAC call
		1		ACIRW	X'01' Read/Write MAC call
69	(45)	BITSTRING	1	ACICMDTP	A 5 command version
70	(46)	BITSTRING	1	ACIFLAG2	Processing flags
BITS DEFINED IN ACIFLAG2:					
		1		ACIMSGUS	X'01' Indicates the error message should only be issued to the user
		1		ACIPXQGM	X'01' Indicates this is the query to determine if ACITUSR is a member of the group identified in ACIGGNAM or ACIGGID. Redefined following bits in ACIFLAG2 for CP use only
		.... ..1.		ACIPXIDS	X'02' Indicates the query was initiated by a POSIX process. ACIxEUID and ACIxEGID are filled in.
		.... .1..		ACIPXSYS	X'04' Indicates this is a CP-initiated query (as opposed to a user-initiated query). No authority checking should be performed.
		.... 1...		ACIPXUSN	X'08' Indicates supplying a user or group name in ACIxxNAM (as opposed to a UID or GID in ACIxxID)
		...1		ACIPXALL	X'10' Indicates real, effective and saved set IDs are to be set (0 value indicates only effective ID is to be set)
		..1.		ACINQLUN	X'20' The SNA logical unit identifier defined in ACITRMID has a network qualifier, or the logical device identifier defined in ACINWQFR

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.1..		ACINPASS	X'40' User defined with NOPASS operand in system directory
		1...		ACIRMSRC	X'80' A private server is being xautologged by a remote source
71	(47)	BITSTRING	1	ACIRSNCD	SECLABEL relationship - for MDISK
		..1.		ACISD	X'20' Subject dominates object
72	(48)	CHARACTER	12	ACIEVENT	Event name
84	(54)	SIGNED	4	ACIVMDBK	Address of the VMDBK for a user who is being autologged or logged on
88	(58)	CHARACTER	8	ACIBYVAL	BYUSER ID
96	(60)	CHARACTER	8	ACISLAB	Source SECLABEL
104	(68)	CHARACTER	8	ACITLAB	Target SECLABEL
112	(70)	CHARACTER	8	ACIALAB	Alternate SECLABEL
120	(78)	SIGNED	4	ACIDATA (0)	Start of variable length data
		EXPRESSION		ACISIZE	"(ACIDATA-ACIPARMS+7)/8" ACIPARMS size in double word
120	(78)	CHARACTER	40	ACIPASS	LOGON passwords
		EXPRESSION		ACIPWSZ	"(*-ACIDATA+7)/8" Size of ACIPASS
160	(A0)	CHARACTER	8	ACINWQFR	The network qualifier of the SNA logical unit identifier of the logical device identifier provided in ACITRMID
168	(A8)	BITSTRING	1	ACILOPTS	LOGON/(X)AUTOLOG option settings
BITS DEFINED IN ACILOPTS:					
		1...		ACILSIDE	X'80' If the bit is on, it indicates the option settings are provided by the ESM. Otherwise, the options settings are deferred to CP and the ACILSIDA field is unused.
		.1..		ACILSIDA	X'40' Indicates the user is allowed to set other user's POSIX IDs
		..1.		ACICHGPW	X'20' Request for password change.

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		...1		ACIUSEPP	X'10' Password phrase is in ACIPASSP.
		.... 1...		ACIPPLGO	X'08' Password phrase entered in the password field of the system logon screen.
120	(78)	BITSTRING	1	ACIPWLEN	Password length
121	(79)	CHARACTER	39	ACIPSWD	Password
120	(78)	BITSTRING	1	ACIOPWL	Current password length
121	(79)	CHARACTER	8	ACIOPSWD	Current password
129	(81)	BITSTRING	1	ACINPWL	New password length
130	(82)	CHARACTER	8	ACINPSWD	New password
208	(D0)	CHARACTER	604	ACIPASSA	Password phrase area
208	(D0)	SIGNED	2	ACIOPPLN	Password phrase length
210	(D2)	CHARACTER	200	ACIOPP	Old password phrase
410	(19A)	SIGNED	2	ACINPPLN	New password phrase length
412	(19C)	CHARACTER	200	ACINPP	New password phrase
48	(30)	CHARACTER	8	ACITRMID	Terminal ID
56	(38)	CHARACTER	8	ACIDSPID	Dispatched userid
24	(18)	CHARACTER	8	ACIWUSR	Working (target) user ID
32	(20)	CHARACTER	8	ACIAUSR	Alternate user ID assigned
48	(30)	CHARACTER	8	ACINSPLD	New SECLABEL
120	(78)	BITSTRING	8	ACISEGNM	EBCDIC shared segment name
120	(78)	SIGNED	4	ACIRX	Contents of the Rx register
124	(7C)	SIGNED	4	ACIRX1	Contents of the Rx+1 register
128	(80)	SIGNED	4	ACIRY	Contents of the Ry register
132	(84)	SIGNED	4	ACIRY1	Contents of the Ry+1 register
		EXPRESSION		ACIDISZ	"(*-ACIRX+7)/8"ACIDATA size for diagnose event
120	(78)	CHARACTER	2	ACISUBC	Diagnose subcode

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
122	(7A)	CHARACTER	4	ACITCYL	Cyl/blk number. This field is used for subcode 03 of DIAGNOSE X'E4'
		EXPRESSION		ACIDE4SZ	"(*-ACISUBC+7)/8" ACIDATA size for DIAGNOSE X'E4'
For DIAGNOSE X'290':					
120	(78)	BITSTRING	2	ACI290SC	DIAGNOSE X'290' subcode issued
124	(7C)	CHARACTER	8	ACI290UI	Target userid
132	(84)	BITSTRING	4	ACI290DN	Target device num (subcd 4 only)
132	(84)	BITSTRING	3	ACI290Q	Target spool queue (subcd 0 only)
136	(88)	BITSTRING	4	ACI290ID	Target spool file ID (subcd 0 only)
		EXPRESSION		ACID290SZ	"(*-ACI290SC+7)/8" ACIDATA size for DIAGNOSE X'290'
120	(78)	CHARACTER	8	ACIORIG	Spool file origin ID
128	(80)	SIGNED	4	ACIFSTPG	DASD address of first page
132	(84)	SIGNED	2	ACISPLID	Spoolid
134	(86)	BITSTRING	6	ACITOD	Old TOD stamp of first page
140	(8C)	BITSTRING	8	ACICMDIS	Command issuer
		EXPRESSION		ACISPLSZ	"(*-ACIORIG+7)/8"ACIDATA size for SPXTAPE event
134	(86)	SIGNED	2	ACISPIDN	New spool ID
120	(78)	SIGNED	2	ACIPATH	Pathid for IUCV/APPCVM event
122	(7A)	BITSTRING	6		Reserved for future IBM use
		EXPRESSION		ACIUCVSZ	"(*-ACIPATH+7)/8"ACIDATA size for IUCV event
		EXPRESSION		ACISEVSZ	"(*-ACIPATH+7)/8"ACIDATA size for APPCVM SEVER
128	(80)	CHARACTER	8	ACISERVR	Server user ID
136	(88)	CHARACTER	8	ACIQUAL	LU name qualifier (Gate_LU)
144	(90)	CHARACTER	8	ACITLUN	Target LU name (Gate_known_LU)
		EXPRESSION		ACICONSZ	"(*-ACIPATH+7)/8"ACIDATA size for APPCVM CONNECT

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
120	(78)	BITSTRING	24	ACIVMCF	Command name for VMCF event
		EXPRESSION		ACIVMCSZ	"(*-ACIVMCF+7)/8"ACIDATA size for VMCF event
120	(78)	CHARACTER	4	ACIDETAD	VDEV returner's device (the device was GIVEn to this user)
124	(7C)	CHARACTER	4	ACIRECAD	VDEV of the Giver's device. (this user is now receiving the device back.)
		EXPRESSION		ACIGIVSZ	"(*-ACIDETAD+7)/8"ACIDATA size for GIVE event
120	(78)	SIGNED	4	ACISCYL	Starting cyl/blk on the DASD
124	(7C)	SIGNED	4	ACIECYL	Ending cylinder/block
128	(80)	CHARACTER	4	ACIRDEV	Real device number
132	(84)	CHARACTER	6	ACIVOLSR	Volume serial
		EXPRESSION		ACICCWSZ	"(*-ACISCYL+7)/8"ACIDATA size for Maintccw event
For POSIX set ID function:					
120	(78)	SIGNED	4	ACIORUID	Old (current) real UID
124	(7C)	SIGNED	4	ACIOEUID	Old (current) effective UID
128	(80)	SIGNED	4	ACIOSUID	Old (current) saved set-UID
132	(84)	SIGNED	4	ACIORGID	Old (current) real GID
136	(88)	SIGNED	4	ACIOEGID	Old (current) effective GID
140	(8C)	SIGNED	4	ACIOSGID	Old (current) saved set-UID
144	(90)	SIGNED	4	ACINUID	New UID (equal to ACIORUID if UID not being changed)
148	(94)	SIGNED	4	ACINGID	New GID (equal to ACIORGID if GID not being changed) if ACIPXUSN is off. Otherwise, not used.
152	(98)	CHARACTER	8	ACINGNAM	The group name that identifies the new GID if ACIPXUSN is on. Otherwise, not used. May be in mixed case.
160	(A0)	SIGNED	4	ACIOSGCT	Count of old (current) SGIDs (Valid only for code ACISETSG)

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
164	(A4)	SIGNED	4	ACIOSGLS	Address of old (current) SGID list (Valid only for code ACISETSG)
168	(A8)	SIGNED	4	ACINSGCT	Count of SGIDs in new list (Valid only for code ACISETSG)
172	(AC)	SIGNED	4	ACINSGLS	Address of new SGID list (Valid only for code ACISETSG)
		EXPRESSION		ACISIDSD	"(ACISIDSB+7)/8" ACIPARMS size for POSIX set IDs functions
For POSIX group database query functions:					
120	(78)	SIGNED	4	ACIGRUID	Real UID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
124	(7C)	SIGNED	4	ACIGEUID	Effective UID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
128	(80)	SIGNED	4	ACIGSUID	Saved set-UID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
132	(84)	SIGNED	4	ACIGRGID	Real GID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
136	(88)	SIGNED	4	ACIGEGID	Effective GID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
140	(8C)	SIGNED	4	ACIGSGID	Saved set-GID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
144	(90)	SIGNED	4	ACIGMCNT	Count of members of the group.
148	(94)	SIGNED	4	ACIGGID	GID of the POSIX group for which information is to be returned if ACIPXUSN is off.
152	(98)	CHARACTER	8	ACIGGNAM	The group name of the POSIX group for which information is to be returned if ACIPXUSN is on. May be in mixed case.
		EXPRESSION		ACIGB1SD	"(ACIGB1SB+7)/8" ACIPARMS size without buffer list for POSIX group database
176	(B0)	CHARACTER	0	ACIGBUFL	Group database buffer list. Repeats the following fields: ACIGRPMA and ACIGRPML

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
176	(B0)	SIGNED	4	ACIGRPMA	Address of the buffer to contain the member names of the input group
180	(B4)	SIGNED	4	ACIGRPML	Length of the buffer pointed to by ACIGRPMA
		EXPRESSION		ACIGDBSD	"(ACIGDBSB+7)/8" ACIPARMS size for 1 POSIX group database buffer list entry
For POSIX user database query functions:					
120	(78)	SIGNED	4	ACIURUID	Real UID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
124	(7C)	SIGNED	4	ACIUEUID	Effective UID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
128	(80)	SIGNED	4	ACIUSUID	Saved set-UID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
132	(84)	SIGNED	4	ACIURGID	Real GID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
136	(88)	SIGNED	4	ACIUEGID	Effective GID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
140	(8C)	SIGNED	4	ACIUSGID	Saved set-GID of the process requesting the data if ACIPXIDS is on. Otherwise, unpredictable and should not be used.
144	(90)	SIGNED	4	ACIUUID	UID for which information is to be returned if ACIPXUSN is off.
148	(94)	CHARACTER	8	ACIUUNAM	User name for which information is to be returned if ACIPXUSN is on. May be in mixed case.
156	(9C)	SIGNED	4	ACIUGID	GID of the user's primary POSIX group. (output only)
160	(A0)	CHARACTER	8	ACIUGNAM	Group name of the user's primary POSIX group. (output only)
		EXPRESSION		ACIUB1SD	"(ACIUB1SB+7)/8" ACIPARMS size without buffer list for POSIX user database

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
184	(B8)	CHARACTER	0	ACIUBUFL	User database buffer list. Repeats the following fields: ACIUIWDA and ACIUIWDL
184	(B8)	SIGNED	4	ACIUIWDA	Address of the buffer to contain user's initial working directory
188	(BC)	SIGNED	4	ACIUIWDL	Length of the buffer pointed to by ACIUIWDA
BITS DEFINED IN ACIUIWDL:					
188	(BC)	BITSTRING	4	ACIUIWDI	Indicate who provided POSIX IWDIR
		`1... ....`		ACIUIWDE	POSIX IWDIR provided by ESM.
192	(C0)	SIGNED	4	ACIUIUPA	Address of the buffer to contain user's initial user program
196	(C4)	SIGNED	4	ACIUIUPL	Length of the buffer pointed to by ACIUIUPA
BITS DEFINED IN ACIUIUPL:					
196	(C4)	BITSTRING	4	ACIUIUPI	Indicate who provided POSIX IUPGM
		`1... ....`		ACIUIUPE	POSIX IUPGM provided by ESM.
200	(C8)	SIGNED	4	ACIUFSRA	Address of the buffer to contain user's file system root
204	(CC)	SIGNED	4	ACIUFSRL	Length of the buffer pointed to by ACIUFSRA
BITS DEFINED IN ACIUFSRL:					
204	(CC)	BITSTRING	4	ACIUFSRI	Indicate who provided POSIX FSROOT
		`1... ....`		ACIUFSRE	POSIX FSROOT provided by ESM.
208	(D0)	SIGNED	4	ACIUSGIA	Address of the buffer to contain the user's supplementary GID list
212	(D4)	SIGNED	4	ACIUSGIL	Length of the buffer pointed to by ACIUSGIA
BITS DEFINED IN ACIUSGIL:					
212	(D4)	BITSTRING	4	ACIUSGII	Indicate who provided POSIX SGIDs
		`1... ....`		ACIUSGIE	POSIX SGIDs provided by ESM.

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		EXPRESSION		ACIUDBSD	"(ACIUDBSB+7)/8" ACIPARMS size for 1 POSIX user database buffer list entry
For COUPLE (COUPLEN) network command:					
120	(78)	SIGNED	4	ACILVIDA	Address of buffer to contain the halfword VLAN ids
124	(7C)	SIGNED	4	ACILVIDL	Buffer size in bytes (2 times the number of VLAN ids)
128	(80)	CHARACTER	0	ACIVLANC	Variable audit data
		EXPRESSION		ACICPLSD	"(ACICPLSB+7)/8" ACIPARMS size without buffer list for COUPLE (COUPLEN)
For Directory Command Audit-Only Call:					
120	(78)	CHARACTER	3	ACIDIRTO	'TO '
123	(7B)	CHARACTER	8	ACIDIRUI	User identifier
131	(83)	CHARACTER	2	ACIDIRND	': '
133	(85)	CHARACTER	0	ACIDIRCM	Command
120	(78)	BITSTRING	1	ACIACCESS	Access type
BITS DEFINED IN ACIACCESS:					
		`.... ....`		ACIACQUERY	X'00' Query access
		`.... ...1`		ACIACREAD	X'01' Read access
		`.... ..11`		ACIACWRITE	X'02'+ACIACREAD Write access
		`.... .111`		ACIACPRIV	X'04'+ACIACWRITE Privileged access
		`.... 1111`		ACIACFULL	X'08'+ACIACREAD Full access
121	(79)	BITSTRNG	1	ACILOGGING	Log bits
BITS DEFINED IN ACILOGGING:					
		`.... ....`		ACILOGPERESM	X'00' ESM determines log
		`.... ...1`		ACILOGNOFAIL	X'01' Authorization failure not recorded

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
		.... ..1.		ACILOGNONE	X'02' Attempt not recorded
		11		ACILOGNOSTAT	X'03' No log; no resource statistics updated
124	(7C)	SIGNED	4	ACIREASON	Reason code
128	(80)	CHARACTER	8	ACICLASS	Class name
136	(88)	SIGNED	2	ACIRESNAMELEN	Length of the resource name
138	(8A)	CHARACTER	246	ACIRESNAME	Resource name
384	(180)	SIGNED	2	ACILOGDATALEN	Length of log string
386	(182)	CHARACTER	255	ACILOGDATA	Free form log string for log record
For LOGOFF due to a successful guest relocation:					
120	(78)	CHARACTER	12	ACIRLODT	"RELOCATE TO " keywords
132	(84)	CHARACTER	8	ACIRLODN	Destination node
140	(8C)	CHARACTER	4	ACIRLOBY	"BY" keyword
144	(90)	CHARACTER	8	ACIRLOCI	VMRELOCATE command issuer's name
152	(98)	CHARACTER	4	ACIRLOAT	"AT" keyword
156	(9C)	CHARACTER	8	ACIRLOIN	VMRELOCATE command issuer's node
For LOGOFF of skeleton VMDBK due to a failed guest relocation:					
120	(78)	CHARACTER	14	ACIRLAFA	"RELOCATE FROM" keywords
134	(86)	CHARACTER	8	ACIRLASC	Source node
142	(8E)	CHARACTER	8	ACIRLAAK	"ABORTED" keyword
SECURITY CONSTANTS FOR CP USE ONLY:					
		.... ..1.		ACIXAQ	X'02' Query command
		11		ACIXAQV	X'03' Query virtual operand
		.... .1..		ACIXAQVM	X'04' Query virtual miscellaneous
		.... .1.1		ACIXAS	X'05' SET command

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name (Dim)**	**Description**
For LOGON and LOGOFF of an AT command guest:					
120	(78)	CHARACTER	12	ACIATCDT	"AT command" keywords
132	(84)	CHARACTER	8	ACIATCDN	Target node
140	(8C)	CHARACTER	4	ACIATCBY	"BY" keyword
144	(90)	CHARACTER	8	ACIATCCI	Command issuer
152	(98)	CHARACTER	4	ACIATCAT	"AT" keyword
156	(9C)	CHARACTER	8	ACIATCIN	Issuer's node

CP Calls to the ACI

The following sections describe the format of the ACIPARMS parameter list, that CP passes on the call to the ESM, for each of the commands and diagnose codes that support ESM protection. For more information about CP commands, see *z/VM: CP Commands and Utilities Reference*. For more information about diagnose codes, see Part 1, “CP DIAGNOSE Instructions,” on page 1.

Generic Command and DIAGNOSE Audit Calls

This section specifies ACIPARMS formats for the audit-only calls made for the majority of commands and diagnose codes. These calls are made to HCPRPIRA.

Generic Audit Call for CP Commands

This is the ACIPARMS format for commands which are audited from the command router. The command router will also make this call in the case of privilege failure for commands which are normally audited from the command processor. In addition, some command processors use this format for audit-only calls in error cases. Table 81 on page 639 shows the ACIPARMS format for the command router audit call.

Table 81. Generic command audit format of ACIPARMS

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	Audit setting for each privilege class or ACIANYAU if this is an 'ANY' class command.
ACIENV	ACIXAC
ACIRUSR	Command issuer's user ID.
ACIRGRP	Command issuer's ACI group name.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD | ACIBAD + ACIANY (if this is an 'ANY' class command)
ACICMDTP	Command version **Note:** This field is not filled in for 'ANY' class commands.
ACISLAB	Command issuer's SECLABEL.
ACIEVENT	Command name padded with blanks.

Table 81. Generic command audit format of ACIPARMS (continued)

Label	Contents
ACIDATA	Command line.

Table 82 on page 640 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 82. Supported Return Codes

RC	Meaning
X'00'	Processing continues.
X'04'	Processing continues.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

Generic Audit Call for DIAGNOSE Codes

Table 83 on page 640 shows the ACIPARMS parameter list for the DIAGNOSE router audit call.

Table 83. Generic DIAGNOSE Call Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDISZ (in bytes)
ACIBMAPA	ACIDGNAU
ACIENV	ACIXAD
ACIRUSR	DIAGNOSE issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD | ACIBAD
ACISLAB	Command issuer's SECLABEL.
ACIEVENT	DIAGxxx padded with blanks.
ACIRX	Contents of Rx register (in hex).
ACIRY	Contents of Ry register (in hex).
ACIRX1	Contents of Rx+1 register (in hex).
ACIRY1	Contents of Ry+1 register (in hex).

If the diagnose processor has detected an error prior to making the audit call, then no return code checking is done after the HCPRPIRA call, and CP will continue with the error processing for the condition detected.

Table 84 on page 640 shows the return code (ACICODE) checking performed on the return from HCPRPIRA in the case where no error was detected prior to the call.

Table 84. Supported Return Codes

RC	Status
X'00'	Processing continues.
X'04'	Processing continues.

Table 84. Supported Return Codes (continued)

RC	Status
X'20'	DIAGNOSE fails: • If DIAGNOSE code X'04', X'34', or X'3C', then reflect a specification exception. • Otherwise, reflect a privileged operation exception.

ACIPARMS Parameter Lists for CP Commands

This section specifies the ACIPARMS formats that individual CP commands pass to the ESM. The calls are made to HCPRPIRA unless otherwise stated.

An ESM can provide security control and audit ability for guest LANs and virtual switches. Valid VLAN IDs range from x'0001'-x'0FFE'. The ESM keeps the VLAN ID list for each User ID and Owner/LANNAME combination, and it must provide the VLAN IDs in ascending sequence.

If there is not enough space to complete the initial ESM request, previously allocated space is released, and a call is made with a buffer with enough space for all the VLAN IDs. If there are more than the allowed 2000 VLAN IDs available, message HCP6528E is issued, access is denied, and the return call to HCPRPI is not made.

Note: Table 85 on page 641 shows examples of different access levels and their corresponding buffer size.

Table 85. ACILVIDL and Buffer Data Examples for VLAN AWARE Virtual Switches

Access Level	Buffer Size	Buffer Content
Access allowed for VLANs 1, 2, & 3	6	X'000100020003'
Access allowed for VLAN 2	2	X'0002'
Access allowed - no VLANs	2	X'FFFF'

Table 86. ACILVIDL and Buffer Data Examples for Guest LANs and VLAN UNAWARE virtual switches

Access Level	Buffer Size	Buffer Content
Access allowed — no VLANs	0	
Access allowed — no VLANs	2	X'FFFF'

For MAC processing, CP provides the SECLABEL for the requester, but the ESM must determine and maintain the SECLABEL for the LAN. The scope of this User ID and Owner/LANNAME combination does not extend beyond a router. If MAC processing is turned on, but protect is not turned on, MAC processing returns a return code of 0, with no VLANs. CP checks it's own control blocks for both permission and VLANs, and then treats this case as if the ESM returned a DEFER.

Both the ESM and CP can participate in allowing the COUPLE command to complete. If AUDIT is requested, the AUDIT bit is turned off for the initial ESM call. After both the ESM and CP have determined if the COUPLE command is allowed, another ESM call is made requesting that the AUDIT record be created by the ESM. This AUDIT record shows the final result.

AUTOLOG and XAUTOLOG Commands

Table 87 on page 642 shows the ACIPARMS parameter list for the ESM call for the AUTOLOG and XAUTOLOG commands. The call is made to HCPRPWEP. CP will not expect or accept the POSIX-related information unless the ESM has previously informed CP that it contains POSIX support. See “Entry Point HCPRPEPX - Notify CP of POSIX capabilities” on page 602 for additional information.

Table 87. (X)AUTOLOG Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACILOG
ACILEN	ACILGNSB
ACIBMAPA	Class AB, G, or ABG or X'00'.
ACIBMAPP	Class AB, G, or ABG or X'00'.
ACIBMAPM	Class AB, G or, ABG or X'00'.
ACIENV	ACIAUTO
ACIRUSR	Command issuer's userid, unless ACIRMSRC is set in ACIFLAG2.
ACITGRP	Target's ACI group name ((x)autologee).
ACITUSR	User ID of the target (user being (x)autologged).
ACILGOPT	Logon options: **ACINOPAS** No password check requested. **ACINPMT** No password prompt requested.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIWO or ACIGOOD + ACIWO + ACIRW **Note:** ACIRW will only be set for XAUTOLOG ON when the target is a logical device.
ACICMDTP	Class AB (X'C0') or class G (X'02').
ACIFLAG2	• ACIRMSRC is set if a private server is being xautologged by a remote source. In this case, an effective source is not available and ACIRUSR will not be set. • ACINPASS is set if the user ID being autologged is defined with the NOPASS operand in its directory entry.
ACISLAB	Command issuer's SECLABEL.
ACIEVENT	"AUTOLOG" or "XAUTOLOG" padded with blanks.
ACIVMDBK	VMDBK address of the user logging on.
ACIPASS	Contains password in EBCDIC format padded with blanks. If no password is set, this field will be all blanks.
ACITRMID	Terminal ID.
ACINWQFR	If ACINQLUN is on, the network qualifier of the SNA logical unit identifier defined in ACITRMID.
ACILBUFL	(X)AUTOLOG data buffer list. A list of contiguous entries describing the output buffers for this request. Each entry consists of a buffer address and length (in bytes). A buffer address of zero indicates that the corresponding database information, and its length, should not be returned. ACILEN can be used to calculate the number of entries in the list. The address and length field of each entry is described below.
ACILUIDA	Address of the buffer to contain the user's POSIX UID.
ACILUIDL	Length of the buffer pointed to by ACILUIDA.
ACILGIDA	Address of the buffer to contain the user's POSIX GID.

Table 87. (X)AUTOLOG Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACILGIDL	Length of the buffer pointed to by ACILGIDA.
ACILSGIA	Address of the buffer to contain the user's Supplementary GID list.
ACILSGIL	Length of the buffer pointed to by ACILSGIA.

Table 88 on page 643 shows the output fields set by the ESM in ACIPARMS on the return from HCPRPWEP.

Table 88. (X)AUTOLOG ESM output fields in ACIPARMS

Label	Contents
ACILOPTS	Flags, as follows: • ACILSIDE indicates that the ESM has provided the user's ACILSIDA value. • ACILSIDA indicates that the user is permitted to set other users' POSIX IDs. This replaces CP's POSIXOPT SETIDS value for the user.
ACILBUFL	(X)AUTOLOG data buffer list. If ACICODE = X'00', the high-order bit of each length field indicates whether the ESM has chosen to provide the corresponding data or to defer it to CP. If this bit is on, the data that the ESM wishes to provide has been placed in the buffers, and the length field of the entry contains the length, in bytes, of the data placed in the buffer pointed to by that entry. If this bit is off, CP will use its own data (usually from the user directory or a default value). If the Supplementary GID list was returned, it must include the user's primary GID. If ACICODE = X'28', the high-order bit of each length field indicates whether the ESM would provide the corresponding data or defer it to CP. If this bit is on, the length field of the entry contains the length, in bytes, of the buffer necessary to contain the corresponding information.

Table 89 on page 643 shows the return codes that CP supports in ACICODE on the return from HCPRPWEP.

Table 89. Supported HCPRPWEP Return Codes for the AUTOLOG and XAUTOLOG

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred; CP performs appropriate authorization checks. • Password check for AUTOLOG and XAUTOLOG PROMPT|PASSWORD • Directory check for XAUTOLOG G POSIX database information is acquired from the directory.
X'08'	The command fails with an error message: • HCP059E for AUTOLOG and XAUTOLOG PASSWORD|PROMPT • HCP6050E for XAUTOLOG G
X'0C' and X'10'	The command fails, and no message is issued.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

Table 89. Supported HCPRPWEP Return Codes for the AUTOLOG and XAUTOLOG (continued)

RC	Meaning
X'28'	One or more of the buffers provided on input is too small to contain the requested data. The length field of each ACILBUFL entry whose high-order bit of the length field is on contains the required buffer length, in bytes, for that item. CP can be expected to acquire larger buffers and request all of the data from the ESM again, via a POSIX User Database query.

AUTOLOG and XAUTOLOG Error Case Audit-Only Call

When the AUTOLOG, or XAUTOLOG, processor detects an error during command processing, an audit-only call will be made (if audit is on) to HCPRPIRA with the ACIPARMS format given in "Generic Audit Call for CP Commands" on page 639.

CHANGE TO Authorization Call

See "TRANSFER Command Authorization" on page 664.

CHANGE Command Auditing

Table 90 on page 644 shows ACIPARMS parameter list for a CHANGE command audit call.

Table 90. CHANGE Command Audit Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	Audit settings.
ACIENV	ACIXAC
ACIRUSR	Command issuer's user ID.
ACIFLAG	ACIGOOD or ACIBAD
ACIVERS	ACIVERS1
ACICMDTP	Command version.
ACIEVENT	"CHANGE" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACIDATA	Command line.

Table 91 on page 644 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 91. Supported HCPRPIRA Return Codes

RC	Meaning
X'00'	Processing continues.
X'04'	Processing continues.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

CHANGE Command MAC/AUDIT

Table 92 on page 645 shows ACIPARMS for the MAC/AUDIT call for each file changed. Only the Class G version of the CHANGE command will be subject to MAC checking. Only a D Class user can use the SECLABEL option on the CHANGE command.

Table 92. CHANGE Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT settings.
ACIBMAPM	MAC settings.
ACIENV	ACISPF
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIWO
ACICMDTP	Command version.
ACIEVENT	"CHANGE" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACITLAB	Spool file SECLABEL.
ACIORIG	Spool file origin ID.
ACIFSTPG	DASD address of first page.
ACISPLID	Spool ID.
ACINSPLD	New SECLABEL value if the SECLABEL option was used.

Table 93 on page 645 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 93. Supported HCPRPIRA Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP356E, indicating access denied.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

CLOSE TO Command Authorization Call

This call is controlled by the TRANSFER class G protect setting.

Note: The CLOSE command is audited generically from the command router.

Table 94 on page 645 shows the ACIPARMS parameter list for a DAC HCPRPIRA call for the CLOSE command, when the TO option is used, and the target is not the issuer.

Table 94. CLOSE TO Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACISPOOL
ACILEN	ACISIZE (in bytes)
ACIBMAPP	Transfer's protect settings.

Table 94. CLOSE TO Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIENV	ACITRANS
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACITGRP	ACI group name of user ID to whom files are being sent.
ACITUSR	User ID of user to whom files are being sent.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Class G (X'02').
ACIEVENT	"CLOSE" padded with blanks.
ACISLAB	Command issuer's SECLABEL.

Table 95 on page 646 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 95. Supported HCPRPIRA Return Codes for the CLOSE TO Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues
X'08'	The command fails with error message HCP007E, indicating an invalid user ID.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

COUPLE Command MAC/AUDIT

Note: If the target of the couple was not successfully found, then a MAC authorization ESM call will not be made. This means that when the 'target SECLABEL' is not available, MAC checking will not be requested. Table 96 on page 646 shows the ACIPARMS parameter list for an HCPRPIRA call for the COUPLE command.

Table 96. COUPLE Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	AUDIT settings.
ACIBMAPM	MAC settings.
ACIENV	ACIXAC
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIRW
ACICMDTP	Command version.
ACIEVENT	Command name padded with blanks.

Table 96. COUPLE Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIDATA	Command line data.
ACISLAB	Command issuer's SECLABEL.
ACITLAB	SECLABEL of target user ID if successfully obtained by CP, otherwise from X'00'.

Table 97 on page 647 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 97. Supported HCPRPIRA Return Codes for the COUPLE Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP644, indicating an unknown/unauthorized CP command.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

COUPLE (COUPLEN) Network Command

Note: An ESM audit call may be made after both the ESM and CP have made their decisions about whether to return the VLAN list. Table 98 on page 647 shows the ACIPARMS parameter list for an HCPRPIRA call for the COUPLE (COUPLEN) Network command.

Table 98. COUPLEN Command Format of the ACIPARMS Parameter List

Label	ORG	IPUSER	Contents
ACIRUSR	DS	CL8	Requester's user ID.
ACITUSR	DS	CL8	Target LAN owner.
ACILFLAG	DS	X	Audit promiscuous mode.
ACINODE	DS	CL8	Target <lanname> (ACILNID).
ACIRSECL	DS	CL8	Requester's SECLABEL.
ACILVIDA	DS	A	ADDR of buffer to contain VLAN IDs. VLAN IDs are two bytes each.
ACILVIDL	DS	F	Buffer size in bytes.
ACIVLANC	DS	0D	Variable command data.
ACICPLSZ	EQU		(*ACIVLANC+7)/8

Table 99 on page 647 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 99. Supported HCPRPIRA Return Codes for the COUPLEN Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP6011E, indicating the user ID is not authorized to COUPLE to the specified LAN.

Table 99. Supported HCPRPIRA Return Codes for the COUPLEN Command (continued)

RC	Meaning
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.
X'28'	Buffer is too small. ACILVIDL contains the total bytes needed, and the External Security Manager writes as many as space allows. A maximum of 2000 VLAN IDs is allowed. If the buffer size indicates that the ESM wants to return more then the allowed maximum, error message HCP6528E is issued, and the COUPLE command does not complete.
Other	The command fails with error message HCP644E, indicating an unknown CP command, if a return code other than the ones listed above is received.

FOR Command MAC/AUDIT

Table 100 on page 648 shows the ACIPARMS parameter list passed to HCPRPIRA for a MAC/AUDIT call made by the FOR command.

Table 100. FOR Command MAC/AUDIT Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	AUDIT call only. Not set for MAC call. **CLASSC** If the user ID issuing the FOR command is not the SECUSER of the target user ID. **CLASSG** If the user ID issuing the FOR command is the SECUSER of the target user ID.
ACIBMAPM	MAC call only. Not set for AUDIT call. **CLASSC** If the user ID issuing the FOR command is not the SECUSER of the target user ID. **CLASSG** If the user ID issuing the FOR command is the SECUSER of the target user ID.
ACIENV	ACIXAC
ACIRUSR	User ID that issued the FOR command.
ACITUSR	MAC call only. Not set for AUDIT call. • User ID of the FOR command target.
ACIVERS	ACIVERS1
ACIFLAG	• AUDIT call: ACIGOOD | ACIBAD • MAC call: ACIGOOD + ACIRW
ACICMDTP	CLASSC + CLASSG
ACISLAB	SECLABEL of the FOR command issuer.
ACITLAB	SECLABEL of the FOR command target user ID.
ACIEVENT	"FOR" padded on the right with blanks.
ACIDATA	The FOR command issued without PATH and TOKEN operands.

Table 101 on page 649 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 101. Supported Return Codes for the FOR Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP070E, indicating the user issuing the FOR command is not authorized.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

GIVE (Auditing the Return of a Given Device)

Table 102 on page 649 shows the ACIPARMS format for an HCPRPIRA audit call when a given device is returned via DETACH or LOGOFF. This call is based on the audit bit for the GIVE command. The GIVE command is audited generically from the command router.

Table 102. GIVE return

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIGIVSZ (in bytes)
ACIBMAPA	CLASSB
ACIENV	ACIGVRTN
ACIRUSR	Requestor (user returning the given device).
ACITUSR	Target user (original giver, now getting device back).
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Command version.
ACIEVENT	"GIVERETN" padded with blanks.
ACISLAB	Requestor's SECLABEL.
ACITLAB	Target SECLABEL (SECLABEL of the giver).
ACIDETAD	VDEV returner's device (the device was given to this user).
ACIRECAD	VDEV of the Giver's device (this user is now receiving the device back).

There is no return code checking performed after the ESM call.

IPL of a RESTRICTED Segment

See “RSTDSEGt” on page 693.

LINK Command Authorization/Audit Call

Note: The MDISK function, and not LINK, controls LINKs-to-self.

The LINK authorization call is based off the audit, protect, and MAC bits for the LINK command. If, after making the call and receiving ESM authorization, CP denies or downgrades the requested access, then an audit-only call will be made. The ACIPARMS format for this audit-only call will be the same as below and:

- ACIBMAPP and ACIBMAPM will be zeroes.
- ACIMODE will be 'RR' if the access has been downgraded to read and 'XX' if the access has been denied.

Table 103 on page 650 shows the ACIPARMS format for the LINK command.

Table 103. LINK Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACILINK
ACILEN	ACISIZE (in bytes)
ACIBMAPA	Class G (X'02') or X'00'.
ACIBMAPP	Class G (X'02') or X'00'.
ACIBMAPM	Class G (X'02') or X'00'.
ACIENV	ACIDISK
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACITGRP	ACI group name of the minidisk owner user ID.
ACITUSR	Owning user ID of the minidisk that is being linked to.
ACIMODE	Minidisk access mode in EBCDIC format.
ACIADDR	Link-To address in EBCDIC format.
ACIVERS	ACIVERS1
ACITADDR	Link-As address in EBCDIC format.
ACIFLAG	ACIRO or ACIRW + ACIGOOD
ACICMDTP	Class G (X'02').
ACIEVENT	"LINK" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACITOD	Additional field passed only for a shareable virtual disk in storage; contains the virtual disk in storage identifier (the TOD stamp for the creation of the virtual disk in storage).

Table 104 on page 650 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 104. Supported HCPRPIRA Return Codes for the LINK Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred, and CP performs the authorization check.
X'08'	The command fails with error message HCP298E, indicating the minidisk is not linked.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

LINK Audit-Only Call in Error Processing

When the LINK processor detects an error during command processing before the ESM is called for authorization, then an audit-only call will be made (if audit is on) to HCPRPIRA with the ACIPARMS format

given in “Generic Audit Call for CP Commands” on page 639. This call will not be made if the call described in “LINK Command Authorization/Audit Call” on page 649 has been made.

LOGOFF Command

Table 105 on page 651 shows the ACIPARMS parameter list for an HCPRPIRA call which is made when a virtual machine is being logged off. For example, when a user LOGOFFs or is FORCEd off.

Table 105. LOGOFF Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIDEL
ACILEN	ACIPARMS (in bytes)
ACIBMAPA	ACIANYAU or X'00'.
ACIBMAPP	ACIANYPR
ACIENV	ACIXAC
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY
ACIEVENT	"LOGOFF" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
The following fields are valid only for a logoff due to a successful guest relocation:	
ACIRLODT	"RELOCATE TO"
ACIRLODN	The name of the SSI cluster member that is the target destination of the relocation.
ACIRLOBY	"BY"
ACIRLOCI	The userid of the VMRELOCATE command issuer.
ACIRLOAT	"AT"
ACIRLOIN	The SSI cluster member name where the command issuer was logged on.
The following fields are valid only for a skeleton VMDBK logoff due to an unsuccessful guest relocation:	
ACIRLAFA	"RELOCATE FROM"
ACIRLASC	The name of the SSI cluster member that was the source for the relocation.
ACIRLAAK	"ABORTED"

CP does not check for a return code on the return from HCPRPIRA.

LOGOFF of an AT Command Guest

Table 106 on page 651 shows the ACIPARMS parameter list for an HCPUSP call for the logoff of a guest that has issued the AT command.

Table 106. AT Command Guest LOGOFF Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIDEL
ACILEN	ACIATCSD * 8

Table 106. AT Command Guest LOGOFF Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIBMAPA	ACIANYAU if LOGOFF is audited.
ACIBMAPP	ACIANYPR
ACIENV	ACIXAC
ACIRGRP	ACI group name.
ACIRUSR	User ID of the AT command guest.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY
ACIEVENT	"LOGOFF" padded with blanks.
ACIVMDBK	Address of the AT command guest VMDBK.
ACISLAB	Security label.
ACIATCDT	"AT_LOGOFF"
ACIATCDN	The target node on which the AT command was executed.
ACIATCBY	"BY"
ACIATCCI	User ID of the AT command issuer.
ACIATCAT	"AT"
ACIATCIN	Node of the AT command issuer.

LOGON Command

Table 107 on page 652 shows the ACIPARMS parameter list for an HCPRPWEP call for the LOGON command.

Table 107. LOGON Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACILOG
ACILEN	ACILGNSB
ACIBMAPA	ACIANYAU or X'00'
ACIBMAPP	ACIANYPR
ACIBMAPM	ACIANYMC or X'00'
ACIENV	ACIXAC
ACIRGRP	ACI group name from the user directory entry of the virtual machine being logged on.
ACIRUSR	User ID of the virtual machine being logged on.

Table 107. LOGON Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACILGOPT	Logon options: **ACINPMT** No password prompt requested **ACIREC** This is a reconnect **ACILOGCL** Logical terminal **ACILOCAL** Local terminal **ACISNA** SNA (VTAM) terminal **ACILOGIP** Logical terminal with IP address **ACITTY** TTY terminal (may be set if either ACILOCAL or ACISNA is set)
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD + ACIRW
ACIFLAG2	ACINPASS is set if the user ID being logged on is defined with the NOPASS operand in its directory entry.
ACIEVENT	"LOGON" padded with blanks.
ACIPASS	Contains password in EBCDIC format padded with blanks. If no password is set, this field will be blank.
ACIVMDBK	Address of the VMDBK for the user logging on. This field is guaranteed to be valid only until the first loss of control after entry to HCPRPWEP.
ACIBYVAL	The user ID following the BY operand of the LOGON command, if the BY operand was specified, or zeros.
ACISLAB	Specified SECLABEL padded with blanks, or zeros if no SECLABEL was specified.
ACITLAB	Creator of the logical device's SECLABEL or zeros if not a LOGON through a logical device.
ACITRMID	One of the following: • Terminal ID. • If ACILOGCL and ACILOGIP are set in ACILGOPT, an IPv4 address (see Note “1” on page 654). • If ACILOGCL is not set, but ACILOGIP is set, the 8 high-order bytes of an IPv6 address. The 8 low-order bytes are in ACITRMI2..
ACITRMI2	If ACILOGCL is not set, but ACILOGIP is set, the 8 low-order bytes of an IPv6 address.
ACIDSPID	Dispatched user ID.
ACILBUFL	LOGON data buffer list. A list of contiguous entries describing the output buffers for this request. Each entry consists of a buffer address and length (in bytes). A buffer address of zero indicates that the corresponding database information, and its length, should not be returned. ACILEN can be used to calculate the number of entries in the list. The address and length field of each entry is described below.

Table 107. LOGON Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACILUIDA	Address of the buffer to contain the user's POSIX UID.
ACILUIDL	Length of the buffer pointed to by ACILUIDA.
ACILGIDA	Address of the buffer to contain the user's POSIX GID.
ACILGIDL	Length of the buffer pointed to by ACILGIDA.
ACILSGIA	Address of the buffer to contain the user's Supplementary GID list.
ACILSGIL	Length of the buffer pointed to by ACILSGIA.

Notes:

1. IP addresses are normally written in dotted decimal format (for example, 9.130.58.78). In the LOGON command parameter list, each segment of the IP address is converted to a two-digit hexadecimal value. For example, 9 is converted to 09, and 130 is converted to 82. The result is an eight-byte string of four two-digit hexadecimal numbers in character form. So, 9.130.58.78 becomes the character string 09823A4E.
2. If the ESM supports password phrases, then ACIPASS is blank and the following fields are filled in:

   ACIOPPLN
   Contains the length of the password phrase. If no password phrase is set, this field will contain zero.

   ACIOPP
   Contains the password phrase in EBCDIC.

Table 108 on page 654 shows the output fields set by the ESM in ACIPARMS on the return from HCPRPWEP.

Table 108. LOGON ESM output fields in ACIPARMS

Label	Contents
ACILOPTS	Flags, as follows: • ACILSIDE indicates that the ESM has provided the user's ACILSIDA value. • ACILSIDA indicates that the user is permitted to set other users' POSIX IDs. This replaces CP's POSIXOPT SETIDS value for the user.
ACILBUFL	LOGON data buffer list. If ACICODE = X'00', the high-order bit of each length field indicates whether the ESM has chosen to provide the corresponding data or to defer it to CP. If this bit is on, the data that the ESM wishes to provide has been placed in the buffers, and the length field of the entry contains the length, in bytes, of the data placed in the buffer pointed to by that entry. If this bit is off, CP will use its own data (usually from the user directory or a default value). If the Supplementary GID list was returned, it must include the user's primary GID. If ACICODE = X'28', the high-order bit of each length field indicates whether the ESM would provide the corresponding data or defer it to CP. If this bit is on, the length field of the entry contains the length, in bytes, of the buffer necessary to contain the corresponding information.

Table 109 on page 655 shows the return codes that CP supports in ACIPARMS on the return from HCPRPWEP.

Table 109. Supported HCPRPWEP Return Codes for the LOGON Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred for CP to perform password verification. POSIX database information is acquired from the directory.
X'08'	The command fails with error message HCP050E, indicating logon was unsuccessful due to an incorrect password.
X'0C'	The command fails, no message is issued, and the logon skeleton is forced off the system.
X'10'	The command fails and no message is issued (logon prompt is displayed).
X'20'	The command fails. Error message HCP6525E, indicating the ESM is unavailable, is sent to the issuing user and to the system operator.
X'28'	One or more of the buffers provided on input is too small to contain the requested data. The length field of each ACILBUFL entry whose high-order bit of the length field is on contains the required buffer length, in bytes, for that item. CP can be expected to acquire larger buffers and request all of the data from the ESM again, using a POSIX User Database query.

LOGON of an AT Command Guest

Table 110 on page 655 shows the ACIPARMS parameter list for an HCPXAT call for the LOGON of a guest that will issue the AT command.

Table 110. AT Command Guest LOGON Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACILOG
ACILEN	ACIATCSB
ACIBMAPA	ACIANYAU if LOGON command is audited.
ACIBMAPP	ACIANYPR
ACIBMAPM	ACIANYMC
ACIENV	ACIXAC
ACIRGRP	ACI group name.
ACIRUSR	User ID of the AT command guest.
ACIVERS	ACIVERS1
ACIDSPID	User ID the AT command was initiated from on the target node.
ACIFLAG	ACIANY + ACIGOOD.
ACIEVENT	AT "LOGON" padded with blanks.
ACIVMDBK	Address of the AT command guest VMDBK.
ACISLAB	Security label.
ACIATCDT	"AT_LOGON"
ACIATCDN	The target node on which the AT command will be executed.
ACIATCBY	"BY"

Table 110. AT Command Guest LOGON Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIATCCI	User ID of the AT command issuer.
ACIATCAT	"AT"
ACIATCIN	Node of the AT command issuer.

MESSAGE Command MAC/AUDIT

Table 111 on page 656 shows the ACIPARMS parameter list for an HCPRPIRA call for the MESSAGE command.

Note: MAC will not be performed when:

- The ALL option is specified.
- The sender or the receiver is the system operator.

Table 111. MESSAGE Command MAC/AUDIT Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	AUDIT settings.
ACIBMAPM	MAC settings.
ACIENV	ACIXAC
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD|ACIBAD + ACIWO + ACIANY (for the 'ANY' class message command)
ACICMDTP	CLASSA+CLASSB (for message and warning all), CLASSB (for MSGNOH), CLASSG (for smsg) or CLASSA+CLASSB+CLASSC (Warning not to ALL).
ACIEVENT	"MESSAGE" or "MSGNOH" or "SMSG" or "WARNING" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACITLAB	SECLABEL of target user ID if valid user ID found.
ACIDATA	Command line data.

Table 112 on page 656 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 112. Supported Return Codes for the MESSAGE Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP003E, indicating an invalid option.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

MSGNOH Function MAC/AUDIT

See “MESSAGE Command MAC/AUDIT” on page 656.

PURGE Command Audit

The PURGE command calls for audit with the ACIPARMS format shown in “Generic Audit Call for CP Commands” on page 639.

Table 113 on page 657 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 113. Supported HCPRPIRA Return Codes for the PURGE Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'20'	If Purge command processing detected an error prior to the ESM call then the purge processor will issue the appropriate message. Otherwise, the command fails with error message HCP6525E, indicating the ESM is unavailable.

QUERY TAG and TAG QUERY Command MAC/AUDIT

Table 114 on page 657 shows the ACIPARMS parameter list for an HCPRPIRA call for the QUERY TAG and TAG QUERY commands. For QUERY TAG file and TAG QUERY file this is a MAC/AUDIT call. Otherwise, it is audit-only.

Table 114. QUERY TAG FILE Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes).
ACIBMAPA	AUDIT Settings.
ACIBMAPM	MAC Settings (only for QUERY TAG and TAG QUERY file).
ACIENV	ACIXAC
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIRO
ACICMDTP	Class G (X'02')
ACIEVENT	"TAG" padded with blanks.
ACISLAB	Command issuer's user SECLABEL.
ACITLAB	SECLABEL of target spool file (only for TAG QUERY file and QUERY TAG file).
ACIDATA	Command line data.

Table 115 on page 658 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 115. Supported Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP026E, indicating a missing or invalid operand.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

QUERY RDR/PRT/PUN Command

Table 116 on page 658 shows the ACIPARMS parameter list for a MAC-only HCPRPIRA call to determine whether key fields can be displayed in the command response.

Note: The QUERY RDR/PRT/PUN command is audited generically with a separate call.

Table 116. QUERY RDR/PRT/PUN Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPM	MAC Settings.
ACIENV	ACISPF
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIRO + ACIGOOD
ACICMDTP	Command version.
ACIEVENT	Command operand (READER, PRINTER or PUNCH) padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACITLAB	SECLABEL of target spool file.
ACIORIG	Spool file origin ID.
ACIFSTPG	DASD address of first page.
ACISPLID	Spool file ID.

Table 117 on page 658 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 117. Supported HCPRPIRA Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues
X'08'	All fields except ORIGINID, FILEID, CLASS, HOLD, DATE and TIME will be filled with asterisks.
X'20'	All fields except ORIGINID, FILEID, CLASS, HOLD, DATE and TIME will be filled with asterisks.

SEND Command Audit

Table 118 on page 659 shows the ACIPARMS parameter list for an HCPRPIRA call for the audit call made by the SEND command.

Table 118. SEND Command Audit Call Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE (in bytes)
ACIBMAPA	CLASSC+CLASSG
ACIENV	ACIXAC
ACIRUSR	Issuer's user ID
ACITUSR	Target user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD or ACIBAD
ACISLAB	Issuer's SECLABEL
ACITLAB	Target user's SECLABEL
ACICMDTP	User's privelege classes
ACIEVENT	'SEND' padded with blanks

There is no return code checking performed after the ESM call.

SEND Command Security Label MAC Check

Table 119 on page 659 shows the ACIPARMS parameter list for an HCPRPIRA call for the SEND command security label MAC check.

Table 119. SEND Command Security Label MAC Check Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE (in bytes)
ACIBMAPA	ACISYSMC
ACIENV	ACIXAC
ACIRUSR	Issuer's user ID
ACITUSR	Target user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD + ACIRW or ACIANY + ACIGOOD + ACIWO
ACISLAB	Issuer's SECLABEL
ACITLAB	Target user's SECLABEL
ACIEVENT	'SEND_SECLBL' padded with blanks

Table 120 on page 660 shows the ACICODE return codes checked on return from HCPRPIRA.

Table 120. Supported Return Codes

RC	Meaning
<= X'04'	Authorization is granted and processing continues.
> X'04'	The command fails with error message HCP068E, indicating the user's SECLABELs failed the MAC check.

SMSG Function MAC/AUDIT

See "MESSAGE Command MAC/AUDIT" on page 656.

SPOOL TO Command

This call is controlled by the TRANSFER class G protect setting.

Note: The SPOOL command is audited generically from the command router.

Table 121 on page 660 shows the ACIPARMS parameter list for a DAC HCPRPIRA call for the SPOOL command, when the TO or FOR option is used, and the target is not the issuer.

Table 121. SPOOL Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACISPOOL
ACILEN	ACISIZE (in bytes)
ACIBMAPP	Transfer's protect settings.
ACIENV	ACITRANS
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACITGRP	ACI group name of user ID to whom files are being sent.
ACITUSR	User ID of user to whom files are being sent.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Class G (X'02').
ACIEVENT	"SPOOL" padded with blanks.
ACISLAB	Command issuer's SECLABEL.

Table 122 on page 660 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 122. Supported HCPRPIRA Return Codes for the SPOOL TO Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP007E, indicating an invalid user ID.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

SPXTAPE Dumping and Loading of Files

Table 123 on page 661 shows the ACIPARMS format for an audit-only call for each file dumped or loaded by the SPXTAPE DUMP or LOAD command. The SPXTAPE command is audited generically.

Table 123. ACIPARMS format

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT Settings.
ACIENV	ACISPTAP
ACIRUSR	Spool file owner's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	CLASSE+CLASSD | CLASSD | CLASSG
ACIEVENT	"SPLDUMP" or "SPLLOAD" padded with blanks.
ACITLAB	Spool file SECLABEL.
ACIORIG	Spool file origin ID.
ACIFSTPG	DASD address of first page.
ACISPLID	Spool file ID.
ACITOD	TOD clock value at time of SPXTAPE DUMP or LOAD.
ACICMDIS	Command issuer's user ID.

Table 124 on page 661 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 124. Supported Return Codes

RC	Meaning
X'00'	Processing continues.
X'04'	Processing continues.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

START Real Printer SECLABEL Authorization Call

Table 125 on page 661 shows the ACIPARMS format for the HCPRPIRA call for the STARTing a real device.

This call is based on MAC bit for the START command. This is an authorization-only call. The START command is audited from the command router.

Table 125. START real printer with SECLABEL option authorization call

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE (in bytes)
ACIBMAPM	CLASSD (X'10')

Table 125. START real printer with SECLABEL option authorization call (continued)

Label	Contents
ACIENV	ACIXAC
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuers user ID.
ACIVERS	ACIVERS1
ACITADDR	Address of printer right justified in EBCDIC.
ACIFLAG	ACIGOOD + ACIWO
ACICMDTP	CLASSD
ACIEVENT	"START" padded with blanks.
ACISLAB	Command issuers SECLABEL.
ACITLAB	Existing SECLABEL of the printer.
ACINSPLD	SECLABEL printer is being started at.

Table 126 on page 662 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 126. Supported Return Codes for the START Real Printer with SECLABEL

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP1013E, indicating an invalid operand was supplied for SECLABEL.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

STORE HOST Command

Table 127 on page 662 shows the ACIPARMS parameter list for the DAC/audit call for the STORE command when used with the HOST operand.

Table 127. STORE HOST Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	Audit setting.
ACIBMAPP	Protect setting.
ACIENV	ACIXAC
ACIRUSR	Command issuer's user ID.
ACIRGRP	Command issuer's ACI group name.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD | ACIBAD
ACICMDTP	Class C (X'20').

Table 127. STORE HOST Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACISLAB	Command issuer's SECLABEL.
ACIEVENT	Command name padded with blanks.
ACIDATA	Command line.

Table 128 on page 663 displays the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 128. Supported Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP003E, indicating an invalid option.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

TAG QUERY Command MAC/AUDIT

See “QUERY TAG and TAG QUERY Command MAC/AUDIT” on page 657.

TAG Command MAC/DAC/Audit

Table 129 on page 663 shows the ACIPARMS parameter list for an HCPRPIRA call for the TAG device command and TAG file command.

Note: MAC is only applicable to the TAG file command.

Table 129. TAG Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACITAG
ACILEN	ACISIZE + ACIDATA size (in bytes)
ACIBMAPA	Class G (X'02') or X'00'.
ACIBMAPP	Class G (X'02') or X'00'.
ACIBMAPM	Class G (X'02') or X'00 '. **Note:** The MAC setting can only be on for TAG FILE.
ACIENV	ACITAGN
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuers user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIWO
ACINODE	The first token of the tag text (padded with blanks if < 8 characters).
ACICMDTP	Class G (X'02').
ACIEVENT	"TAG" padded with blanks.
ACISLAB	Command issuers SECLABEL.

Table 129. TAG Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACITLAB	SECLABEL of target spool file (For the TAG file command).
ACIDATA	Command line data.

Table 130 on page 664 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 130. Supported Return Codes for the TAG Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP003E, indicating an invalid option.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

TRANSFER Command Authorization

Table 131 on page 664 shows the ACIPARMS parameter list for an HCPRPIRA DAC call for the TRANSFER and CHANGE TO commands when the file is being sent to someone other than the issuer.

Note: This is a DAC-only call. The CHANGE and TRANSFER commands are audited generically.

Table 131. TRANSFER Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACISPOOL
ACILEN	ACISIZE (in bytes)
ACIBMAPP	PROTECT settings.
ACIENV	ACITRANS
ACIRGRP	Command issuer's ACI group name from user's directory entry.
ACIRUSR	Command issuer's user ID.
ACITGRP	ACI group name of user ID to whom files are being sent.
ACITUSR	User ID of user to whom files are being sent.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD | ACIBAD
ACICMDTP	Class DG (X'12') or CLASS G (X'02').
ACIEVENT	"TRANSFER" or "CHANGE" padded with blanks.
ACISLAB	Command issuer's SECLABEL.

Table 132 on page 664 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 132. Supported Return Codes for the TRANSFER Command

RC	Meaning
X'00'	Authorization is granted and processing continues.

Table 132. Supported Return Codes for the TRANSFER Command (continued)

RC	Meaning
X'04'	Authorization is deferred, and CP performs the authorization check.
X'08'	The command fails with error message HCP007E, indicating an invalid user ID.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

TRANSFER Command

Table 133 on page 665 shows the ACIPARMS parameter list for an HCPRPIRA audit-only call for each file transferred as a result of the TRANSFER or CHANGE TO command.

Table 133. TRANSFER Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT Settings.
ACIENV	ACIXAT
ACIRUSR	Spool file owner ID.
ACITUSR	User ID to whom files are being sent
ACIEVENT	"TRANSFER" padded with blanks.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Command version.
ACISLAB	Command issuer's SECLABEL.
ACITLAB	SPOOL file's SECLABEL.
ACIORIG	Spool file origin ID.
ACIFSTPG	DASD address of first page.
ACISPLID	Old spool ID.
ACISPIDN	New spool ID.
ACICMDIS	Command issuer's user ID.

Table 134 on page 665 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 134. Supported HCPRPIRA Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is granted and processing continues.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

TRSAVE TO Command

This call is controlled by the TRANSFER class D protect setting.

Note: The TRSAVE command is audited generically from the command router.

Table 135 on page 666 shows the ACIPARMS parameter list for an HCPRPIRA DAC call for the TRSAVE TO command when the target is not the issuer.

Table 135. TRSAVE TO Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACISPOOL
ACILEN	ACISIZE (in bytes)
ACIBMAPP	Transfer's protect settings.
ACIENV	ACITRANS
ACIRGRP	Command issuer's ACI group name.
ACIRUSR	Command issuer's user ID.
ACITGRP	ACI group name of the user ID to whom files are being sent.
ACITUSR	User ID to whom files are being sent.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Class D (X'10').
ACIEVENT	"TRSAVE" padded with blanks.
ACISLAB	Command issuer's SECLABEL.

Table 136 on page 666 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 136. Supported HCPRPIRA Return Codes for the TRSAVE TO Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The command fails with error message HCP003E, indicating an invalid option.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

TRSOURCE Command

Table 137 on page 666 shows the ACIPARMS parameter list for a DAC/audit call for the TRSOURCE command.

Table 137. TRSOURCE Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA size (in bytes)
ACIBMAPA	Class C (X'20') or X'00'.
ACIBMAPP	Class C (X'20') or X'00'.
ACIENV	ACIXAC
ACIRGRP	Command issuer's ACI group name from user's directory entry.

Table 137. TRSOURCE Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIRUSR	Command issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Class C (X'20').
ACISLAB	Command issuer's SECLABEL.
ACIEVENT	"TRSOURCE" padded with blanks.
ACIDATA	Command line.

Table 138 on page 667 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 138. Supported HCPRPIRA Return Codes for the TRSOURCE Command

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is granted and processing continues.
X'08'	The command fails with error message HCP644E, indicating an unknown/unauthorized CP command.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

TRSOURCE ENABLE Command (VMGROUP)

Table 139 on page 667 shows the ACIPARMS parameter list for a MAC call for the TRSOURCE ENABLE command in the case of a VMGROUP.

Table 139. TRSOURCE ENABLE Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACINSSEG
ACILEN	ACISIZE + ACIDISZ (in bytes)
ACIBMAPM	CLASSC (X'20')
ACIENV	ACISEG
ACIRUSR	Command issuer's user ID.
ACIEVENT	"TRSOURCE" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACIVERS	ACIVERS1
ACIFLAG	ACIRW
ACICMDTP	CLASS C (X'20')
ACISEGNM	VMGROUP name.

Table 140 on page 668 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 140. Supported Return Codes

RC	Meaning
X'00'	Authorization is granted, SDFSCLAB is set to the command issuer's SECLABEL, and processing continues.
X'04'	Processing continues.
X'08'	The command fails with error message HCP644E, indicating an unknown/unauthorized CP command.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

VMRELOCATE Command

Table 141 on page 668 shows the ACIPARMS parameter list for an HCPRPIRA call on the destination system for the VMRELOCATE command.

Table 141. VMRELOCATE Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACILOG
ACILEN	ACIRLOSB
ACIBMAPA	ACIANYAU
ACIBMAPP	ACIANYPR
ACIBMAPM	ACIANYMC
ACIENV	ACIXAC
ACIRGRP	ACI group name from the user directory entry of the virtual machine being relocated.
ACIRUSR	User ID of the virtual machine being relocated.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD
ACIEVENT	"RELOCATE" padded with blanks.
ACIVMDBK	Address of the VMDBK for the user being relocated. This field is guaranteed to be valid only until the first loss of control after entry to HCPRPIRA.
ACISLAB	SECLABEL padded with blanks of the guest being relocated, or zeros if no SECLABEL
ACIDSPID	Dispatched user ID
ACIRLODT	"RELOCATE TO"
ACIRLODN	The name of the SSI cluster member that is the target destination of the relocation.
ACIRLOBY	"BY"
ACIRLOCI	The userid of the VMRELOCATE command issuer.
ACIRLOAT	"AT"
ACIRLOIN	The SSI cluster member name where the command issuer was logged on.

VMDUMP TO Command and DIAGNOSE X'94'

This call is controlled by the TRANSFER class G protect setting.

Note: VMDUMP and DIAGNOSE code X'94' are audited generically from the command and DIAGNOSE router respectively.

Table 142 on page 669 shows the ACIPARMS parameter list for an HCPRPIRA DAC call for the VMDUMP and for DIAGNOSE code X'94', when the TO option is used, and the target is not the issuer.

Table 142. VMDUMP TO Command and DIAGNOSE X'94' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACISPOOL
ACILEN	ACISIZE (in bytes)
ACIBMAPP	Transfer's protect settings.
ACIENV	ACITRANS
ACIRGRP	Command issuer's ACI group name.
ACIRUSR	Issuer's user ID.
ACITGRP	ACI group name of the user ID to whom dump is being sent.
ACITUSR	User ID to whom dump is being sent.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD
ACICMDTP	Class G (X'02').
ACIEVENT	"VMDUMP" or "DIAG094" padded with blanks.
ACISLAB	Issuer's SECLABEL.

Table 143 on page 669 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 143. Supported Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues
X'08'	• VMDUMP command The command fails with message HCP003E, indicating an invalid option. • DIAGNOSE code X'94' The DIAGNOSE fails with return code X'6C' and condition code 2.
X'20'	• VMDUMP command The command fails with error message HCP6525E, indicating the ESM is unavailable. • DIAGNOSE code X'94' The DIAGNOSE fails with return code X'6C' and condition code 2.

WARNING Command MAC/AUDIT

See “MESSAGE Command MAC/AUDIT” on page 656.

XAUTOLOG Command

See “AUTOLOG and XAUTOLOG Commands” on page 641.

ACIPARMS Parameter Lists for DIAGNOSE Codes

This section specifies the different ACIPARMS formats that CP diagnose codes pass to the ESM. The calls are made to HCPRPIRA unless otherwise stated.

DIAGNOSE Code X'14'

Table 144 on page 670 shows the ACIPARMS parameter list for a MAC call for subcodes X'04', X'08', X'FFE', and X'FFF', when they return SFBLOK data.

Table 144. DIAGNOSE Code X'14' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDSIZ (in bytes)
ACIBMAPM	ACIDGNMC
ACIENV	ACIXAD
ACIRUSR	DIAGNOSE issuers's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD + ACIRO
ACIEVENT	"DIAG014" padded with blanks.
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACITLAB	SECLABEL of target spool file.

Table 145 on page 670 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 145. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'14'

RC	Meaning
X'00'	Processing continues.
X'04'	Processing continues.
X'08'	The selected spool file is treated as if it does not exist.
X'20'	The selected spool file is treated as if it does not exist.

DIAGNOSE Code X'64'

Table 146 on page 670 shows the ACIPARMS parameter list for the audit call for DIAGNOSE code X'64'. This call does not include DIAGNOSE code X'64' invocations that load or find a restricted segment. “RSTDSEGt” on page 693 shows the ACIPARMS format for LOAD/FIND of a restricted segment.

Table 146. DIAGNOSE Code X'64' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDSIZ (in bytes)
ACIBMAPA	ACIDGNAU
ACIENV	ACISEG
ACIRUSR	DIAGNOSE issuer's user ID.

Table 146. DIAGNOSE Code X'64' Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD|ACIBAD
ACIEVENT	"DIAG064" padded with blanks.
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACISEGNM	DCSS name.
ACIRY	Contents of user's Ry register.

Table 147 on page 671 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 147. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'64'

RC	Meaning
X'00'	Processing continues.
X'04'	Processing continues.
X'20'	Condition code 2 and return code 449 (X'1C1') is returned, indicating the user is not authorized.

DIAGNOSE Code X'68'

Table 148 on page 671 shows the ACIPARMS parameter list for an HCPRPIRA MAC/audit call for DIAGNOSE code X'68'. This call is not made if running on the SYSTEM VMDBK.

Note: This call is only made for the SEND, SEND/RECEIVE, SENDX, RECEIVE, REPLY, and IDENTIFY subfunctions of DIAGNOSE code X'68'. The other subfunctions are not auditable.

Table 148. DIAGNOSE Code X'68' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIVMCSZ (in bytes)
ACIBMAPA	AUDIT setting.
ACIBMAPM	MAC setting.
ACIENV	ACIXAV
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD|ACIBAD + ACIRO (receive)|ACIWO (send, sendx and reply) | ACIRW (send/receive and Iidentify)
ACIRUSR	DIAGNOSE issuer's user ID.
ACITUSR	Target user ID.
ACIEVENT	"DIAG068" padded with blanks.
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACITLAB	DIAGNOSE target userid's SECLABEL, if available.
ACIVMCF	"DIAGNOSE 68 SEND" or "DIAGNOSE 68 SEND/RECEIVE" or "DIAGNOSE 68 SENDX" or "DIAGNOSE 68 RECEIVE" or "DIAGNOSE 68 REPLY" or "DIAGNOSE 68 IDENTIFY".

Table 149 on page 672 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 149. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'68'

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The DIAGNOSE fails with a return code, indicating the target user is not available.
X'20'	The DIAGNOSE fails with a return code, indicating the target user is not available.

DIAGNOSE Code X'88'

Table 150 on page 672 shows the ACIPARMS parameter list for an HCPRPIRA protect/audit call for DIAGNOSE code X'88'.

Table 150. DIAGNOSE Code X'88' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIVMCSZ (in bytes)
ACIBMAPP	PROTECT setting.
ACIBMAPA	AUDIT setting.
ACIENV	ACIXAD
ACIVERS	ACIVERS1
ACIFLAG	• If the user has OPTION DIAG88: ACIANY + ACIGOOD • If the user does not have OPTION DIAG88: ACIANY + ACIBAD
ACIEVENT	"DIAG088" padded with blanks.
ACIRUSR	DIAGNOSE issuer's user ID.
ACIRX	Contents of user's Rx register.
ACIRY	Contents of user's Ry register.
ACISLAB	DIAGNOSE issuer's SECLABEL.

Table 151 on page 672 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 151. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'88'

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.

DIAGNOSE Code X'94' with the TO Option

See “VMDUMP TO Command and DIAGNOSE X'94'” on page 668.

DIAGNOSE Code X'B8'

Table 152 on page 673 shows the ACIPARMS parameter list for an HCPRPIRA MAC/audit call for DIAGNOSE code X'B8' when a spool file is opened. This call is based on the security settings for SPF_OPEN. In addition to this call, DIAGNOSE code X'B8' is audited from the DIAGNOSE router.

Table 152. DIAGNOSE Code X'B8' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT Settings.
ACIBMAPM	MAC Settings.
ACIENV	ACISPF
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIANY + ACIRO (subcode X'00') | ACIWO (subcode X'04')
ACIRUSR	DIAGNOSE issuer's user ID.
ACIEVENT	"SPF_OPEN" padded with blanks.
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACITLAB	Spool file's SECLABEL.
ACISPLID	Spool ID of the owning X'AB' to be opened.
ACICMDIS	Command issuer's user ID.
ACIORIG	Spool file origin user ID.
ACIFSTPG	DASD address of first page.

Table 153 on page 673 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 153. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'B8'

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and CP performs the authorization check.
X'08'	The DIAGNOSE fails with a return code, indicating access is denied.
X'20'	The DIAGNOSE fails with a return code, indicating access is denied.

DIAGNOSE Code X'BC'

Table 154 on page 673 shows the ACIPARMS parameter list for an HCPRPIRA call for the DIAGNOSE code X'BC'.

Table 154. DIAGNOSE Code X'BC' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDISZ (in bytes)
ACIBMAPM	ACIDGNMC

Table 154. DIAGNOSE Code X'BC' Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIENV	ACIXAD
ACIRUSR	DIAGNOSE issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIRO+ACIANY+ACIGOOD
ACIEVENT	"DIAG0BC" padded with blanks.
ACISLAB	Command issuer's SECLABEL.
ACITLAB	SECLABEL of target spool file.
ACIRX	Rx register.
ACIRX1	Rx+1 register.
ACIRY	Ry register.
ACIRY1	Ry+1 register.

Table 155 on page 674 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 155. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'BC'.

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is granted and processing continues.
X'08'	The command fails with key fields having '*' in them.
X'20'	The command fails with key fields having '*' in them.

DIAGNOSE Code X'D4'

Table 156 on page 674 shows the ACIPARMS parameter list for an HCPRPIRA call for DIAGNOSE code X'\D4'. See also “APPC Setting of VMDALTID” on page 679.

Table 156. DIAGNOSE Code X'D4' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIALTU
ACILEN	ACISIZE + ACIDISZ (in bytes)
ACIBMAPA	ACIDGNAU (X'80') if audit on.
ACIBMAPP	ACIDGNPR (X'80') if protect on.
ACIBMAPM	ACIDGNMC (X'80') if MAC on.
ACIENV	ACIXAD
ACIRUSR	DIAGNOSE issuer's (master) user ID.
ACIWUSR	Worker user ID.
ACIAUSR	Alternate (end user) user ID.
ACIVERS	ACIVERS1

Table 156. DIAGNOSE Code X'D4' Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIFLAG	ACIGOOD+ACIRW+ACIANY
ACIEVENT	"DIAG0D4" padded with blanks.
ACIRX	Contents of user's Rx register.
ACIRY	Contents of user's Ry register.
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACIALAB	Alternate user ID's SECLABEL (specified in the DD4ALTSC field of the DIAGNOSE code X'D4' parameter list).
ACITLAB	Worker's user ID's SECLABEL (SECLABEL of the user specified in DD4PALT of the DIAGNOSE code X'D4' parameter list, or zeroes if that user is not logged on).

Table 157 on page 675 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 157. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'D4'

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred, and CP performs the authorization check.
X'08'	The DIAGNOSE fails with return code 12 (in Rx), indicating the issuer is not authorized.
X'20'	The DIAGNOSE fails with return code 12 (in Rx), indicating the issuer is not authorized.

DIAGNOSE Code X'E4'

Table 158 on page 675 shows the ACIPARMS parameter list for an HCPRPIRA call for DIAGNOSE code X'E4'.

Table 158. DIAGNOSE Code X'E4' Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDE4SZ (in bytes)
ACIBMAPA	ACIDGNAU or X'00'
ACIBMAPP	ACIDGNPR or X'00'
ACIENV	ACIDIAE4
ACIRUSR	DIAGNOSE issuer's user ID.
ACITUSR	User ID that owns the minidisk (used for subcodes 0,1, and 2).
ACIMODE	Access mode in EBCDIC (used for subcodes 2 and 3).
ACIADDR	Link-to address in EBCDIC.
ACIVERS	ACIVERS1
ACITADDR	Link-As address in EBCDIC (used for subcodes 2 and 3).
ACIFLAG	ACIANY+ACIGOOD
ACIEVENT	"DIAG0E4" padded with blanks.

Table 158. DIAGNOSE Code X'E4' Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACISUBC	DIAGNOSE subcode in EBCDIC.
ACITCYL	Cylinder number in EBCDIC (used for subcode 3).
ACIETCYL	Extended Address Volume (EAV) cylinder number in EBCIDIC (used for subcode 3).
ACITOD	Additional field passed only for a shareable virtual disk in storage; contains the virtual disk in storage identifier (the TOD stamp for the creation of the virtual disk in storage).

Table 159 on page 676 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA. In cases where CP has detected an error before the ESM call, and the ESM returns an error return code, the CP detected error will be reflected to the user.

Table 159. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'E4'

RC	Meaning
X'00'	Authorization is granted and processing continues. For subcodes 2 and 3, CP performs a directory check for DEVMAINT.
X'04'	Authorization is deferred, and CP performs directory checks (DEVINFO or DEVMAINT for subcodes 0 and 1, DEVMAINT for subcodes 2 and 3).
X'08'	The DIAGNOSE fails with condition code 1 and return code (in Ry) 5 indicating the issuer is not authorized.
X'20'	The DIAGNOSE fails with condition code 1 and return code (in Ry) 5 indicating the issuer is not authorized.

DIAGNOSE Code X'290'

Table 160 on page 676 shows the ACIPARMS parameter list for an HCPRPIRA call for DIAGNOSE code X'290'.

Table 160. DIAGNOSE Code X'290' Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACID290SZ (in bytes)
ACIBMAPA	ACIDGNAU or X'80' if audit on
ACIENV	ACIXAD
ACIRUSR	DIAGNOSE issuer's user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY+ACIGOOD
ACIEVENT	"DIAG290" padded with blanks
ACISLAB	DIAGNOSE issuer's SECLABEL.
ACI290SC	DIAGNOSE X'290' subcode issued
ACI290UI	Target userid
ACI290DN	Target device number (subcode 4 only)

Table 160. DIAGNOSE Code X'290' Command Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACI290Q	Target spool queue (subcode 0 only)
ACI290ID	Target spool file ID (subcode 0 only)

Table 161 on page 677 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 161. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'290'

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.

PERMIT to Address Space MAC (DIAGNOSE Code X'23C' Subcode X'03')

Table 162 on page 677 shows the ACIPARMS parameter list for an HCPRPIRA MAC only call for the PERMIT subfunction of DIAGNOSE code X'23C'. DIAGNOSE code X'23C' is audited generically from the DIAGNOSE router.

Table 162. DIAGNOSE Code X'23C' Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDISZ (in bytes)
ACIBMAPM	ACIDGNMC
ACIENV	ACIXAD
ACIRUSR	User ID of grantee (the user who is being permitted).
ACITUSR	Userid of grantor (the DIAGNOSE issuer).
ACIVERS	ACIVERS1
ACIFLAG	ACIRW or ACIRO + ACIANY
ACIEVENT	"DIAG23C" padded with blanks.
ACISLAB	SECLABEL of grantee (the user who is being permitted).
ACITLAB	SECLABEL of grantor (the DIAGNOSE issuer).
ACIRX	Contents of Rx.

Table 163 on page 677 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 163. Supported HCPRPIRA Return Codes for DIAGNOSE Code X'23C'

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues
X'08'	The DIAGNOSE fails with return code 32, indicating the issuer is not authorized.
X'20'	The DIAGNOSE fails with return code 32, indicating the issuer is not authorized.

ACIPARMS Parameter Lists for System Functions

This section specifies the different ACIPARMS formats that CP system functions pass to the ESM. The calls are made to HCPRPIRA unless otherwise stated.

APPC CONNECT

Table 164 on page 678 shows the ACIPARMS format for an HCPRPIRA call for MAC and audit of APPC connect.

Table 164. APPC CONNECT Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACICONSZ (in bytes)
ACIBMAPA	AUDIT Setting
ACIBMAPM	MAC Setting
ACIENV	ACICNCT
ACIRUSR	Effective Source - on whose behalf the connect is being done: • VMDUSER if non-communication server without alternate user ID • CONALTID if non-communication server with alternate user ID • IPVMID if communication server issued connect
ACITUSR	Target user ID (Specified resource ID)
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD|ACIBAD + ACIRW **Note:** ACIRW will not be set in error (audit-only) cases.
ACIEVENT	"APPCCON" padded with blanks
ACISLAB	Source SECLABEL (SECLABEL of user in ACIRUSR)
ACITLAB	Target SECLABEL if the target is logged on **Note:** This field may not be set on audit-only error cases.
ACIPATH	APPC path ID **Note:** This field may not be set on audit-only error cases.
ACIQUAL	LUNAME qualifier from the IPARMLX
ACITLUN	Target LU from the IPARMLX
ACISERVR	Local source - user ID of the virtual machine who issued the connect

If CP processing detected an error before the ESM call, the call will be made for auditing purposes only and the ESM return code is not checked. The IPRCODE for the connect will be set to the CP-detected error code. Otherwise, the ESM return codes are as displayed in Table 165 on page 678.

Table 165. Supported HCPRPIRA Return Codes for APPC CONNECT

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.

Table 165. Supported HCPRPIRA Return Codes for APPC CONNECT (continued)

RC	Meaning
X'08'	The connect fails. IPRCODE=IPRCBADR.
X'20'	The connect fails. IPRCODE=IPRCBADR.

APPC Setting of VMDALTID

Table 166 on page 679 shows the ACIPARMS format for an HCPRPIRA call for audit and authorization for setting an alternate userid. This call is based off the security settings of DIAGNOSE code X'D4'.

Table 166. APPC setting of VMDALTID format of ACIPARMS

Label	Contents
ACIFCN	ACIALTU
ACILEN	ACISIZE + ACIDISZ (in bytes)
ACIBMAPA	ACIDGNAU (X'80') if audit on
ACIBMAPP	ACIDGNPR (X'80') if protect on
ACIBMAPM	ACIDGNMC (X'80') if MAC on
ACIENV	ACIXAD
ACIRUSR	Issuer's' user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD|ACIBAD + ACIANY + ACIRW
ACIEVENT	"DIAG0D4" padded with blanks
ACISLAB	Requestor's SECLABEL
ACITLAB	SECLABEL of user ID specified in IPXALTID if that user is logged on **Note:** This field is not set if ACIBAD is on in ACIFLAG.
ACIALAB	SECLABEL of user ID specified in IPXALTID if that user is logged on **Note:** This field is not set if ACIBAD is on in ACIFLAG.
ACIWUSR	Resource ID (IPRESID), target user ID (IPXTRGLU) or gateway ID (IPXLQUAL) **Note:** This field is not set if ACIBAD is on in ACIFLAG.
ACIAUSR	Altid to be set up (IPXALTID) **Note:** This field is not set if ACIBAD is on in ACIFLAG.

Table 167 on page 679 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 167. Supported Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The connect fails. IPRCODE=IPRCNAUT.
X'20'	The connect fails. IPRCODE=IPRCNAUT.

APPC SEVER

Table 168 on page 680 shows the ACIPARMS format for an APPC SEVER.

Table 168. APPC SEVER Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISEVSZ (in bytes)
ACIBMAPA	ACISYSAU
ACIENV	ACISEVER
ACIRUSR	Issuer's user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD
ACIEVENT	"APPCSEV" padded with blanks
ACISLAB	Requestor's SECLABEL
ACIPATH	APPC path ID

There is no ESM return code checking.

APPCPWVL

Table 169 on page 680 shows the ACIPARMS format for an HCPRPIRA call for audit and authorization for APPC CONNECT with password validation. This call is controlled by the APPCPWVL security settings.

Table 169. APPC connect with password validation

Label	Contents
ACIFCN	ACILOG if calling for authorization; otherwise, ACIVMCMD
ACILEN	ACISIZE + ACIPWSZ (in bytes)
ACIBMAPA	ACISYSAU (X'80') if audit on
ACIBMAPP	ACISYSPR (X'80') if protect on
ACIENV	ACIAPPW
ACIRGRP	Blanks
ACIRUSR	Userid whose password is being checked
ACITUSR	Local source (userid who issued the connect)
ACILGOPT	ACIVAL if calling for authorization; otherwise, zeroes
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD|ACIBAD
ACIEVENT	"APPCPWVL" padded with blanks
ACIPWLEN	Length of password (not included in some audit-only error cases)
ACIPSWRD	Password (masked) (not included in some audit-only error cases)

Table 170 on page 681 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 170. Supported Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and the CP directory password is checked.
X'08'	The connect fails. IPRCODE=IPRCNAUT.
X'20'	The connect fails. IPRCODE=IPRCNAUT.

Directory Command Audit-Only Call

Table 171 on page 681 shows the ACIPARMS format for an audit-only call for a command executed automatically upon logon through the use of the COMMAND directory control statement. (See the COMMAND directory control statement in *z/VM: CP Planning and Administration* for more information.)

Table 171. Directory Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIDATA (in bytes)
ACIBMAPA	Audit setting for each privilege class or ACIANYU if this is an 'ANY' class command.
ACIENV	ACICPAUD
ACIRUSR	Command issuer's user ID.
ACIRGRP	Command issuer's ACI group name.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD | ACIBAD + ACIANY (if this is an 'ANY' class command)
ACICMDTP	Command version. **Note:** This field is not filled in for 'ANY' class commands.
ACISLAB	Command issuer's SECLABEL.
ACIEVENT	'DIRECTRY_CMD'
ACIDATA	'TO userid: ' followed by command line.

Table 172 on page 681 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 172. Supported Return Codes for a Directory Command

RC	Meaning
X'00'	Processing continues.
X'04'	Processing continues.
X'20'	The command fails with error message HCP6525E, indicating the ESM is unavailable.

IUCV CONNECT

Table 173 on page 682 shows the ACIPARMS parameter list for the HCPRPIRA call for MAC and audit of an IUCV CONNECT.

Table 173. IUCV CONNECT Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACIUCVSZ (in bytes)
ACIBMAPA	AUDIT Setting
ACIBMAPM	MAC Setting
ACIENV	ACICNCT
ACIRUSR	Issuer's user ID
ACITUSR	Target user ID of the connect (from IPVMID)
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD|ACIBAD + ACIRW
ACIEVENT	"IUCVCON" padded with blanks
ACISLAB	Command issuer's SECLABEL
ACITLAB	Target user ID's SECLABEL, if the target is logged on. If the target is a system service, then ACITLAB will be set to "SYSNONE."
ACIPATH	IUCV path ID

If CP processing detected an error before the ESM call, the call will be made for auditing purposes only and the ESM return code is not checked. The IPRCODE for the connect will be set to the CP-detected error code. Otherwise, the ESM return codes are as displayed in Table 174 on page 682.

Table 174. Supported HCPRPIRA Return Codes for IUCV CONNECT

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	The ESM defers any judgement on the user's status. Processing continues.
X'08'	The connect fails. IPRCODE=IPRCBADR.
X'20'	The connect fails. IPRCODE=IPRCBADR.

IUCV SEVER

Table 175 on page 682 shows the ACIPARMS format for an IUCV SEVER.

Table 175. IUCV SEVER Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISEVSZ (in bytes)
ACIBMAPA	ACISYSAU
ACIENV	ACISEVER
ACIRUSR	Issuer's user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD
ACIEVENT	"IUCVSEV" padded with blanks

Table 175. IUCV SEVER Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACISLAB	Requestor's SECLABEL
ACIPATH	IUCV path ID

There is no ESM return code checking.

MAINTCCW Audit

Table 178 on page 684 shows the ACIPARMS parameter list for an HCPRPIRA audit call for the MAINTCCW function. This will audit the issuances of diagnostic CCW's. The audit call will be made only once per channel program.

Table 176. MAINTCCW Format of ACIPARMS

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACICCWSZ (in bytes)
ACIBMAPA	ACISYSAU
ACIENV	ACIXACCW
ACIRUSR	Diagnostic CCW issuer's user ID
ACIADDR	Virtual device number
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD
ACIEVENT	"MAINTCCW" padded with blanks
ACISLAB	Diagnostic CCW issuer's SECLABEL
ACISCYL	Starting cyl/blk on the DASD
ACIECYL	Ending cyl/blk on the DASD
ACIRDEV	Real device number
ACIVOLSR	Volume serial

Table 177 on page 683 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 177. Supported HCPRPIRA Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'20'	The argument of keyword INDICATOR is set to "NO."

MDISK Command

Table 178 on page 684 shows the ACIPARMS parameter list for an HCPRPIRA audit/DAC/MAC call for the MDISK function.

Note: The MDISK function, and not LINK, controls LINKs-to-self.

If after making the call and receiving ESM authorization, CP denies or downgrades the requested access, then an audit-only call will be made. The ACIPARMS format for this audit-only call will be the same as below and:

- ACIBMAPP and ACIBMAPM will be zeroes
- ACIMODE will be 'RR' if the access has been downgraded to read and 'XX' if the access has been denied

Table 178. MDISK Command Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACILINK
ACILEN	ACISIZE (in bytes)
ACIBMAPA	Class G (X'02'). or X'00'
ACIBMAPM	Class G (X'02'). or X'00'
ACIBMAPP	Class G (X'02'). or X'00'
ACIENV	ACIDISK
ACIRGRP	Command issuer's ACI group name from user's directory entry
ACIRUSR	Command issuer's user ID
ACITGRP	ACI group name of the minidisk owner user ID (same as ACIRGRP in this case)
ACITUSR	Owning user ID of the minidisk that is being linked to (same as ACIRUSR in this case)
ACIMODE	Minidisk access mode in EBCDIC format
ACIADDR	Link-To address in EBCDIC format
ACIVERS	ACIVERS1
ACITADDR	Link-As address in EBCDIC format
ACIFLAG	ACIRO or ACIRW
ACICMDTP	Class G (X'02')
ACIEVENT	"MDISK" padded with blanks
ACISLAB	Command issuer's SECLABEL

Table 179 on page 684 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 179. Supported HCPRPIRA Return Codes for MDISK

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred, and CP performs the authorization check if needed.
X'08'	• If ACIRSNCD=0, then the command fails and message HCP298E '*userid vdev* not linked; request denied.' is issued. • If ACIRSNCD=ACISD and access mode 'WR' or 'MR' was requested, then the R/O access is granted and message HCP1156I 'DASD *device number* forced R/O; unauthorized for R/W' is issued.
X'20'	The command fails with an error message.

POSIX Set ID Functions

Table 180 on page 685 shows the ACIPARMS parameter list for the call to HCPRPIRA. CP will not invoke the ESM for POSIX set ID requests unless the ESM has previously informed CP that it contains POSIX support. See "Entry Point HCPRPEPX - Notify CP of POSIX capabilities" on page 602 for additional information.

Table 180. POSIX Set ID Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACISETID
ACILEN	ACISIDSB
ACIBMAPP	ACIDGNPR
ACIENV	One of the following: ACISETUI, ACISETEU, ACISETGI, ACISETEG, ACIEXEC, ACINWGRP, or ACISETSG.
ACIRGRP	Access control group name from the user directory entry of the user identified in ACIRUSR.
ACISLAB	SECLABEL of the user identified in ACIRUSR.
ACIRUSR	Issuer's user ID (user whose IDs are to change).
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIANY
ACIFLAG2	Flags, as follows: • ACIPXUSN if supplying a group name in ACINGNAM (as opposed to a GID in ACINGID).
ACIEVENT	"POSIXSETID", left-justified and padded with blanks.
ACIORUID	Old (current) real UID.
ACIOEUID	Old (current) effective UID.
ACIOSUID	Old (current) saved set-UID.
ACIORGID	Old (current) real GID.
ACIOEGID	Old (current) effective GID.
ACIOSGID	Old (current) saved set-GID.
ACINUID	New UID (equal to ACIORUID if UID not being changed).
ACINGID	If ACIPXUSN is off, the new GID (equal to ACIORGID if GID not being changed); otherwise, not used as input to the ESM.
ACINGNAM	If ACIPXUSN is on, the group name that identifies the new GID; otherwise, not used. May be in mixed case.
ACIOSGCT	If ACIENV=ACISETSG, count of old (current) SGIDs; otherwise, not used.
ACIOSGLS	If ACIENV=ACISETSG, address of old SGID list; otherwise, not used.
ACINSGCT	If ACIENV=ACISETSG, count of new SGIDs; otherwise, not used.
ACINSGLS	If ACIENV=ACISETSG, address of new SGID list; otherwise, not used.

Table 181 on page 686 shows the output fields set by the ESM in ACIPARMS on the return from HCPRPIRA.

Table 181. POSIX Set IDs ESM output fields in ACIPARMS

Label	Contents
ACIFLAG2	Flags, as follows: • ACIPXALL indicates that all three (real, effective and saved set) POSIX UIDs and/or GIDs are to be set to the new value. ACIPXALL is only valid when ACIENV is either ACISETUI or ACISETGI.
ACINGID	If ACIPXUSN is on, the GID associated with the input group name passed in ACINGNAM.

Table 182 on page 686 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 182. Supported HCPRPIRA Return Codes for POSIX Set ID Functions

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues. CP performs authorization checking.
X'08'	Authorization is denied. If invoked during DIAGNOSE code X'29C', it fails with return code SPXEAUTH (5).
X'20'	The ESM is unavailable. If invoked during DIAGNOSE code X'29C', it fails with return code SPXEDBAS (9).
X'24'	The input group does not exist. If invoked during DIAGNOSE code X'29C', it fails with return code SPXENFND (6).
X'2C'	ACINUID or ACINGID contains an invalid value. If invoked during DIAGNOSE code X'29C', it fails with return code SPXEID (8).

POSIX Group Database Query Function

This function returns data from the POSIX group database. Table 183 on page 686 shows the ACIPARMS parameter list for the call to HCPRPIRA. CP will not invoke the ESM for POSIX database queries unless the ESM has previously informed CP that it contains POSIX support. See “Entry Point HCPRPEPX - Notify CP of POSIX capabilities” on page 602 for additional information.

Table 183. Query POSIX Group Database Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIQGDB
ACILEN	ACIGB1SB + length of ACIGBUFL buffer list (in bytes)
ACIBMAPP	ACIDGNPR
ACIENV	ACIPQGDB
ACIRGRP	Access control group name from the user directory entry of the user identified in ACIRUSR, if any; otherwise, zeros. If ACIRUSR is “*SYSTEM*”, ACIRGRP is set to zeros.
ACISLAB	SECLABEL of the user identified in ACIRUSR, if any; otherwise, zeros. If ACIRUSR is “*SYSTEM*”, ACISLAB is set to “SYSHIGH”.
ACIRUSR	If ACIPXSYS is off, issuer's user ID; otherwise, “*SYSTEM*”. The data must be left-justified and padded with blanks.

Table 183. Query POSIX Group Database Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACITUSR	If ACIPXQGM is on, user whose group membership is being queried; otherwise, zeros.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIANY
ACIFLAG2	Flags, as follows: • ACIPXUSN if supplying a group name in ACIGGNAM (as opposed to a GID in ACIGGID) • ACIPXSYS if this is a CP-initiated query (as opposed to a user-initiated query). No authority checking should be performed. • ACIPXIDS if the query was initiated by a POSIX process. ACIGRUID, ACIGEUID, ACIGSUID, ACIGRGID, ACIGEGID and ACIGSGID are filled in. • ACIPXQGM if this is a query to determine if ACITUSR is a member of the group identified in ACIGGNAM/ACIGGID.
ACIEVENT	“POSIXGROUPDB”, left-justified and padded with blanks
ACIGRUID	If ACIPXIDS is on, real UID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGEUID	If ACIPXIDS is on, effective UID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGSUID	If ACIPXIDS is on, saved set-UID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGRGID	If ACIPXIDS is on, real GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGRGID	If ACIPXIDS is on, real GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGEGID	If ACIPXIDS is on, effective GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGSGID	If ACIPXIDS is on, saved set-GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIGGID	If ACIPXUSN is off, GID of the POSIX group for which information is to be returned
ACIGGNAM	If ACIPXUSN is on, the group name of the POSIX group for which information is to be returned. May be in mixed case.
ACIGBUFL	Group database buffer list. A list of contiguous entries describing the output buffers for this request. Each entry consists of a buffer address and length (in bytes). ACILEN can be used to calculate the number of entries in the list. The entries describe the buffers to contain the only data item requested via this interface, the list of group members.

Table 184 on page 687 shows the output fields set by the ESM in ACIPARMS on the return from HCPRPIRA.

Table 184. Query POSIX group database output fields in ACIPARMS

Label	Contents
ACIGGID	If ACIPXUSN is on, the GID corresponding to the input group name
ACIGGNAM	If ACIPXUSN is off, the group name corresponding to the input GID

Table 184. Query POSIX group database output fields in ACIPARMS (continued)

Label	Contents
ACIGMCNT	Count of members in the group.
ACIGBUFL	Group database buffer list. If ACICODE = X'00', the user names of the members of the input group have been placed in the buffers and the length field of each entry contains the length, in bytes, of the data placed in the buffer pointed to by that entry. The data consists of contiguous user names that are left-justified and padded on the right with blanks to a length of 8 characters each. If there are no user names to be returned, then the length fields in the group buffer list must be set to zero. Before returning this list to the requestor, CP will translate all user names to lower case. If ACICODE = X'28', the length field of the first entry contains the length, in bytes, of the buffer necessary to contain the entire member list. If ACICODE is not X'00' or X'28', the buffer list is unchanged.

Table 185 on page 688 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 185. Supported HCPRPIRA Return Codes for POSIX group database query

RC	Meaning
X'00'	Authorization is granted. If requested, the buffers contain the member list, and the buffer length fields have been updated accordingly. If ACIPXQGM was specified on input, ACITUSR is a member of the group identified in ACIGGNAM/ACIGGID. CP will acquire any deferred data items from the directory or use default values, without performing any authorization checks.
X'04'	Authorization is deferred and processing continues. CP performs authorization checking and provides the database information to authorized requestors.
X'08'	Authorization is denied. If invoked during DIAGNOSE code X'2A0', it fails with return code QPXEAUTH (5).
X'20'	The ESM is unavailable. If invoked during DIAGNOSE code X'2A0', it fails with return code QPXEDBAS (9).
X'24'	The input group does not exist. If invoked during DIAGNOSE code X'2A0', it fails with return code QPXENFND (6).
X'28'	Authorization is granted, but the buffers provided on input are too small to contain the entire group member list. The length field of the first ACIGBUFL entry contains the total required buffer length, in bytes. CP can be expected to acquire larger buffers and request all of the data from the ESM again.
X'30'	ACITUSR is not a member of the group identified in ACIGGNAM/ACIGGID. This ACICODE is only possible when ACIPXQGM was specified on input.

POSIX User Database Query Function

This function returns data from the POSIX user database. Table 186 on page 688 shows the ACIPARMS parameter list for the call to HCPRPIRA. CP will not invoke the ESM for POSIX database queries unless the ESM has previously informed CP that it contains POSIX support. See “Entry Point HCPRPEPX - Notify CP of POSIX capabilities” on page 602 for additional information.

Table 186. User Database Query Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIQUDB

Table 186. User Database Query Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACILEN	ACIUDBSB + length of ACIUBUFL buffer list (in bytes)
ACIBMAPP	ACIDGNPR
ACIENV	ACIPQUDB
ACIRGRP	Access control group name from the user directory entry of the user identified in ACIRUSR, if any; otherwise, zeros. If ACIRUSR is “*SYSTEM*”, ACIRGRP is set to zeros.
ACISLAB	SECLABEL of the user identified in ACIRUSR, if any; otherwise, zeros. If ACIRUSR is “*SYSTEM*”, ACISLAB is set to “SYSHIGH”.
ACIRUSR	If ACIPXSYS is off, issuer's user ID; otherwise, “*SYSTEM*”. The data must be left-justified and padded with blanks.
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIANY
ACIFLAG2	Flags, as follows: • ACIPXUSN if supplying a user name in ACIUUNAM (as opposed to a UID in ACIUUID). • ACIPXSYS if this is a CP-initiated query (as opposed to a user-initiated query). No authority checking should be performed. • ACIPXIDS if the query was initiated by a POSIX process. ACIURID, ACIUEID, ACIUSUID, ACIURGID, ACIUEGID and ACIUSGID are filled in.
ACIEVENT	“POSIXUSERDB”, left-justified and padded with blanks
ACIURUID	If ACIPXIDS is on, real UID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIUEUID	If ACIPXIDS is on, effective UID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIUSUID	If ACIPXIDS is on, saved set-UID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIURGID	If ACIPXIDS is on, real GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIUEGID	If ACIPXIDS is on, effective GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIUSGID	If ACIPXIDS is on, saved set-GID of the process requesting the data. Otherwise, unpredictable and should not be used.
ACIUUNAM	If ACIPXUSN is on, user name (user ID) for which information is to be returned. Must be in upper case.
ACIUUID	If ACIPXUSN is off, UID for which information is to be returned
ACIUBUFL	User database buffer list. A list of contiguous entries describing the output buffers for this request. Each entry consists of a buffer address and length (in bytes). A buffer address of zero indicates that the corresponding database information, and its length, should not be returned. ACILEN can be used to calculate the number of entries in the list. The address and length field of each entry is described below.
ACIUIWDA	Address of the buffer to contain the user's initial working directory
ACIUIWDL	Length of the buffer pointed to by ACIUIWDA

Table 186. User Database Query Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIUIUPA	Address of the buffer to contain the user's initial user program
ACIUIUPL	Length of the buffer pointed to by ACIUIUPA
ACIUFSRA	Address of the buffer to contain the user's file system root
ACIUFSRL	Length of the buffer pointed to by ACIUFSRA
ACIUSGIA	Address of the buffer to contain the user's supplementary GID list
ACIUSGIL	Length of the buffer pointed to by ACIUSGIA

Table 187 on page 690 shows the output fields set by the ESM in ACIPARMS on the return from HCPRPIRA.

Table 187. Query POSIX user database output fields in ACIPARMS

Label	Contents
ACIUUNAM	If ACIPXUSN is off, the user name (user ID) corresponding to the input UID. CP will translate this to lower case before returning it to the requestor.
ACIUUID	If ACIPXUSN is on, the UID corresponding to the input user name (user ID).
ACIUGNAM	Group name of the user's primary POSIX group. May be in mixed case.
ACIUGID	GID of the user's primary POSIX group.
ACIUBUFL	User database buffer list. If ACICODE = X'00', the high-order bit of each length field indicates whether the ESM has chosen to provide the corresponding data or to defer it to CP. If this bit is on, the data that the ESM wishes to provide has been placed in the buffers, and the length field of the entry contains the length, in bytes, of the data placed in the buffer pointed to by that entry. If this bit is off, CP will use its own data (usually from the user directory or a default value). CP preserves the case of the data pointed to by ACIUIWDA, ACIUIUPA and ACIUFSRA. These strings must be no longer than 1023 characters. If the Supplementary GID list was returned, it must include the user's primary GID. It is recommended, but not required, that the primary GID be the first one in the list. This list must contain no more than {NGROUPS_MAX} GIDs. If ACICODE = X'28', the length field of each entry whose high-order bit of the length field is on contains the length, in bytes, of the buffer necessary to contain the corresponding database information. If ACICODE is not X'00' or X'28', the buffer list is unchanged.

Table 188 on page 690 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 188. Supported HCPRPIRA Return Codes for user database query

RC	Meaning
X'00'	Authorization is granted. The ACIUBUFL entries indicate whether the buffers contain the requested user database information. The buffer length fields have been updated accordingly. CP will acquire any deferred data items from the directory or use default values, without performing any authorization checks.
X'04'	Authorization is deferred and processing continues. CP performs authorization checking and provides the database information to authorized requestors.

Table 188. Supported HCPRPIRA Return Codes for user database query (continued)

RC	Meaning
X'08'	Authorization is denied. If invoked during DIAGNOSE code X'2A0', it fails with return code QPXEAUTH (5).
X'20'	The ESM is unavailable. If invoked during DIAGNOSE code X'2A0', it fails with return code QPXEDBAS (9).
X'24'	The input user does not exist. If invoked during DIAGNOSE code X'2A0', it fails with return code QPXENFND (6).
X'28'	Authorization is granted for all requested data, but one or more of the buffers provided on input is too small to contain the requested data. The length field of each ACIUBUFL entry whose high-order bit of the length field is on contains the required buffer length, in bytes, for that item. CP can be expected to acquire larger buffers and request all of the data from the ESM again.

Resource Access Authorization Check

This function returns data from a resource access authorization check. It can be called by any part of CP and operates similarly to a RACROUTE REQUEST=AUTH call.

If access is denied or deferred (ACICODE = ACINOAC or ACIDEFR), the ACIREASON label will contain an additional explanation. For more information on class and resource names and return and reason codes, see *z/VM: Security Server RACROUTE Macro Reference*.

Table 189 on page 691 shows the ACIPARMS parameter list for a call to HCPRPIRA.

Table 189. Resource Access Authorization Check Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIRSCHK
ACILEN	The length of ACIPARMS (in bytes).
ACIVERS	ACIVERS1
ACIBMAPP	Protect settings.
ACIENV	Event type.
ACIFLAG	ACIGOOD or ACIBAD, and ACIANY if required.
ACIEVENT	Command, diagnose, or function name.
ACIRUSR	User ID that issued the command.
ACITUSR	User ID that is the target of the command (optional). If set, the target user's access to the specified resource will be verified. Otherwise, the issuing user's access will be verified.
ACISLAB	SECLABEL of the user identified in ACIRUSR.

Table 189. Resource Access Authorization Check Format of the ACIPARMS Parameter List (continued)

Label	Contents
ACIACCESS	Either: • ACIACQUERY to query the highest permitted level of access. • ACIACREAD to verify read-only access. • ACIACWRITE to verify write access. • ACIACPRIV to verify privileged access. • ACIACFULL to verify full access to the resource. **Note:** If ACIACQUERY is specified, this field will be overlaid with the highest permitted access if the return code is zero.
ACICLASS	Resource class name, padded on the right with blanks.
ACILOGGING	Either: • ACILOGPERESM where ESM audit settings are honored (default). • ACILOGNOFAIL to disable authorization failure audits. • ACILOGNONE to disable the creation of an audit record. • ACILOGNOSTAT is the same as ACILOGNONE, but without updating access statistics.
ACIRESNAME	Resource name.
ACIRESNAMELEN	Length of the resource name (in bytes).
ACILOGDATA	Caller-defined string to be included in the audit record.
ACILOGDATALEN	Length of the string in ACILOGDATA (in bytes).

Table 190 on page 692 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 190. Supported HCPRPIRA Return Codes for Resource Access Authorization Check

RC	Meaning
X'00'	Authorization is granted or deferred and processing continues.
X'08'	Authorization is denied.
X'0C'	Authorization failed.
X'10'	Password is expired.
X'20'	The ESM is unavailable.

Table 191 on page 692 shows the list of class and resource names used by CP.

Table 191. Class and Resource Names Used by CP for Resource Access Authorization Check

Class Name	Resource Name	Used By
SURROGAT	LOGONBY.*userid*	• DIAGNOSE X'88' subcode 8 • FOR command
VMDEV	RDEV.*nnnn*, where *nnnn* is the 4-digit hexadecimal device number.	• DEDICATE command • ATTACH command • GIVE command

RSTDSEGt

Table 192 on page 693 shows the ACIPARMS parameter list for an HCPRPIRA call for LOAD and FIND of a RESTRICTED NSS/DCSS. This function can be invoked using DIAGNOSE code X'64' or the IPL command.

Table 192. RSTDSEG format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	Either: • ACIDCSEG for DCSS's • ACINSSEG for NSS's
ACILEN	ACISIZE + ACIDSIZ (in bytes)
ACIBMAPA	X'00' or X'80' (ACISYSAU)
ACIBMAPP	X'00' or X'80' (ACISYSPR)
ACIBMAPM	X'00' or X'80' (ACISYSMC)
ACIENV	ACISEG
ACIRUSR	Command issuers user ID.
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIRW|ACIRO + ACIGOOD|ACIBAD
ACIEVENT	"RSTDSEG" padded with blanks
ACISLAB	Command issuer's SECLABEL
ACISEGNM	DCSS name
ACIRY	Contents of user's Ry register (for calls invoked via DIAGNOSE code X'64' only)

Table 193 on page 693 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 193. Supported HCPRPIRA Return Codes for RSDTSEG.

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred, and CP performs the authorization check.
X'08'	If invoked by DIAGNOSE code X'64', condition code 2 and return code 449 (X'1C1') is returned, indicating user is not authorized. If invoked by the IPL command, the command fails. Message HCP449E 'Your userid is not authorized to IPL system *name*' is issued.
X'20'	If invoked by DIAGNOSE code X'64', condition code 2 and return code 449 (X'1C1') is returned, indicating user is not authorized. If invoked by the IPL command, the command fails. Message HCP449E 'Your userid is not authorized to IPL system *name*' is issued.

SCIF Event Audit

Table 194 on page 694 shows the ACIPARMS parameter list for an HCPRPIRA call for the audit of a SCIF event.

Table 194. SCIF Event Audit Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE+ACIVMCSZ (in bytes)
ACIBMAPA	ACISYSAU
ACIENV	ACISCIF
ACIRUSR	Primary user ID
ACITUSR	Secondary user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIANY+ACIGOOD or ACIANY+ACIBAD
ACISLAB	Primary user's SECLABEL
ACITLAB	Secondary user's SECLABEL
ACIEVENT	'SCIF' padded with blanks
ACIVMCF	'SCIF EVENT- CHECKED' or 'SCIF EVENT- UNCHECKED'

There is no return code checking performed after the ESM call.

SCIF Event MAC Check

Table 195 on page 694 shows the ACIPARMS parameter list for an HCPRPIRA call for the MAC check of a SCIF event.

Table 195. SCIF Event MAC Check Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD Default function code
ACILEN	ACISIZE (in bytes)
ACIBMAPM	ACISYSMC
ACIENV	ACISCIF
ACIRUSR	Secondary user ID
ACITUSR	Primary user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIANY+ACIRO+ACIGOOD
ACISLAB	Secondary user's SECLABEL
ACITLAB	Primary user's SECLABEL
ACIEVENT	'SCIF' padded with blanks

Table 196 on page 694 shows the ACICODE return codes checked on return from HCPRPIRA.

Table 196. Supported Return Codes

RC	Meaning
<= X'04'	Authorization is granted and processing continues.

Table 196. Supported Return Codes (continued)

RC	Meaning
> X'04'	The command fails with error message HCP6768I, indicating SECUSER or observation is not functional.

SPF_OPEN, SPF_CREATE, SDF_OPEN, SDF_CREATE

Table 197 on page 695 ACIPARMS format for an audit call for SPF_CREATE and SDF_CREATE and for a MAC/AUDIT call for SPF_OPEN and SDF_OPEN. These system functions can be invoked in various ways, for example DIAGNOSE codes X'14', X'34', and X'E0'.

Note: MAC checking will not be performed when running on the system VMDBK.

Table 197. SPFOPEN Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT Setting
ACIBMAPM	MAC Setting (only for SPF_OPEN and SDF_OPEN)
ACIENV	ACISPF
ACIRUSR	Spool file owner's user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIANY + ACIGOOD + ACIRO
ACIEVENT	"SPF_OPEN", "SDF_OPEN", "SPF_CREATE", "SDF_CREATE"
ACISLAB	Command issuer's SECLABEL If running on the system VMDBK, system VMDBK's SECLABEL
ACITLAB	Spool file's SECLABEL
ACIORIG	Spool file origin user ID
ACIFSTPG	DASD address of first page
ACISPLID	Spool ID of file to be opened
ACICMDIS	Command issuer's user ID

Table 198 on page 695 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 198. Supported HCPRPIRA Return Codes

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.
X'08'	The function fails and HCPSFSOR returns R15=12 and CC=3 to the caller, indicating access is denied.
X'20'	The function fails and HCPSFSOR returns R15=12 and CC=3 to the caller, indicating access is denied.

SPF_DELETE and SDF_DELETE audit

Table 199 on page 696 ACIPARMS format for SPF_DELETE and SDF_DELETE.

Table 199. ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT Setting (X'80' or X'00')
ACIENV	ACISPF
ACIRUSR	Spool file owner's user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIANY
ACIEVENT	"SPF_DELETE" or "SDF_DELETE" padded with blanks
ACISLAB	Issuer's SECLABEL
ACITLAB	Spool file's SECLABEL
ACIFSTPG	DASD address of first page
ACISPLID	Spool ID of file to be deleted
ACIORIG	Spool file origin user ID
ACICMDIS	Command issuer's user ID

There is no return code checking done on this audit call.

SNIFFER_MODE Function

Table 200 on page 696 shows the ACIPARMS parameter list for the HCPRPIRA call to audit promiscuous mode.

Table 200. ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	Length of parameter list in bytes
ACIBMAPA	AUDIT Setting
ACIENV	ACISNIF
ACIRGRP	Command issuer's ACI group name
ACIRUSR	Requester's user ID
ACITUSR	Guest LAN or virtual switch owner
ACILFLAG	Promiscuous mode on or off (ACILGOPT)
ACIVERS	ACIVERS1
ACINODE	Guest LAN or virtual switch name (ACILNID)
ACITADDR	Requester's VDEV address
ACIFLAG	ACIGOOD or ACIBAD

Table 200. ACIPARMS Parameter List (continued)

Label	Contents
ACIEVENT	SNIFFER_MODE
ACISLAB	Requester's SECLABEL
ACILVIDL	VLAN ID buffer size
ACILVIDA	List of authorized VLAN IDs

Table 201 on page 697 shows the return codes the CP supports in ACICODE on the return from HCPRPIRA.

Table 201. Supported HCPRPIRA Return Codes for a Promiscuous Mode Audit

RC	Meaning
X'00'	Successful audit
X'20'	ESM was unable to process the request

UTLPRINT Function

Table 202 on page 697 shows the ACIPARMS parameter list for the HCPRPIRA.

Table 202. PRINT Format of the ACIPARMS Parameter List

Label	Contents
ACIFCN	ACIVMCMD
ACILEN	ACISIZE + ACISPLSZ (in bytes)
ACIBMAPA	AUDIT Setting
ACIBMAPM	MAC Setting
ACIENV	ACISPF
ACIRUSR	File owner's user ID
ACIVERS	ACIVERS1
ACIFLAG	ACIGOOD + ACIRO + ACIANY
ACIEVENT	"UTLPRINT" padded with blanks
ACITLAB	SECLABEL of the file being printed
ACIORIG	Spool file origin ID
ACISPLID	Spool file ID
ACIFSTPG	Starting DASD address

Table 203 on page 697 shows the return codes that CP supports in ACICODE on the return from HCPRPIRA.

Table 203. Supported HCPRPIRA Return Codes for PRINT

RC	Meaning
X'00'	Authorization is granted and processing continues.
X'04'	Authorization is deferred and processing continues.

Table 203. Supported HCPRPIRA Return Codes for PRINT (continued)

RC	Meaning
X'08'	The file is placed in system HOLD status, and the following messages are issued: • HCP356E 'Access denied; User *userid* file *spoolid* not printed.' • HCP1561E 'User *userid*'s file *spoolid* is held.'
X'20'	The printer is drained, the file is requeued, and the following messages are issued: • HCP2514E 'Printer *rdev* was drained because the external security manager is unavailable.' • HCP1561E 'User *userid*'s file *spoolid* is requeued.'

Chapter 12. Account System Service (*ACCOUNT)

An installation may write an application to run in a guest virtual machine that has the authorization to use the IUCV interface to receive accounting records from the z/VM control program supporting it. This IUCV authorization is defined within the IUCV directory control statement of the guest virtual machine. The IUCV control statement must name **ACCOUNT* as the CP system service to which a communication path will be established. The user ID of the guest virtual machine may also be identified to the control program during system generation so that records can be accumulated for the virtual machine before it has connected to the system service.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

The Account system service (*ACCOUNT) in CP supports both 1-way and 2-way IUCV protocols when sending records to authorized virtual machines. When a **1-way** IUCV SEND function is issued by CP, the virtual machine to which the accounting record is sent cannot issue an IUCV REPLY function but must issue a RECEIVE function. When a **2-way** IUCV SEND function is issued by CP, the virtual machine to which the accounting record is sent must issue an IUCV REPLY function. Response data may not be sent on an IUCV REPLY function. The reply buffer length field in the IUCV parameter list, IPBFLN2F, must contain zeros. To do this, set a register to zero and code ANSLEN=(register) on the IUCV REPLY function.

Establishing Communication

Before issuing the IUCV CONNECT function to the Account system service, a virtual machine must issue a DECLARE BUFFER request to IUCV to provide an external interrupt buffer. The virtual machine must be enabled for IUCV interrupts in Control Register 0, and the PSW must be set to enable external interrupts.

The connection with the Account system service is created by issuing the IUCV CONNECT function, specifying the user ID as *ACCOUNT. The use of the 2-way protocol for gathering accounting records from the CP Account system service is specified by the virtual machine in the IPUSER data area when it issues the IUCV CONNECT function to the Account system service. This area must contain X'02' at offset 8 if the application is written to issue the IUCV REPLY function after data is received from the Account system service. If the area does not contain a X'02', the default is 1-way communication. The CONNECT parameter list must also indicate that you do not want to receive messages with data in the parameter list. This is indicated by specifying or defaulting to the PRMDATA=NO option on the IUCV CONNECT function.

A virtual machine is not allowed to issue the IUCV SEND function to the Account system service (the path is quiesced by CP recording services). A virtual machine may only have one communication path to the Account system service. CP Recording system services will only send records with the PRTY=NO option.

When an application running in a virtual machine issues the CONNECT function to the accounting system service, the connection is either completed successfully (by ACCEPT) or rejected (by SEVER). If the connection is accepted, IUCV returns a path ID to the application that must be specified on all subsequent IUCV requests to the system service. Only one CONNECT can be issued by a virtual machine to the Account system service.

If the connection is severed, the Account system service places a 1-byte code at offset 9 of the IPUSER field of the IPARML to indicate why. A code of:

- X'04' indicates that the virtual machine already has a connection to the Account system service.
- X'08' indicates that the virtual machine made a protocol error on the CONNECT request. The PRMDATA=YES option was specified, but it should not have been.
- X'0C' indicates that the limit of 100 recording table entries has been reached and there is no room for another.

© Copyright IBM Corp. 1991, 2020

The data format of an accounting record is identical to the records recorded through use of the RETRIEVE ACCOUNT command. More than one user ID may be authorized to use this service.

Receiving Accounting Records

To obtain an accounting record, when the application is notified by an external interrupt that one is available, the IUCV RECEIVE function must be issued. The Account system service does not send another record until either a response (when the application indicates that data is to be sent to it using the 1-way protocol) or a REPLY (when an application indicates that data is sent to it using the 2-way protocol) is received by CP recording services to the previous record sent.

The Account system service maintains a threshold limit which indicates when to notify the system operator and the receiving virtual machine that uncollected records are accumulating in host storage. The default value is 20 for accounting records. This value may be changed using the CP RECORDING command. For more information on using CP commands, see *z/VM: CP Commands and Utilities Reference*.

To stop the receipt of records temporarily, you may issue the IUCV SEVER function. CP continues to queue records for your virtual machine until a CP RECORDING ACCOUNT OFF command is issued specifying your user ID. To resume receiving records, you may issue the IUCV CONNECT function specifying USERID=*ACCOUNT.

If the CP abends while a virtual machine is collecting accounting data, accounting records not received by the virtual machine are checkpointed and requeued to the virtual machine on a subsequent warm or force start of the CP. The virtual machine is also logged onto the system automatically by CP if it is identified on the SYSTEM_USERIDS statement in the system configuration file.

Disconnecting from the Accounting System Service

You can terminate collection of accounting records by issuing the IUCV SEVER function or the IUCV RETRIEVE BUFFER function for your accounting system service path. SEVER may be initiated by the system due to virtual machine reset or an IUCV RETRIEVE BUFFER request. CP continues to queue records for your virtual machine until a CP RECORDING ACCOUNT OFF command is issued specifying your user ID.

Accounting Record Formats

CP produces the following types of accounting records:

- Virtual machine user records (record type 1).
- Records for devices dedicated to a virtual machine user (record type 2).
- Records for temporary disk space dedicated to a virtual machine user (record type 3).
- Records that are written when CP detects that a user has entered enough LOGON, AUTOLOG, XAUTOLOG, or APPCVM CONNECT invocations with an invalid password to reach or exceed an installation-defined threshold value (record type 04).
- Records that are written when CP detects that a user has successfully entered a LINK command to a protected minidisk not owned by the user (record type 05).
- Records that are written when CP detects that a user has entered enough LINK commands with an invalid password to reach or exceed an installation-defined threshold value (record type 06).
- Records generated whenever a user logs off or disconnects from a device controlled by the VCNA or VSCS (record type 07).
- Records that are written when CP detects that a user has successfully entered a LINK command to a protected minidisk not owned by that user's virtual machine (record type 08). Record type 08 is also generated when the user logs off or disconnects or when a SHUTDOWN or FORCE command causes a

logged-on virtual machine to be forced off the system. Disconnected virtual machines do not have another 08 record generated for them if they are forced off.

- Records generated about ISFC (record type 09).
- Records logging changes to a user's privilege class (record type 0A)
- Records for virtual disk in storage space (record type B)
- Records for Network Data Transmissions (record type C)
- Records generated as a result of a user-initiated DIAGNOSE X'4C' instruction (record type C0)
- Records for CPU capability (record type D).

Note: Record types 04, 05, 06, and 08 are generated only when LOGON, AUTOLOG, XAUTOLOG, LINK, and CONNECT journaling is on.

These records are all 80-character card images.

The size of the fields in the accounting records restricts the data limit that each field can contain. To find the available field size, refer to each record type. Since the fields are limited in their capacity, it is recommended that you cut accounting records at least once a day.

Accounting Records for Virtual Machine Resource Usage (Record Type 1)

Type 1 accounting records are produced whenever a user logs off or the ACNT command is entered. These records can also be produced when a DEFINE CPU command issued from the command line specifies a change in CPU type. If a DEFINE CPU command is issued through a COMMAND directory statement, no accounting records are produced. The state of CPU affinity during the accounting interval will determine primary and secondary CPU usage. When there is usage on both primary and secondary types, two type 1 accounting records are produced. In this case, only the first record contains data in columns 41 through 64 and 73 through 76. Columns 1 through 28 and 79 and 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable). All reserved columns contain EBCDIC blanks. If the records are produced as a result of the DEFINE CPU command, the records contain usage information for only the CPU being redefined.

Column
Contents

1–8
User ID

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Number of seconds connected to CP

33–36
Milliseconds of processor time used, including time for supervisor functions (see Note “3” on page 702)

37–40
Milliseconds of virtual CPU time used (see Note “3” on page 702)

41–44
Number of page reads

45–48
Number of page writes

49–52
Number of requested virtual I/O starts for non-spooled I/O

53–56
Number of virtual punch cards sent to a virtual punch

57–60
Number of virtual print lines sent to a virtual printer (this includes one line for each carriage control command)

61–64
Number of virtual punch cards received from a virtual reader

65
Virtual CPU type code (see Note “2” on page 702)

66
Real CPU type code (see Note “2” on page 702)

67
Number of threads on the Real CPU on which the virtual CPU was dispatched (see Note “4” on page 702)

68–72
Reserved

73–76
Number of completed virtual I/O starts for non-spooled I/O (except DIAGNOSE X'58' and DIAGNOSE X'98')

77–78
CPU address (for the SYSTEM VMDBK, this is the real processor address)

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (1)

Notes:

1. User virtual machine time may be recorded in more than one record entry. For example:
   - Processor time is accumulated under connect time (bytes 29 - 32) and processor time (bytes 33 - 40)
2. Virtual and Real CPU Type Codes:
   - X'00' – general purpose Central Processor (CP)
   - X'02' – IBM zEnterprise® Application Assist Processor (zAAP)
   - X'03' – IBM Integrated Facility for Linux (IFL)
   - X'04' - Internal Coupling Facility (ICF)
   - X'05' – IBM z Integrated Information Processor (zIIP)
3. If simultaneous multithreading is not installed on the hardware or not enabled on the system, these fields contain the raw time reported by the hardware CPU timer (and identical to the MT-1 equivalent time). If multithreading is enabled, these time fields contain the MT-1 equivalent time and a type F accounting record is generated. For the raw time and prorated core time values, see “Accounting Records for Virtual Machine Resource Usage 2 (Record Type F)” on page 717.
4. This is the number of threads on the Real CPU on which the virtual CPU was dispatched for the time accrued in this accounting record, or an EBCDIC blank (equivalent to binary integer 64). The number of threads is between 1 and 32. When the number is 1, the CPU time used on that CPU will be identical regardless of whether it is reported as raw time or MT-1 equivalent time. This allows the type of CPU time reported in this record to apply to all CPUs on the system. If multithreading is not supported on the hardware or is not installed, the value will be 1. If this column contains an EBCDEC blank (equivalent to binary integer 64), the Real CPU is running single threaded and the value should be treated as 1.

Accounting Records for Dedicated Devices (Record Type 2)

A type 2 accounting record is produced whenever a virtual machine user releases a previously dedicated device. Columns 1 through 28 and 79 and 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable). For translation codes in columns 33 through 36, see Device Class and Type Codes in *z/VM: CP Planning and Administration*. See the documentation for each specific device for more complete information on model numbers.

Column
Contents

1–8
User ID

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Number of seconds since the virtual disk was created or the number of seconds since the last accounting record was cut for this virtual disk

33
Device class

34
Device type

35
Device model (if any)

36
Device features (if any)

37–78
Reserved

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (2)

Accounting Records for Temporary Disk Space (Record Type 3)

A type 3 accounting record is produced whenever a virtual machine user releases temporary disk space. Columns 1 through 28 and 79 and 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable). See Device Class and Type Codes in *z/VM: CP Planning and Administration* for information about translation codes for columns 33 through 36. See the documentation for each device for more information on model numbers.

Column
Contents

1–8
User ID

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Number of seconds since the TDISK was created or the number of seconds since the last accounting record was cut for this TDISK

33
Device class

34
Device type

35
Device model (if any)

36
Device features (if any)

37–38
CKD and ECKD Only Number of cylinders of temporary disk space used (only present for CKD or ECKD architected device types)

39–40
CKD and ECKD Only Reserved

37–40
FBA Only Number of FBA blocks used (only present for FBA architected device types)

41–44
Number of 4-KB pages used (present for all device types)

45–78
Reserved

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (3)

Accounting Records for Journaling (Record Types 04, 05, 06, 08, and 0I)

When LOGON, AUTOLOG, XAUTOLOG, LINK, or APPCVM CONNECT journaling is on, CP may create type 04, type 05, type 06, type 08, and type 0I accounting records.

A type 04 accounting record is written whenever CP detects that a user has issued enough LOGON, AUTOLOG, XAUTOLOG, or APPCVM CONNECT invocations with an invalid password to reach or exceed an installation-defined threshold value. This record has the following format:

Column
Contents

1–8
User ID specified on the command

9–16
Reserved for IBM use

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Terminal address (see Note “1” on page 707)

33–40
Invalid password (see Note “2” on page 707)

41–48
User ID that entered AUTOLOG, XAUTOLOG, APPCVM CONNECT, or the BYUSER ID that entered LOGON (see Note “4” on page 707).

49–51
Reserved for IBM use

52–53
Current invalid password count in hexadecimal

54–55
Accounting record limit in hexadecimal

56
Blank

57–62
Reserved

63–70
Network qualifier for SNA terminal. Host virtual machine name for TCP/IP terminal.

71–78
LUNAME for SNA terminal. IP address for TCP/IP terminal (see Note "5" on page 707).

79–80
Accounting card identification code (04)

A type 05 accounting record is produced whenever CP detects that a user has successfully entered a LINK command to a minidisk protected by a password and not owned by that user. This record is always produced when an external security manager authorizes the link. This record has the following format:

Column
Contents

1–8
User ID that entered the command

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Terminal address (see Note "1" on page 707)

33–40
Reserved for IBM use

41–48
User ID of the user that owns the minidisk

49–52
The minidisk address for which the LINK command was entered

53
Type of minidisk linked (see Note "3" on page 707)

54–55
Reserved for IBM use

56–57
Blank

58–62
Reserved

63–70
Network qualifier for SNA terminal. Host virtual machine name for TCP/IP terminal.

71–78
LUNAME for SNA terminal. IP address for TCP/IP terminal (see Note "5" on page 707).

79–80
Accounting card identification code (05)

A type 06 accounting record is produced whenever CP detects that a user has entered enough LINK commands with an invalid password to reach or exceed an installation-defined threshold value. This record has the following format:

Column
Contents

1–8
User ID that entered the command

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Terminal address (see Note "1" on page 707)

33–40
Invalid password (see Note "2" on page 707)

41–48
User ID of the user that owns the minidisk

49–51
Reserved for IBM use

52–53
Invalid password count in hexadecimal

54–55
Invalid password limit in hexadecimal

56
Blank

57–60
Minidisk address for which the LINK command was entered

61–62
Reserved

63–70
Network qualifier for SNA terminal. Host virtual machine name for TCP/IP terminal.

71–78
LUNAME for SNA terminal. IP address for TCP/IP terminal (see Note "5" on page 707).

79–80
Accounting card identification code (06)

A type 08 record is generated when a user logs off or disconnects or when a SHUTDOWN or FORCE command causes that logged-on user to be forced off the system. Disconnected users do not have another 08 record generated for them if they are forced off. This record has the following format:

Column
Contents

1–8
User ID

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–48
Reserved

49–56
LUNAME for SNA terminal. IP address for TCP/IP terminal (see Note "5" on page 707).

57–64
Network qualifier for SNA terminal. Host virtual machine name for TCP/IP terminal.

65–72
Terminal identification (logical device number, real device number, LUNAME for SNA terminal or NONE)

73–78
Reserved

79–80
Accounting record identification code (08)

Notes:

1. For the terminal address, columns 29 through 32 may contain one of the following:
   - NONE—if no terminal is found
   - SNA—if terminal is SNA (LUNAME is in columns 71–78)
   - A real or logical device number in the form L*nnn*, where *nnn* is the logical device number, for all other cases.
2. For the invalid password, columns 33 through 40 may contain one of the following:
   - Incorrect password
   - TOO LONG—if entered password is more than 8 characters.
3. For the type of minidisk linked, column 53 may contain one of the following:
   - X'00'–if the link is to a user's minidisk
   - X'10'–if the link is to a full-pack overlay minidisk.
4. A by-user is a user who logs on to a virtual machine using the BY operand of the LOGON command. The by-user's own password is used for LOGON authorization checking for the virtual machine, so the invalid password attempts are counted against the by-user ID, not the user ID of the virtual machine.
5. IP addresses are normally written in dotted-decimal format (for example, 9.130.58.78). In journal records, each segment of the IP address is converted to a two-digit hexadecimal value. For example, 9 is converted to 09, and 130 is converted to 82. The result is an eight-byte string of four two-digit hexadecimal numbers in character form. So, 9.130.58.78 becomes the character string 09823A4E.

A type 0I record is generated when the user is logged on through a logical device with an associated IPv6 address. This record has the following format:

Column
Contents

1–8
VM user ID

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–36
Host virtual machine name

37–68
IPv6 address (see note "1" on page 707)

69–78
Reserved

79–80
Accounting record identification code (0I)

Note:

1. IPv6 addresses are normally written in hexadecimal text representation.

   Example: `0123:4567:89AB:CDEF:FEDC:BA98:7654:3210`

In type 0I records, each segment of the IP address is converted from binary to character form. The result is a 32-byte string of 16 two-digit hexadecimal numbers in character form. So, the address shown above becomes 0123456789ABCDEFFEDCBA9876543210.

Accounting Records for SNA/CCS (Record Type 07)

A type 07 accounting record is produced whenever a user logs off or disconnects from a device controlled by VCNA or VSCS. The record indicates the user's share of the VCNA/VSCS resource used. Columns 1 through 16 and 79 and 80 of this record contain character data. See the *VCNA Installation and Terminal Use Guide* for details of VM/VCNA accounting records and the *ACF/VTAM Planning and Installation* book for details of VSCS accounting records.

Column
Contents

1–8
User ID

9–16
Account number

17–78
VSCS/VCNA accounting data

79–80
Accounting record identification code (07)

Accounting Records for Inter-System Facility for Communications (Record Type 09)

A type 09 accounting record is produced when the ALL option of the ACNT command is entered. Accounting records are generated for all active conversations and all active links.

There are four different categories of ISFC accounting records:

- Initialization accounting records
- Conversation accounting records
- Link statistics accounting records
- Termination accounting records.

ISFC produces the initialization accounting record during its initialization on the z/VM system. This record, along with the termination accounting record indicates the time frame in which ISFC was active on the z/VM system.

The format of the initialization accounting record is:

Column
Contents

1–8
SYSISFC, indicating that this record was created by the z/VM domain controller

9–12
Initialization record identifier, ISFI

13–16
Reserved for IBM use

17–28
Date and time the accounting record is generated

29–78
Reserved for IBM use

79–80
ISFC accounting record identifier

The format of the conversation start accounting record is:

Column
Contents

1–8
SYSISFC, indicating that this record was created by the z/VM domain controller

9–12
Conversation start accounting record identifier, ISFS

13–16
Conversation ID

17–28
Date and time the accounting record is generated

29–36
User ID of the user that initiated the conversation

37–59
Reserved for IBM use

60
Type of name in bytes 61–68. R indicates a global resource. G indicates a gateway name. U indicates a private resource server virtual machine or workstation user ID. I indicates IUCV.

61–68
Resource name, a gateway name, the user ID of the private resource server virtual machine or workstation, or target userid for an IUCV CONNECT.

69–78
Reserved for IBM use

79–80
ISFC accounting record identifier

The format of the conversation active and conversation end accounting records is:

Column
Contents

1–8
SYSISFC, indicating that this record was created by the z/VM domain controller

9–12
Conversation record identifier. ISFA indicates a conversation active accounting record; ISFE, a conversation end accounting record.

13–16
Conversation ID

17–28
Date and time the accounting record is generated

29–32
Number of bytes received from the remote domain controller since the conversation started or since the last conversation active accounting record was issued.

33–36
Number of bytes sent to the remote domain controller since the conversation started or since the last conversation active accounting record was issued.

37–78
Reserved for IBM use.

79–80
ISFC accounting record identifier

The format of the link statistics accounting records is:

Column
Contents

1–8
SYSISFC, indicating that this record was created by the z/VM domain controller

9–12
Link statistics record identifier, ISFL

13–16
Reserved for IBM use

17–28
Date and time the accounting record is generated

29–32
Number of bytes of data received (unsigned binary fullword)

33–36
Number of bytes of data sent since the link came up or since the last accounting record was generated for this link (unsigned binary fullword)

37–40
Unit address of the link

41–44
Number of attention interrupts

45–48
Number of write operations that result in collisions

49–52
Number of successful write operations

53–56
Number of successful read operations

57–78
Reserved for IBM use.

79–80
ISFC accounting record identifier

The format of the termination accounting record is:

Column
Contents

1–8
SYSISFC, indicating that this record was created by the z/VM domain controller

9–12
Termination record identifier, ISFT.

13–16
Reserved for IBM use

17–28
Date and time the accounting record was generated

29–78
Reserved for IBM use

79–80
ISFC accounting record identifier

Accounting Records for logging changes to a user's privilege (Record Type 0A)

A type 0A accounting record is produced whenever a SET PRIVCLASS command is successfully issued. The record tracks changes to a user's privilege class and their ability to change their privilege class settings.

There are 4 subtypes of this record.

Subtype
Description

L
SET PRIVCLASS LOCK has been issued

U
SET PRIVCLASS UNLOCK has been issued

C
The user's privilege class(es) have been changed via the SET PRIVCLASS command.

R
SET PRIVCLASS RESET has been issued

The format for subtypes 'L' (SET PRIVCLASS LOCK) and 'U' (SET PRIVCLASS UNLOCK) appears below. Columns 1 through 36 and 78 through 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable).

Column
Contents

1–8
User ID of the issuer of the SET PRIVCLASS command.

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–36
Target user ID. User ID that is the target of the SET PRIVCLASS lock or unlock

37–77
Reserved

78
Subtype

Character
Subtype Meaning

L
SET PRIVCLASS LOCK has been issued

U
SET PRIVCLASS UNLOCK has been issued

79–80
Accounting record identification code (0A)

The format for subtypes 'C' (SET PRIVCLASS change) and 'R' (SET PRIVCLASS RESET) appears below. Columns 1 through 36 and 78 through 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable). The privilege class information consists of bits representing privilege classes A-Z, and 1-6 respectively. The bit definitions are defined in HCPCLASS COPY.

Column
Contents

1–8
User ID of the issuer of the SET PRIVCLASS command

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–36
Target user ID. User ID whose settings are being changed

37–40
User privilege classes before the change or reset

41–44
User privilege classes after the change or reset

45–48
User privilege classes as indicated in the directory

49–77
Reserved

78
Subtype

Character
Subtype Meaning

C
The User's privilege class has been changed

R
The User's privilege class has been reset

79–80
Accounting record identification code (0A)

Accounting Records for virtual disk in storage space (Record Type B)

A type B accounting record is produced for a virtual disk in storage. Columns 1 through 28 and 79 and 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable). For translation codes in columns 33 through 36, see Device Class and Type Codes in *z/VM: CP Planning and Administration*. See the documentation for each specific device for more complete information on model numbers.

Column
Contents

1–8
User ID. This user ID is defined as follows:

- If the virtual disk in storage is defined in the directory, this is the user ID which contains the MDISK definition for this virtual disk in storage.
- If the virtual disk in storage was defined using the CP DEFINE command, this is the user ID that issued the DEFINE command.

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Number of seconds connected to CP

33
Device class

34
Device type

35
Device model (if any)

36
Device features (if any)

37–40
Number of FBA blocks used

41–44
Number of 4 KB pages used (present for all device types)

45
Sub-type of virtual disk accounting record. The only volume defined is zero. If non-zero, the contents of bytes 1-44 and 46-78 are undefined.

46–78
Reserved

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (B)

Accounting Records Network Data Transmissions (Record Type C)

Type C accounting records may be produced for any virtual machine user with NETAccounting or NETRouter specified as an option in its user directory. The account records are produced when accounting is performed (CP ACNT command), or when a field in the accounting record is about to overflow, and the user has sent or received data from a Network device. These records are only produced when fields containing byte counters for a device are non-zero.

Network devices to be included in these counts are:

- Virtual network interface cards (NIC)
- Virtual Channel to Channel Adapters
- IUCV and APPC/VM connection paths

The contents of Network Data Transmission accounting records depends on the type of Network device being used. There are 3 different formats, designated by a subtype in byte 78. For subtype 00 (Virtual NIC records), an additional 4 subformats, designated in byte 77, indicate what type of network data is counted in the record.

Type C account record describing data transfer involving routers through a Virtual NIC (Network device type 00). Columns 1 through 28, 35 through 50, and 79 through 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable.)

Column
Contents

1-8
VM Userid (Owner of the Virtual NIC)

9-16
Account number

17-28
Date and time of accounting (mmddyyhhmmss)

29-30
Base VDEV address of the Virtual NIC

31-34
Default IP Address for this network adapter

35-42
LAN or virtual switch owner

43-50
LAN name or virtual switch name

51-58
Bytes Sent (See Data Descriptor)

59-66
Bytes Received (See Data Descriptor)

67-76
Reserved

77
Data Descriptor:

00 - Bytes sent to or received from Routers
01 - Bytes sent to or received from non-Routers
02 - Bytes sent to or received via Broadcast
03 - Bytes sent to or received via Multicast

78
Network device type (00) Virtual NIC

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (C)

Notes:

1. Hardware headers which appear before the IP Header are not included in the byte counts reported for simulated network adapters.
2. IP addresses are normally written in dotted-decimal format (for example, 9.130.58.78). In journal records, each segment of the IP address is converted to a two-digit hexadecimal value. For example, 9 is converted to 09, and 130 is converted to 82. The result is an 8-byte string of four 2-digit hexadecimal numbers in character form. So, 9.130.58.78 becomes the character string 09823A4E.

TYPE C accounting record for data transfer with a Virtual Channel to Channel Adapter. Columns 1 through 28, 35 through 42, and 79 through 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable.)

Column
Contents

1-8
VM Userid (owner of the local CTCA)

9-16
Account number

17-28
Date and time of accounting (mmddyyhhmmss)

29-30
VDEV address of the local CTCA

31-34
Reserved

35-42
Userid of the owner of the remote CTCA

43-44
VDEV address of the remote CTCA

45-50
Reserved

51-58
Number of bytes sent to the remote CTCA.

59-66
Number of bytes received from the remote CTCA.

67-77
Reserved

78
Network device type (01) virtual CTCA

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (C)

TYPE C account record for data transfer with an IUCV or APPC/VM connection. Columns 1 through 28, 35 through 42, and 79 through 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable.)

Column
Contents

1-8
VM Userid (Owner of the local connection.)

9-16
Account number

17-28
Date and time of accounting (mmddyyhhmmss)

29-34
Reserved

35-42
Connected VM Userid (remote)

43-50
Reserved

51-58
Number of bytes sent to the remote user.

59-66
Number of bytes received from the remote user.

67-77
Reserved

78
Network device type (02) IUCV or APPC path

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (C)

Notes:

1. Byte count values are reset to zero after an accounting record is created.
2. Byte count values represent data read or written to the device. They will not typically contain bytes transferred as a result of initialization, termination, or error recovery.
3. Accounting for a particular LAN may be controlled by the DEFINE LAN, and SET LAN commands, and default settings may be set by the VMLAN command or VMLAN statement in the system config file.

Accounting Records for CPU Capability (Record Types D and E)

A type D accounting record is produced to record the CPU capability of the processors in the system. A record is generated during z/VM system initialization and whenever the CPU capability changes. All fields of the record contain character data.

Column
Contents

1-16
Reserved

17-28
Date and time of accounting (mmddyyhhmmss)

29-36
Primary CPU capability

37-44
Secondary CPU capability

45-52
Nominal CPU capability

53–55
Capacity-Change Reason

56–58
Capacity-Adjustment Indication

59–78
Reserved

79
Card generator field (0)

80
Accounting record identification code (D)

Notes:

1. When the secondary CPU capability value is zero, it means all CPUs of any CPU type in the configuration have the same capability, as specified by the primary CPU capability.
2. When the nominal CPU capability value is zero, it means the system is not operating at reduced speed, so the primary CPU capability value applies.

A type E (CPU Capability continuation data) accounting record is produced at the same time as the CPU Capability (type D) accounting record. All fields of the record contain character data. The capability fields contain 8-digit approximations of the binary-floating point values and the reserved fields contain blanks.

Column
Contents

1-8
User ID

9-16
Account number

17-28
Date and time of accounting (mmddyyhhmmss)

29-36
Primary CPU capability decimal value (see note 1)

37-44
Secondary CPU capability decimal value (see note 1)

45-52
Nominal CPU capability decimal value (see note 1)

53–78
Reserved

79
Card generator field (0)

80
Accounting record identification code (E)

Notes:

1. These fields will be reported as "OVERFLOW" if the capability value is greater than or equal to 100000000 (decimal). These fields will be reported as "UNDERFLO" if the capability value is less than or equal to 10^{-7}.
2. When the secondary CPU capability value is zero, it means all CPUs of any CPU type in the configuration have the same capability, as specified by the primary CPU capability.
3. When the nominal CPU capability value is zero, it means the system is not operating at reduced speed, so the primary CPU capability value applies.

Accounting Records for Virtual Machine Resource Usage 2 (Record Type F)

Type F accounting records are a continuation of the type 1 record. When multithreading is enabled on the system, a single type F record is produced for each virtual CPU for which one or two type 1 accounting records (for primary and secondary CPU types) are produced. Columns 1 through 28, 61 through 68, and 79 and 80 of this record contain character data; all other data is in hexadecimal form (the hexadecimal data is unprintable). All reserved columns contain binary zeros.

Column
Contents

1–8
User ID

9–16
Account number

17–28
Date and time of accounting (*mmddyyhhmmss*)

29–32
Milliseconds of processor time used on the primary CPU type, including time for supervisor functions, in raw time.

33–36
Milliseconds of virtual CPU time used on the primary CPU type, in raw time.

37–40
Milliseconds of processor time used on the secondary CPU type, including time for supervisor functions, in raw time.

41–44
Milliseconds of virtual CPU time used on the secondary CPU type, in raw time.

45–48
Milliseconds of processor time used on the primary CPU type, including time for supervisor functions, in prorated core time (see Note “1” on page 718)

49–52
Milliseconds of virtual CPU time used on the primary CPU type, in prorated core time (see Note “1” on page 718)

53–56
Milliseconds of processor time used on the secondary CPU type, including time for supervisor functions, in prorated core time (see Note “1” on page 718)

57–60
Milliseconds of virtual CPU time used on the secondary CPU type, in prorated core time (see Note “1” on page 718)

61–68
Name of the CPU pool (see Note “2” on page 718)

69–76
Reserved

77
Codes regarding this accounting record:

X'80'
The prorated core time fields in this record are valid.

78
Reserved

79
Card generator field (C if Diag 4C has been issued, A if adjunct configuration record, or 0 otherwise)

80
Accounting record identification code (F)

Notes:

1. When APAR VM65680 is applied and multithreading is enabled, prorated core times are calculated and reported for every virtual machine. The prorated core code provided in column 77 of this record indicates whether these values are included in the accounting record.
2. When the virtual machine is not assigned to a CPU pool, this field contains EBCDIC blanks.

Adding Your Own Accounting Records and Source Code

CP allows you to customize the way it collects accounting records in two ways:

- A virtual machine can use the DIAGNOSE instruction to initiate the generation of a virtual machine accounting record. For more information, see "User-Initiated Accounting Records (Record Type C0)" on page 718
- You can add your own source code to the CP accounting exit module. For more information, see CP Accounting Exit in *z/VM: CP Exit Customization*.

User-Initiated Accounting Records (Record Type C0)

A virtual machine user can initiate the creation of an accounting record that contains up to 70 bytes of information of the user's choosing. To do this, the user enters a DIAGNOSE code X'4C' instruction with the following operands (the *z/VM: CP Programming Services* describes how to enter a DIAGNOSE code):

- The address of a data area in virtual storage that contains the information that the user wants in columns 9 through 78 of the card image record. (This information is placed in the accounting record exactly as it appears in the data area. If more than 70 bytes of data are included, only the first 70 bytes appear in the accounting record.)
- A function code of X'10'.
- The length of the data area in bytes.

The information on this type of accounting record is as follows:

Column
Contents

1–8
User ID

9–78
User-formatted data

79–80
Accounting record identification code (C0)

File pool service virtual machines can generate accounting records in this category. For details, see *z/VM: CMS File Pool Planning, Administration, and Operation*.

Chapter 13. Asynchronous CP Command Response System Service (*ASYNCMD)

The Asynchronous CP Command Response (*ASYNCMD) system service lets a virtual machine receive CP command responses that come from CP as result of issuing the CP FOR command, rather than displaying them on the terminal. (The texts of the CP command responses and the messages themselves is not part of the programming interface.)

*ASYNCMD is the assigned Asynchronous CP Command Response system service user ID. You establish communication with this user ID by specifying USERID=*ASYNCMD when issuing the IUCV CONNECT function.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317.

Establishing Communication

Your virtual machine does not need any special authorization to use the Asynchronous CP Command Response system service. To establish IUCV communication, issue IUCV DECLARE BUFFER followed by IUCV CONNECT with USERID=*ASYNCMD.

After a successful IUCV CONNECT to *ASYNCMD, the IUCV path ID returned by the connect may be used on FOR commands to receive command responses, on that path, from the Asynchronous CP Command Response system service. For more information, see the “DECLARE BUFFER Function” on page 331 and the “CONNECT Function” on page 324.

Your virtual machine may have up to 8 communication paths with the Asynchronous CP Command Response system service. If your virtual machine attempts to establish a new IUCV connection while there are 8 communication paths already in use, the attempt will be terminated by the system service with the IUCV SEVER function. You may terminate your connection by using the IUCV SEVER function. For more information, see the “SEVER Function” on page 376.

Message Limits

The Asynchronous CP Command Response system service uses a default value of 16000 for the number of outstanding messages allowed on the path unless the user specifies a lower value for the MSGLIM parameter. If a lower value is specified, that value is used for the number of messages allowed on the path.

If the message limit is exceeded, any additional incoming messages are discarded and the "end of command record" shows the number of records that have been discarded. This situation is most likely to occur when there is a high volume of incoming message and the virtual machine is running with external interrupts disabled.

Sending and Receiving Data

After your virtual machine issues IUCV CONNECT to the Asynchronous CP Command Response system service, FOR commands can be issued specifying the path ID returned from IUCV CONNECT along with a *token* value. The Asynchronous CP Command Response system service will issue the IUCV SEND function to send the data to your virtual machine on your specified path where you can issue IUCV RECEIVE to receive the data. For more information, see the “SEND Function” on page 366 and the “RECEIVE Function” on page 348.

© Copyright IBM Corp. 1991, 2020

Note: Your virtual machine is not allowed to issue the IUCV SEND function to the Asynchronous CP Command Response system service. Do not quiesce a path to the Asynchronous CP Command Response system service or you could lose messages. The Asynchronous CP Command Response system service will only send records with the PRTY=NO option.

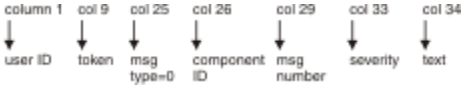
Record Types

There are two types of records which can be received over an IUCV connection to the Asynchronous CP Command Response system service:

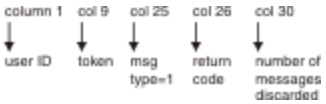
1. Command response records.
2. End of command record.

The command response records are the output records generated by the CP command issued for the target virtual machine. The end of command record is the last record associated with a CP command which includes the return code and count of discarded messages.

Command response record format



End of command record format



userid
is the user ID specified on the FOR command which is the target of the CP command.

token
is the user-specified correlation token specified on the FOR command, or blanks if nothing was specified.

msg type
is a 1-byte EBCDIC value with the following values defined:

X'F0'
is a command response record.

X'F1'
is an end of command record.

component ID
is a 3-character prefix indicating which component generated the message. For example, HCP.

msg number
is the CP message number displayed in character format. If no message number was received, the field is blank.

severity
is a 1-character message severity associated with the CP message. For more information on severities, see *z/VM: CP Messages and Codes*.

text
is the actual response that would have displayed on the target user's display terminal using the EMSG ON setting. This field does not contain the message number or severity.

return code
is a 4-character return code from the CP command that was executed.

number of messages discarded
is a 4-character value indicating how many messages were discarded. The value is zero if there were no messages discarded and padded to 4 characters.

Chapter 14. DASD Block I/O System Service (*BLOCKIO)

The DASD Block I/O system service (*BLOCKIO) provides a virtual machine with device-independent, asynchronous access to its CMS-formatted virtual disk devices.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317. To see a sample program, see Appendix B, “Sample Programs Using DASD Block I/O System Service,” on page 981.

Establishing Communication with the DASD Block I/O System Service

The DASD Block I/O system service uses IUCV to set up communication between itself and a virtual machine. The virtual machine issues the IUCV DECLARE BUFFER function to initialize for IUCV communication; the IUCV processor initializes the DASD Block I/O system service for IUCV communication by issuing the DECLARE BUFFER function for the system service.

After communication is established with IUCV, the virtual machine must issue a CONNECT command to establish a path between itself and the DASD block I/O system service. Only one CONNECT may be issued to the DASD Block I/O system service for each virtual device that is intended to receive I/O requests.

No special authorization is required for a virtual machine to use the DASD block I/O system service. The maximum connection limit (MAXCONN) in the directory can be enlarged to satisfy the user's requirements. The DASD block I/O system service allows connection from any user.

The IUCV macro checks the validity of all the IUCV parameters. Any IUCV errors are handled according to IUCV specifications. The DASD Block I/O system service checks the validity of all the parameters it requires. Any errors resulting from this check are handled as described in the following sections.

IUCV CONNECT to the DASD Block I/O System Service

An IUCV CONNECT is issued by the virtual machine with USERID=*BLOCKIO, and PRMDATA=YES specified in the IUCV CONNECT parameter list. The IPARML DSECT and IPARML COPY files are located in the HCPGPI macro library. The user data field (IPUSER) in the IUCV parameter list must have the following format:

Block size
contains the block size of the specified disk; the block size can be 512, 1K, 2K, or 4K bytes.

Offset
contains the number of sequential blocks used at the beginning of the disk by the CMS file system to implement its structure. *BLOCKIO uses the offset so that the first block after these used blocks is addressed as block 1. *BLOCKIO does not check the validity of the offset; therefore the application can change the number if desired.

Vdevaddr
contains the virtual device address of the disk on which the block I/O is to be performed.

Usage Notes

1. All reserved fields must be set to zero.
2. The disk must be in CMS format. Although not required by *BLOCKIO, the CMS RESERVE command is normally then used to allocate all available blocks of this formatted minidisk to a unique CMS file.

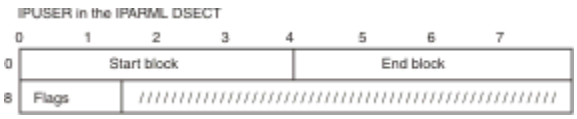
© Copyright IBM Corp. 1991, 2020

3. If the minidisk has been reserved, an application can use the DISKID CMS function to obtain the block size, offset, and virtual device address information. DISKID is described in *z/VM: CMS Macros and Functions Reference*.
4. The *BLOCKIO system service does not support HyperPAV alias devices.
5. The *BLOCKIO system service is limited to a minidisk size of 65520 cylinders, regardless of formatted block size. If the *BLOCKIO request was issued for a minidisk larger than 65520 cylinders, error code X'02' will be returned in the first byte of the IPUSER field.

IUCV ACCEPT

If all parameters required by *BLOCKIO are valid, *BLOCKIO issues an IUCV ACCEPT on the path specifying PRMDATA=YES.

The following information is returned in the IPUSER field of the IUCV connection-complete external interrupt buffer:



Start block
contains 1 minus the offset specified in the IUCV CONNECT. Start block and end block specify the range of block numbers allowable on the *BLOCKIO request.

End block
contains the number of blocks on the specified device minus the offset specified on the IUCV CONNECT. End block and start block specify the range of block numbers allowable on the *BLOCKIO request.

Flags
contains a set of bits defining the status of the virtual device. One bit is defined; the others are reserved.

RDONLY X'0001'
the virtual device is read-only

Note: All reserved fields must be set to zero.

IUCV SEVER

If any of the parameters passed to *BLOCKIO are invalid, *BLOCKIO issues an IUCV SEVER on the path and flags the error.

The first byte of the IPUSER field contains one of the following error codes:

Code
Meaning

X'01'
the virtual device is not defined.

X'02'
the virtual device is not supported.

X'03'
the block size is not supported.

X'04'
the IUCV path already exists for this device.

X'05'
the connection is not using PRMDATA=YES.

X'06'
the reserved field is not set to zero.

IUCV SEND to *BLOCKIO

When the connection is accepted by *BLOCKIO, you can start sending I/O requests to *BLOCKIO. You can specify the TRGCLS=, DATA=PRMMSG, and PRMMSG= options on the IUCV SEND, or you can move the necessary data into the IUCV parameter list yourself. The TRGCLS= option sets the type of I/O requested. The DATA=PRMMSG option sets a flag in IPRFLAGS1, and the PRMMSG option sets up IPRMMSG1 and IPRMMSG2 in the IUCV parameter list.

There are two different interfaces to *BLOCKIO. The single block interface is restricted to, and optimized for, single block transfer. The multiple block interface can be used to read/write from 1 to 256 blocks at a time.

Single Block I/O

The following list defines the input needed by *BLOCKIO for single block requests on an IUCV SEND:

IPRMMSG1
specifies the block number.

IPRMMSG2
specifies the guest absolute data buffer address.

IPTRGCLS
specifies the block I/O service requested.

Code
Meaning

F'01'
Write request, use the minidisk cache (if present).

F'02'
Read request, use the minidisk cache (if present).

F'81'
Write request, bypass the minidisk cache.

F'82'
Read request, bypass the minidisk cache.

Usage Notes

1. The SYNC=YES option on the IUCV SEND macro can be specified with the F'02' service request. If SYNC=YES is specified with any of the other three service requests, it will get a return code of F'06', indicating an invalid request.

   The synchronous processing option is intended for use when reading blocks. Applications that specify the SYNC=YES option have to be prepared for this request to complete synchronously or asynchronously. If the requested block is in the minidisk cache, the SEND request completes synchronously. If the requested block is not in the minidisk cache, the SEND request completes asynchronously.
2. Use the minidisk cache bypass option when reading or writing blocks that are not referenced frequently. This prevents infrequently-used data from filling the cache and flushing out frequently referenced data.
3. The *BLOCKIO IUCV system service and Diagnose X'250' do not honor DASD reserves managed by VM's virtual reserve/release function. Therefore, do not use these I/O interfaces if a minidisk is shared by multiple guests on the same VM image where another guest is expecting to use reserve/release I/O to serialize its data access. Also, *BLOCKIO and Diagnose X'250' do not use reserve/release. Therefore, do not use these interfaces for any DASD (CP-attached or full-pack minidisk) that is shared with other LPARs where another LPAR expects to use reserve/release to serialize its data.

Condition and Return Codes

The condition codes on the IUCV SEND instruction indicate how processing was completed:

Code
Meaning

CC=0
The request has been started. An IUCV Message Complete external interrupt is generated when the request completes.

CC=1
A nonzero value is stored in the IPRCODE field of the IPARML DSECT.

CC=2
The requested block was found in the minidisk cache and has been written into the guest buffer.

The IUCV protocol may have been correct from an IUCV perspective but does not meet the *BLOCKIO requirements. In that case, the DASD Block I/O system service issues an IUCV SEVER on the path and flags the error. The first byte of the IPUSER field contains one of the following error codes:

Code
Meaning

X'07'
IUCV communication was not sent using DATA=PRMMSG.

X'08'
No 1-way messages are allowed on the path.

If you coded the IUCV SEND correctly, *BLOCKIO tries to initiate the request. It uses an IUCV REPLY to return the results of the I/O request. The application's virtual machine is made aware of this response by an IUCV external interrupt. A return code is returned in the IPRMMSG1 field of the IUCV parameter list:

Code
Meaning

F'00'
I/O completed successfully

F'01'
Invalid block number

F'02'
Invalid data buffer address

F'03'
Write on read-only DASD

F'04'
Incorrect block size — format error

F'05'
Unrecoverable I/O error

F'06'
Invalid service requested

F'07'
Protection exception on virtual buffer

If the device is reset, the path is quiesced, and no more requests are allowed. When no I/O requests are outstanding, *BLOCKIO issues an IUCV SEVER on the path and flags the error. The first byte of the IPUSER field contains the following error code:

Code
Meaning

X'09'
Virtual device has been reset

Multiple Chained Block I/O

Using *BLOCKIO, you can read and write up to 256 contiguous or discontiguous CMS-formatted blocks with a single *BLOCKIO request. Individual blocks may be read and written within the same request. Multiple chained block I/O is invoked similarly to single block I/O requests, but the IUCV SEND parameter list is set up as follows:

IPRMMSG1
specifies the number of blocks to process (1 to 256).

IPRMMSG2
specifies the guest absolute address of the multiple block I/O parameter list. This list defines the blocks to be read and written and the buffers to use.

Also returned in the high-order halfword of IPRMMSG2 is the sum of Start Subchannel instructions issued and successful minidisk cache read requests needed to satisfy the multi-block request.

IPTRGCLS
specifies the block I/O service requested.

Code
Meaning

F'03'
Multiple request, use the minidisk cache (if present).

F'83'
Multiple request, bypass the minidisk cache.

Usage Notes

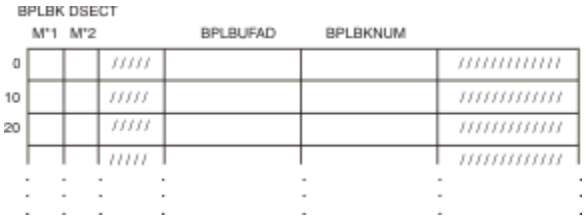
1. The SYNC=YES option on the IUCV SEND macro can be specified with the F'03' service request. If SYNC=YES is specified with the F'83' service request, it will get a return code of F'06', indicating an invalid request.

   The synchronous processing option is intended for use when reading blocks. Applications that specify the SYNC=YES option have to be prepared for this request to complete synchronously or asynchronously. If all the requested blocks are in the minidisk cache, the SEND request completes synchronously. If all or some of the requested blocks are not in the minidisk cache, the SEND request completes asynchronously.
2. Use the minidisk cache bypass option when reading or writing data that is not referenced frequently. This prevents infrequently-used data from filling the cache and flushing out frequently referenced data.
3. The multiple block I/O interface can be used to read or write a single block. However, the single block interface provides somewhat better performance.
4. The I/O for the DASD blocks specified in the block I/O entries of the BPLBK may not occur in the same order that they are listed. If the application requires that the DASD blocks or I/O data buffers be updated in a particular order, then that I/O request should be implemented with separate *BLOCKIO SEND requests.
5. The *BLOCKIO IUCV system service and Diagnose X'250' do not honor DASD reserves managed by VM's virtual reserve/release function. Therefore, do not use these I/O interfaces if a minidisk is shared by multiple guests on the same VM image where another guest is expecting to use reserve/release I/O to serialize its data access. Also, *BLOCKIO and Diagnose X'250' do not use reserve/release. Therefore, do not use these interfaces for any DASD (CP-attached or full-pack minidisk) that is shared with other LPARs where another LPAR expects to use reserve/release to serialize its data.

Multiple Block I/O Parameter List

The IPRMMSG2 field of the IUCV SEND parameter list points to a parameter list describing the blocks to be read and written and the buffers to use. Each 16-byte entry contains the information for one block to be read or written. For performance reasons, try not to let the parameter list cross a page boundary.

The contents of this multiple block I/O parameter list are as follows (up to 256 entries):



BPLQCOD (M*1)
is the 1-byte request code indicating a read or write. X'01' indicates Write, and X'02' indicates Read.

BPLSTAT (M*2)
is the 1-byte area in which a return code is placed, to indicate the status of the request.

BPLBUFAD
is a 4-byte field containing the guest absolute address of the buffer area into which this block should be read or written. Although there are no alignment requirements for the buffer itself, try not to let the buffer cross a page boundary.

BPLBKNUM
is a 4-byte field containing the number of the block to be read or written.

The BPLBK DSECT with the HCPBPLBK COPY files is located in the HCPGPI macro library.

Condition and Return Codes

The condition codes on the IUCV SEND instruction indicate how processing was completed:

Code
Meaning

CC=0
The request has been started. An IUCV Message Complete external interrupt is generated when the request completes.

CC=1
A nonzero value is stored in the IPRCODE field of the IPARML DSECT.

CC=2
All requested blocks were found in the minidisk cache and have been written into the guest buffers.

The IUCV protocol may have been correct from an IUCV perspective but does not meet the *BLOCKIO requirements. In that case, the *BLOCKIO system service issues an IUCV SEVER on the path and flags the error. The first byte of the IPUSER field contains one of the following error codes:

Code
Meaning

X'07'
IUCV communication was not sent using DATA=PRMMSG.

X'08'
No 1-way messages are allowed on the path.

The following conditions are detected by *BLOCKIO before any I/O is initiated. If any errors are detected, no I/O is performed. The return code is stored in the IPRMMSG1 field of the IUCV parameter list as follows:

Code
Description

F'00'
The multiple block parameter list is set up correctly.

Note: Check the return code for each entry in the multiple block I/O parameter list to verify that I/O has completed successfully for each block.

F'02'
The multiple block parameter list is outside the size of the virtual machine.

F'06'
An invalid service was requested.

F'07'
A storage protection error has occurred on the multiple block parameter list.

F'08'
The block count specified is not between 1 and 256, inclusive.

The following conditions are detected on a per-entry basis and do not prevent other blocks within the same request from being read or written. For those entries in error, no I/O is performed for that block (except where RC = X'05', unrecoverable I/O error). The return code is stored in the status field (BPLSTAT) of the individual entry of the multiple block I/O parameter list that contains the error.

Code
Description

X'00'
The block has been successfully processed.

X'01'
The block number specified is invalid.

X'02'
The buffer is outside the size of the virtual machine.

X'03'
A write has been requested to a read-only device.

X'04'
The block size specified is incorrect—format error.

X'05'
An unrecoverable I/O error has occurred.

X'06'
The request code is not X'01' for Write or X'02' for Read.

X'07'
A storage protection error has occurred on the buffer.

Ending Communication with the DASD Block I/O System Service

To end communication when all communication with *BLOCKIO is complete, issue either an IUCV SEVER or an IUCV RETRIEVE BUFFER.

Chapter 15. Error Logging System Service (*LOGREC)

An installation may write an application to run in a guest virtual machine which has been authorized to use the IUCV interface to receive LOGREC records from the z/VM control program supporting it. This IUCV authorization is defined using the IUCV directory control statement of the guest virtual machine. The IUCV control statement must name **LOGREC* as the CP system service to which a communication path is established. The user ID of the guest virtual machine may also be identified to the control program during system generation so that records can be accumulated for the virtual machine before it has connected to the system service.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

The Error Logging system service (*LOGREC) in CP supports both 1-way and 2-way IUCV protocols when sending records to authorized virtual machines. When a **1-way** IUCV SEND is issued by the CP, the virtual machine to which the LOGREC record is sent cannot issue an IUCV REPLY but must issue a RECEIVE. When a **2-way** IUCV SEND is issued by the CP, the virtual machine to which the LOGREC record is sent must issue an IUCV REPLY. Response data may not be sent on an IUCV REPLY. The reply buffer length field in the IUCV parameter list, IPBFLN2F, must contain zeros. This can be accomplished by setting a register to zero and coding ANSLEN=(*reg*) on the IUCV REPLY macro.

Establishing Communications with the Error Logging System Service

Prior to issuing the IUCV CONNECT to the *LOGREC system service, a virtual machine must issue a DECLARE BUFFER request to IUCV to provide an external interrupt buffer. The virtual machine must be enabled for IUCV interrupts in Control Register 0 and the PSW must be set to enable external interrupts.

The connection with the *LOGREC system service is created by issuing IUCV CONNECT, specifying the USERID as *LOGREC. The use of the 2-way protocol for gathering LOGREC records from the CP Error Logging system service is specified by the virtual machine in the IPUSER data area when it issues an IUCV CONNECT to the *LOGREC service. This area must contain a X'02' at offset 8 if the application is written to issue an IUCV REPLY after data is received from the CP *LOGREC system service. If the area does not contain a X'02', the default is 1-way communication. The CONNECT parameter list must also indicate that you do not want to receive messages with data in the parameter list. This is indicated by specifying or defaulting to the PRMDATA=NO option on the IUCV CONNECT.

When you issue CONNECT to the Error Logging system service, the connection is either completed successfully (by ACCEPT) or rejected (by SEVER). If the connection is accepted, IUCV returns a PATHID to you, which must be specified on all subsequent IUCV requests to the system service. Only one CONNECT can be issued by a virtual machine to the Error Logging system service.

If the connection is severed, the Error Logging system service places a 1-byte code at offset 9 of the IPUSER field of the IPARML to indicate why.

A code of:

- X'04' indicates that the virtual machine already has a connection to the Error Logging system service.
- X'08' indicates that the virtual machine made a protocol error on the CONNECT request. The PRMDATA=YES option was specified but it should not have been.
- X'0C' indicates that the limit of 100 recording table entries has been reached and there is no room for another.

A virtual machine is not allowed to issue an IUCV SEND to the *LOGREC service (the path is QUIESCED by CP Recording Services). A virtual machine may only have one communication path to the Error Logging system service. CP Recording services send only records with the PRTY=NO option.

© Copyright IBM Corp. 1991, 2020

The data format of a LOGREC record is identical to the records recorded through use of the CP RETRIEVE EREP command. More than one user ID may be authorized to use this service. For more information on how to use the CP commands, see *z/VM: CP Commands and Utilities Reference*.

Receiving LOGREC Records

To obtain a LOGREC record, when the application is notified by an external interrupt that one is available, issue an IUCV RECEIVE. The Error Logging system service does not send another record until either a response (when the application indicates that data is to be sent to it using the 1-way protocol) or a REPLY (when an application indicates that data is to be sent to it using the 2-way protocol) is received by CP Recording services to the previous record sent.

The Error Logging system service maintains a threshold limit which indicates when to notify the system operator and the receiving virtual machine that uncollected records are accumulating in host storage. The default value is 2 for LOGREC records. This value may be changed using the RECORDING command.

To stop receiving records temporarily, you may issue an IUCV SEVER. CP continues to queue records for your virtual machine until a CP RECORDING EREP OFF command is issued, specifying your user ID. To resume receiving records, you may issue an IUCV CONNECT specifying the user ID as *LOGREC.

If the CP abends while a virtual machine is collecting LOGREC data, LOGREC records not received by the virtual machine are checkpointed and requeued to the virtual machine on a subsequent warm or force-start of the control program. The virtual machine is also logged on the system automatically by CP if it is identified on the SYSTEM_USERIDS statement in the system configuration file. For more information on the SYSTEM_USERIDS statement, see *z/VM: CP Planning and Administration*.

Disconnecting from the Error Logging System Service

You can terminate collection of LOGREC records by issuing IUCV SEVER or IUCV RETRIEVE BUFFER for your *LOGREC system service path. A SEVER may be initiated by the system due to virtual machine reset or an IUCV RETRIEVE BUFFER request. CP continues to queue records for your virtual machine until a CP RECORDING EREP OFF command is issued specifying your user ID.

Chapter 16. Identify System Service (*IDENT)

The Identify system service (*IDENT) is a CP system service that lets authorized virtual machines connect to it and:

- Identify themselves as resource or gateway managers
- Revoke ownership of a resource or gateway
- Communicate with CP to request that a user's effective and saved-set POSIX user ID (UID) and group ID (GID) be set.

The Identify system service maintains a local system resource/gateway table of all resources and gateways managed on that system. A virtual machine manages a resource or gateway only while it is connected to *IDENT.

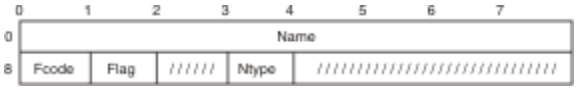
For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

Establishing Communication with the Identify System Service

For a virtual machine to manage or revoke a local, global, or system resource or gateway, it must first be authorized to connect to the identify system service, *IDENT. Your system administrator is the person who can authorize your virtual machine to manage or revoke a particular resource or gateway. To do this, the administrator must specify a special IUCV *IDENT statement in your virtual machine's directory entry. Your system administrator also can authorize the AVS virtual machine to revoke a particular gateway by specifying an IUCV *IDENT statement in the AVS virtual machine's directory entry.

For more information on how a system administrator authorizes virtual machines to manage or revoke resources and gateways, see *z/VM: CP Planning and Administration*.

Once authorized, your virtual machine must issue an IUCV CONNECT to use *IDENT. You must specify the user ID as *IDENT, and the user data field must have the following format:



Name
contains the name of the resource or gateway that you are managing or revoking. The first byte of this name must be alphanumeric; IBM reserves names beginning with characters for its own use. This resource/gateway name cannot be blanks (X'40's), nulls (X'00's), ANY, ALLOW, or SYSTEM.

Fcode
is the function code. FCODE=1 indicates a request to identify or manage a resource or gateway. FCODE=2 indicates a request to revoke a resource or gateway.

Flag
is a flag byte.

For **manage** requests (FCODE=1):

Bit 0 on
Defines the resource as being accessible from outside the local system.

- If bit 0 is on and bit 2 is off, this is a global resource, unique in the TSAF or CS collection and accessible throughout the TSAF or CS collection and by AVS.
- If bit 0 is on and bit 2 is on, this is a system resource, accessible throughout the TSAF or CS collection through the system gateway and by AVS.

© Copyright IBM Corp. 1991, 2020

This bit must be on for a gateway.

Bit 0 off
Defines the resource as being accessible only from the local system (a local resource).

Bit 1 on
Indicates that the resource manager program accepts connections with SECURITY(NONE).

Bit 1 off
Indicates that the resource manager program does not accept connections with SECURITY(NONE).

Bit 2 on
Defines the resource as a system resource. If this bit is on, bit 0 must also be on.

Bit 2 off
Indicates that this resource is not a system resource.

Bits 3-6
Reserved for IBM use.

Bit 7 on
Defines the resource as one that will communicate with CP and reply to requests from CP's support for the family of POSIX exec functions for information about POSIX security values.

If this bit is on, the resource must also be authorized to set another user's POSIX security values and must be prepared to accept traffic along the connection set up by this *IDENT request. See “Communicating with CP” on page 733 for details on this interface.

CP checks the system directory to ensure that the virtual machine is authorized to set other users' POSIX security values. If CP determines that the virtual machine is not authorized, or an error occurs while CP is attempting to check the authorization, CP turns off this bit, and the resource will not be able to participate in operations requested by CP's support for the family of POSIX exec functions.

Bit 7 off
Indicates that the resource does not wish to participate in set_UID and set_GID operations.

For **revoke** requests (FCODE=2):

Bit 0 on
Tells CP to revoke the global resource or gateway, known to the TSAF or CS collection. It must also be on for revoking a system resource (see bit 2).

Bit 0 off
Tells CP to revoke the local resource, known only to the local system.

Bit 2 on
Indicates that this resource is a system resource. If this bit is on, bit 0 must also be on.

Bit 2 off
Indicates that this resource is not a system resource.

Ntype
Indicates the type for NAME. An Ntype of 0 indicates a resource ID; An Ntype of 1 indicates a gateway name.

When you try connecting to *IDENT to manage or revoke a resource or gateway, CP checks the validity of the connection pending parameter list and checks the CP directory to verify that you are authorized to make the connection. CP severs your connection to *IDENT if it detects an error.

For a request to manage a resource or gateway, *IDENT also checks that the resource or gateway is not currently managed by another virtual machine. *IDENT accepts the connection if you are accepted as the resource or gateway manager, and your virtual machine gets a connection complete interrupt. So that you do not send any messages over the path, CP accepts connections to *IDENT by specifying QUIESCE=YES on its IUCV ACCEPT. Because *IDENT quiesces the path to your resource or gateway manager, *IDENT can never receive an incoming message on the path. If you issue an IUCV QUIESCE or an IUCV RESUME on the path, IUCV returns with no action taken. *IDENT severs the connection if your request to manage

the resource or gateway is rejected. If your connection is accepted, *IDENT may sever the connection later if another virtual machine revokes your management of the resource or gateway.

For a request to revoke a resource or gateway, *IDENT severs the connection to your virtual machine and severs the connection to the resource or gateway manager.

See “*IDENT Sever Reason Codes” on page 736 for all the sever reason codes used by the Identify system service.

Handling Connection Requests for the Resource or Gateway

If your virtual machine becomes a local, global, or system resource manager, APPC/VM lets other virtual machines connect to your virtual machine if they specify the resource ID on APPCVM CONNECT.

Assuming your virtual machine is enabled for interrupts, connection pending interrupts are routed to your virtual machine since you are registered as the manager of the resource. You, as the resource manager, can either accept the connection (using IUCV ACCEPT) or sever the connection (using APPCVM SEVER).

When a virtual machine becomes a gateway manager, APPC/VM lets other virtual machines connect to the gateway manager by specifying the gateway name in the connection parameter list extension on the APPCVM CONNECT.

Connections to the gateway name are routed to the registered gateway manager virtual machine. The gateway manager virtual machine can either accept the connection (using IUCV ACCEPT) or sever the connection (using APPCVM SEVER).

Communicating with CP

If the resource being identified is one that communicates with CP's support for the family of POSIX exec functions, then the parameter list passed to *IDENT must have bit 7 turned on in the flag byte on the **manage** request. Turning on this bit notifies CP that the application is capable of handling conversations across the *IDENT path.

In addition to the bit in the parameter list, the virtual machine must have authorization to set other users' UIDs and GIDs. To get this authorization, add the SETIDS option of the POSIXOPT directory statement to the virtual machine's directory entry. If this authorization is not present, and bit 7 is on, *IDENT will turn bit 7 off, and CP will not use the extended *IDENT interface and function.

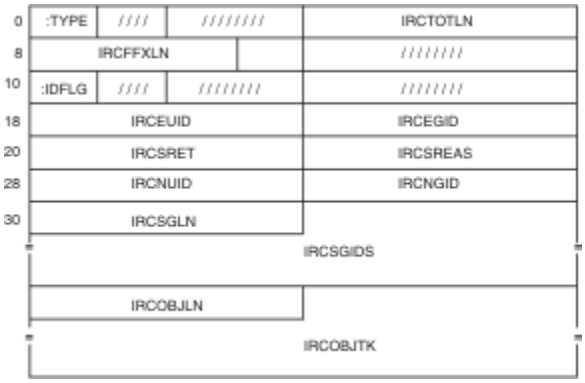
*IDENT Interface for Communication with CP's Support for the Family of POSIX exec Functions

When CP receives a request to process one of the POSIX exec functions for a set_UID and set_GID file, CP notifies the resource. The resource should provide any necessary authorization checking and notify CP of the results, sending the new UID and GID values if the authorization checking is successful.

The interface between CP and the resource is mapped by control block HCPIRCBK, which consists of two major sections: the header, and the function-specific section. The function-specific section is mapped differently for *IDENT communication type, and may include both fixed-length and variable length portions. For upward compatibility, the field IRCFFXLN, the length of the fixed-length portion of the function-specific section, must be filled in during the execution-time creation of the block. Any subsequent references to the variable-length portion must always be based on the address of the beginning of the block plus the lengths of the header and the fixed-length portion of the function-specific section.

The format of the header is:

The format of the IRCBK for the POSIX exec communication type is:



where:

IRCTYPE
is a 1-byte flag containing the type of communication with the resource. Valid values are:

1
Indicates that the information in the block is being communicated on behalf of CP's support for the family of POSIX exec functions.

IRCTOTLN
is an unsigned fullword containing the total length of the parameter list (the IRCBK) in bytes.

IRCFFXLN
is an unsigned fullword containing the length, in bytes, of the fixed-length portion of the function-specific section of the parameter list.

IRCIDFLG
is a 1-byte flag set by the resource containing information about the UID and GIDs. Valid values are:

1... ....
The UID should be changed.

.1.. ....
The GID should be changed.

..1. ....
Reserved

...1 ....
Reserved

.... 1...
Reserved

.... .1..
Reserved

.... ..1.
Reserved

....1
Reserved

If bit zero is on,
the object's access information specifies that the effective and saved-set UID of the process issuing one of the POSIX exec functions should be changed to match the value specified in the IRCNUID field.

If bit one is on,
the object's access information specifies that the effective and saved-set GID of the process issuing one of the POSIX exec functions should be changed to match the value specified in the IRCNGID field.

All other bits
are reserved and should be binary zero.

IRCEUID
is a fullword field containing the current effective UID of the process issuing one of the POSIX exec functions.

IRCEGID
is a fullword field containing the current effective GID of the process issuing one of the POSIX exec functions.

IRCSRET
is a fullword field set by the resource with the return code indicating success or failure of its checks. If this value indicates anything other than success (success being indicated by the server return code equal to zero), both bit zero and bit one of IDFLG are ignored.

CP will check this return code to determine whether it should complete its support for the family of POSIX exec functions or return an error. If this field contains zero, CP will continue. Otherwise CP will assume that an error has occurred which precludes CP's support for POSIX exec functions from completing successfully and will so notify the application issuing one of the POSIX exec functions.

IRCSREAS
is a fullword field set by the resource with the reason code associated with the server return code. CP does not examine this field, but passes it back to the application issuing one of the POSIX exec functions.

IRCNUID
is a fullword field set by the resource containing the UID to which the user's effective and saved-set UID should be set. If the server return code does not indicate success, or bit zero in byte IRCIDFLG is off, the contents of this field are ignored.

IRCNGID
is a fullword field set by the resource containing the GID to which the user's effective and saved-set GID should be set. If the server return code does not indicate success, or bit one in byte IRCIDFLG is off, the contents of this field are ignored.

IRCSGLN
is an unsigned fullword field containing the length of the supplementary GID data, in bytes. The value in this field must include the length of this field and the length of the supplementary GID information specified in the IRCSGIDS field.

IRCSGIDS
is a variable-length field containing the supplementary GIDs of the user whose process is issuing one of the POSIX exec functions.

IRCOBJLN
is an unsigned fullword field containing the length of the object token data, in bytes. The value of this field must include the length of this field and the length of the object token specified in the object token field.

IRCOBJTK
is a variable-length character field containing the token representing the file to be executed. The object token is left-justified.

When Your Resource is Revoked

The following can revoke a resource:

- A virtual machine authorized to revoke the resource
- A resource manager virtual machine (by severing its path to *IDENT)
- The TSAF virtual machine
- ISFC.

A virtual machine must be authorized to connect to *IDENT to revoke a resource or gateway.

To revoke a resource or gateway on your virtual machine, your virtual machine must issue an IUCV SEVER on the path to *IDENT.

The SEVER does not affect existing APPC/VM paths to your virtual machine. However, CP does not establish any new paths to you. If another virtual machine connects to *IDENT to manage the resource or gateway that you revoked, requests to connect to the resource go to that virtual machine.

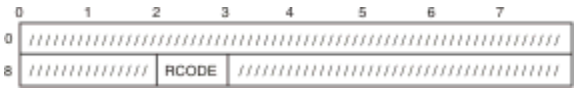
Note: If a virtual machine initiates a connection request to a resource that you manage before your revoke completes, the path may be established.

*IDENT Sever Reason Codes

When *IDENT severs one of its paths, it stores a reason code in byte 10 of the IPUSER field in the IUCV SEVER external interrupt buffer. This code indicates the reason for the SEVER.

Note: If the virtual machine issues the SEVER, it does not receive an external interrupt; in this case the *IDENT reason code is X'00'.

The IPUSER field has the following format:



RCODE
contains the *IDENT sever reason code. It can be one of the following values:

Hex Code	Decimal Code	Meaning
X'00'	0	*IDENT revoked the resource or gateway as requested.
X'01'	1	An I/O error occurred while CP was reading the system directory. *IDENT was checking for authorization to manage or revoke the resource or gateway.
X'02'	2	The IPUSER field in the IUCV CONNECT parameter list was set up incorrectly for a connection to *IDENT.
X'03'	3	The CONNECT parameter list has an invalid parameter PRMDATA.
X'04'	4	The virtual machine is not authorized to connect to *IDENT for the specified resource or gateway.
X'05'	5	The virtual machine is not authorized to identify the resource as a global or system resource.
X'06'	6	The virtual machine is not authorized to revoke the specified resource or gateway.
X'07'	7	The virtual machine is not authorized to revoke the specified resource globally.
X'08'	8	CP cannot identify the global resource or gateway because the CP resource/gateway table currently contains the maximum of 500 entries owned on the local system.
X'09'	9	A virtual machine already manages the resource or gateway being identified. The virtual machine trying to identify a resource or gateway could be the same virtual machine that already manages the resource or gateway.

Hex Code	Decimal Code	Meaning
X'0A'	10	A virtual machine revoked the resource or gateway. The resource or gateway may have been revoked by the virtual machine that managed the resource or gateway.
X'0B'	11	The resource or gateway to be revoked does not exist.
X'0C'	12	The resource or gateway is pending identification by a virtual machine and is not available to be identified or revoked.
X'0D'	13	The resource or gateway is pending a revoke by a virtual machine and is not available to be identified or revoked. An authorized virtual machine is revoking the resource or gateway, or the resource manager virtual machine is severing its path to *IDENT.
X'0E'	14	The CONNECT parameter list has an invalid resource name or gateway name specified.

Chapter 17. Message System Service (*MSG)

The Message system service (*MSG) lets a virtual machine read messages and responses that come from CP, rather than display these on the terminal. (The texts of CP command responses and messages themselves is not part of the programming interface.)

*MSG is the assigned Message system service user ID. You establish communication with this user ID by specifying USERID=*MSG when issuing the IUCV CONNECT function.

Your virtual machine does not need any special authorization to use the Message system service. All you have to do is issue IUCV CONNECT with USERID=*MSG. A virtual machine may only have one communication path to the *MSG system service.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

Once you have issued this, you must specify what kind of messages you want the virtual machine to receive. The Message system service handles virtual console output if the user has specified the IUCV option on the CP SET command, as shown in Table 204 on page 739. There is no SET command for Single Console Image Facility (SCIF) messages; the virtual machine always receives these when it has a connection to the Message system service.

Note: When full-screen CMS is on, most CMS console output is not passed to CP. In addition, applications that use the Message system service and SET VMCONIO IUCV do not trap all CMS output. Before running such applications, it is recommended that you suspend full-screen CMS.

Following is a list of CP SET commands that have an IUCV option. All of these commands assume that you have a connection to IUCV and *MSG.

Table 204. CP SET Commands with an IUCV Option

CP command	Options	Function
SET MSG	IUCV	Passes all messages to the virtual machine using IUCV. The messages do not go into the virtual console spool file.
SET WNG	IUCV	Passes warnings to the virtual machine
SET SMSG	IUCV	Passes special messages along to the virtual machine using IUCV
SET EMSG	IUCV	Passes on both text and error code to the virtual machine using IUCV
SET IMSG	IUCV	Passes on information messages to the virtual machine using IUCV
SET VMCONIO	IUCV/OFF	Passes along responses to the virtual machine using IUCV. If you do not have an IUCV connection, data is handled as if VMCONIO is set to OFF. (Refer to the note preceding this table.)
SET CPCONIO	IUCV/OFF	Passes along all CP responses to the virtual machine using IUCV. If there is not an IUCV connection, data is handled as if CPCONIO is set OFF.

The Message system service uses the IUCV default maximum value of 255 for the number of outstanding messages allowed on the path. The number of outstanding messages can be set below the default by specifying a value for MSGLIM on the IUCV CONNECT to *MSG. The default maximum value can be changed to 16000 by including an IUCV *MSG statement in the directory entry for the user issuing the connect (the MSGLIMIT parameter is ignored). If the message limit is exceeded, any additional incoming

© Copyright IBM Corp. 1991, 2020

messages are routed directly to the virtual machine console, or alternate console, and the virtual machine is not notified about these messages. This situation is most likely to occur when there is a high volume of incoming messages and the virtual machine is running with external interrupts disabled.

The Message system service identifies the source of the message it intercepts by a code in the IUCV message class field. The message source is interpreted as follows:

Class
Message Source

1
Message sent using CP MESSAGE (MSG) or CP MSGNOH

2
Message sent using CP WARNING (WNG)

3
Asynchronous CP messages, CP messages to a CP command executed by a virtual machine using *MSG, and any other console I/O initiated by CP

4
Message sent using CP SMSG command

5
Any data directed to the virtual console by the virtual machine (for example, WRTERM or LINEDIT).

6
Error message from CP (EMSG)

7
Information messages for CP (IMSG)

8
Single console image facility (SCIF) message from CP.

Error and information messages (classes 6 and 7) are types of CP messages and are included in class 3 when EMSG and IMSG are not specifically set to IUCV through the CP SET commands.

The format of the data received from IUCV is as follows:

User ID identifies the sender. *Text* is the actual data the user would have received on their terminal. For class 1 and class 2 type messages, the standard header is suppressed. If the data is not received by a MSG, WNG, SMSG, or using SCIF, the user ID is the recipient.

If a virtual machine has both a valid path to *MSG and a functioning secondary user, incoming messages (except for SMSGs, which are not console messages) are directed to the secondary user instead of the IUCV *MSG path to the primary user.

If a secondary user has a valid path to the Message system service (*MSG) and is disconnected, then output on behalf of the primary user normally directed to the secondary user's console is instead directed through the IUCV *MSG path.

Any output generated by the SET LOGMSG and ECHO commands is always sent to the terminal. The output is not sent over *MSG.

Chapter 18. Message All System Service (*MSGALL)

Like the Message system service, the Message All system service (*MSGALL) lets a virtual machine read messages and responses that come from CP, rather than displaying these on the terminal.

The Message All system service operates as an alternative to the Message system service. If you use *MSGALL, all terminal output is received over IUCV. If the user has issued the CP SET command with the IUCV option, that setting overrides *MSGALL. For example, if the user has issued SET EMSG IUCV, all EMSGs use the *MSG path, rather than the *MSGALL path. All the other terminal output still follows the *MSGALL path.

Note: SMSGs are never sent on the *MSGALL path. They only use the *MSG path.

Your virtual machine does not need any special authorization to use the Message All system service. All you have to do is issue IUCV CONNECT with USERID=*MSGALL. A virtual machine may only have one communication path to the *MSGALL system service.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

Console output is handled as follows:

Console output sent over the *MSGALL path if unsuccessful with *MSG:

- CPCONIO and EMSGs generated as part of a DIAGNOSE code X'08' operation
- MSGs, WNGs, IMSGs, and SCIFed messages.

Console output sent directly to the terminal:

- Asynchronous CPCONIO (including TRACE events) and EMSGs **not** generated as part of a DIAGNOSE code X'08' operation.
- Output generated by the SET LOGMSG and ECHO commands.

Console output never sent over *MSGALL path:

- SMSGs

The Message All system service uses the IUCV default maximum value of 255 for the number of outstanding messages allowed on the path. The number of outstanding messages can be set below the default by specifying a value for MSGLIM on the IUCV CONNECT to *MSGALL. The default maximum value can be changed to 16000 by including an IUCV *MSGALL statement in the directory entry for the user issuing the connect (the MSGLIMIT parameter is ignored). If the message limit is exceeded, any additional incoming messages are routed directly to the virtual machine console, or alternate console, and the virtual machine is not notified about these messages. This situation is most likely to occur when there is a high volume of incoming messages and the virtual machine is running with external interrupts disabled.

© Copyright IBM Corp. 1991, 2020

***MSGALL**

Chapter 19. SCLP System Service (*SCLP)

The SCLP system service (*SCLP) is a CP system service that allows you to receive and transmit Hardware Management Console events. When you connect to the SCLP system service, you register to handle the events for one or more event classes. For a virtual machine to transmit and receive Hardware Management Console events, it must first be authorized to connect to the SCLP system service, *SCLP. Your system administrator is the person who can authorize a virtual machine to establish a connection. To do this, the administrator must specify a special IUCV *SCLP statement in the virtual machine's directory entry. For more information on how a system administrator authorizes virtual machines to connect to IUCV system services, see *z/VM: CP Planning and Administration*.

For more information on the IUCV functions mentioned in this section, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

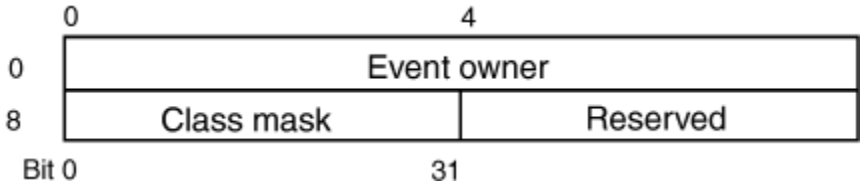
Establishing Communication with the SCLP System Service

The SCLP system service uses IUCV to communicate with a virtual machine. The IUCV macro checks the validity of the IUCV parameters and any errors are handled according to IUCV specifications. The SCLP system service checks the validity of all the parameters it requires. Any errors resulting from this check are handled as described in the remainder of this section.

The first step in establishing IUCV communications with the SCLP system service is to issue an IUCV DECLARE BUFFER. This function initializes the virtual machine for IUCV communication. This function also specifies a buffer where IUCV can store external interruption information.

Connecting to the SCLP System Service

After initializing for IUCV communication, you must issue an IUCV CONNECT and specify USERID=*SCLP in the IUCV CONNECT parameter list (IPARML). The user data field must have the following format:



Event owner
Is a string of eight bytes identifying the owner of the event class and must be "SYSTEM " (two blanks are required at the end).

Class mask
Is a field of 32 bits defining the event classes your virtual machine registers to handle. Bit zero corresponds to event class 1, bit one to event class 2, and so on. Only one virtual machine can be registered for a particular event class.

If the connection request is rejected, the first byte of the user data field has the return code indicating the reason the connection was severed. The following return codes result from connection errors:

Code
Meaning

1
The `Event owner` field value is not "SYSTEM ".

2
The `Class mask` field value is zero.

© Copyright IBM Corp. 1991, 2020

3
The Reserved field value is not zero.

4
The Class mask field includes an unsupported class.

5
The Class mask field includes a class that is already registered.

6
The IUCV ACCEPT failed.

Sending Events

Upon notification of a successful connection, your virtual machine is ready to send events. You may now issue synchronous (SYNC=YES) 1-way (TYPE=1WAY) IUCV SEND requests to the SCLP system service specifying the target event class mask and the event data. Specify the target event class mask as the target class (TRGCLS) in the format of a Class mask (in the mask, only one bit should be on). Specify the BUFFER and BUFLEN parameters on the IUCV SEND.

If a serious error is encountered, the *SCLP connection is severed. The first byte of the user data field has the return code indicating the reason the SEND was rejected. The following return codes result from send errors:

Code
Meaning

7
A target event class was not specified.

8
The event was not sent as a synchronous 1-way message.

9
No event buffer was supplied or DATA=PRMMSG was specified.

10
No connection exists to the *SCLP system service.

11
The original CONNECT request did not enable the specified target event class.

If any other error is encountered, the SEND completes with condition code 1 and an error return code in the IPRCODE field. The following error codes might be presented:

Code
Meaning

3
The message could not be sent to the HMC.

5
The message is too long.

6
A fetch protection exception was detected on the send buffer.

7
An addressing exception was detected on the send buffer.

22
The send buffer list is invalid.

23
The buffer list contains a negative length.

24
The total of the buffer list lengths is incorrect.

26
The buffer list is not on a doubleword boundary.

30
The IPAPPC flag in IPFLAGS1 is not 0.

31
An IUCV function was specified on an APPC/VM path.

91
A paging or storage error was detected in the SEND data area.

If return code 3 is received, a Message Complete external interruption will be presented for the message.

Receiving Events

The SCLP system service passes events to your virtual machine one at a time by issuing a 1-way IUCV SEND. The target class (TRGCLS) of the message is the event class in the same format as a `Class mask`. If you want to stop receiving events from the SCLP system service temporarily, you can use IUCV QUIESCE. The SCLP system service queues events while a path is quiescent. The queued events will be available when you issue an IUCV RESUME.

Disconnecting from the SCLP System Service

When all communications with the SCLP system service are completed, you can terminate communication by issuing either an IUCV SEVER or an IUCV RETRIEVE BUFFER.

Chapter 20. Signal System Service (*SIGNAL)

The Signal system service (*SIGNAL) is a CP system service that allows virtual machines in a virtual machine group to signal each other. The Signal system service can only be used by virtual machines in a virtual machine group. Each virtual machine in a group is identified by a unique 16-bit signal ID. When a virtual machine connects to the Signal system service, it may request that a particular signal ID be assigned to it. If you have not set up the virtual machine to request a specific signal ID, the Signal system service automatically assigns one to your virtual machine.

All members of a virtual machine group can send eight bytes of signal data (user information) to any member in the group, specifying the signal ID of the virtual machine they want to receive the signal data on, by using IUCV SEND. A virtual machine can also signal all members in a group using a broadcast signal. Group members can request notification of members entering and leaving the group by specifying Signal-In and Signal-Out flags when they connect to the Signal system service.

Using the Signal system service requires no directory authorization. The Signal system service allows only one connection per virtual machine.

For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

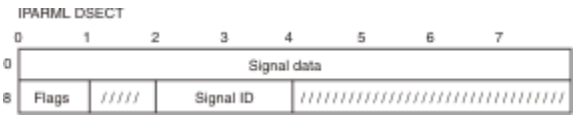
Establishing Communications with the Signal System Service

The Signal system service uses IUCV to communicate between itself and a virtual machine. The IUCV macro checks the validity of all the IUCV parameters and any errors are handled according to IUCV specifications. The Signal system service checks the validity of all the parameters it requires. Any errors resulting from this check are handled as described in the following sections.

Your first step in establishing IUCV communications with the Signal system service is to issue an IUCV DECLARE BUFFER. This initializes the virtual machine for IUCV communication. This function also specifies a buffer where IUCV can store external interrupt information.

IUCV CONNECT to the Signal System Service

After you establish communications with IUCV, you must issue an IUCV CONNECT with USERID=*SIGNAL and PRMDATA=YES in the IUCV CONNECT parameter list (IPARML). The user data field must have the following format:



Signal data
is the eight bytes of user information or signal you want passed to other members of the group. Only group members that have specified the signal-in flag when they connected receive the data.

Flags
is a set of bits defining the signal options chosen by you for your virtual machine. The first three bits are defined, and the others are reserved. The defined bits are:

Code	Meaning
X'80'	Signal-in
X'40'	Signal-out

© Copyright IBM Corp. 1991, 2020

X'20'
Signal ID has been specified

X'1F'
Reserved.

Signal ID
is the signal ID you want assigned to your virtual machine. This signal ID is used by other group members to communicate with your virtual machine. This field is only used if you set the signal ID flag bit (X'20') in the flags field.

You must set all reserved fields and flags to zero.

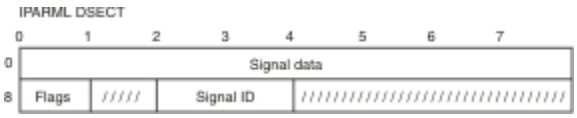
If you specify the signal-on flag, your virtual machine is signaled when future group members enter your group by connecting to the Signal system service.

If you specify the signal-out flag, your virtual machine is signaled when group members in your group break their connection (using IUCV SEVER) with the Signal system service.

The IUCV CONNECT function returns a PATHID to your virtual machine. You must specify this PATHID in the IUCV SEND parameter list (IPARML) for all subsequent communication to the Signal system service.

When you issue IUCV CONNECT to the Signal system service, the connection is either accepted (the Signal system service issues an IUCV ACCEPT) or severed (the Signal system service issues an IUCV SEVER).

If the connection is accepted, the user data field on the IUCV connection complete external interrupt has the following format:



Signal data
is unchanged from the IUCV CONNECT.

Flags
are unchanged from the IUCV CONNECT.

Signal ID
is the signal ID you assigned to your virtual machine. If you did not assign a signal ID, the Signal system service assigns a unique signal ID for you and stores it in this field.

All unused fields remain unchanged from the IUCV CONNECT.

If the connection is rejected, the user data field on the IUCV SEVER external interrupt has the following format:

IPRCODE
is the return code indicating the reason the connection was severed.

Return codes resulting from connection errors:

Code
Meaning

X'01'
You are not a member of a virtual machine group.

X'02'
You are already connected to the Signal system service.

X'03'
You did not specify PRMDATA=YES in the IUCV CONNECT parameter list (IPARML).

X'04'
The reserved fields were not set to zero.

X'05'
The signal ID you specified was not unique.

Sending Signals

Upon notification of a successful connection, your virtual machine is ready to send signals. You may now issue IUCV SEND requests to the Signal system service specifying the 8-byte signal (parameter list data), the target's signal ID, and the flag settings. Specify the target's signal ID and the flag settings in the target class (TRGCLS) with the following format:

Flags
is a set of bits defining the handling of the signal. Only two bits are defined and the others are reserved. The defined bits are:

Code
Meaning

X'10'
Broadcast signal

X'08'
Invalid signal ID

X'E7'
Reserved.

Signal ID
is the target's signal ID.

If you specify the broadcast signal (X'10') flag, a signal is sent to all of the other users in your group that are connected to the Signal system service.

If you send a signal using an invalid signal ID, the Signal system service returns the signal to you with an error indicator (X'08') in the FLAGS field. The target class and the signal data remain unchanged.

If the parameter list data option is not used, or if the signal is not 1-way, the connection is severed.

Return codes resulting from send errors:

Code
Meaning

X'06'
The signal was sent without the DATA=PRMMSG option specified.

X'07'
The signal sent was not a 1-way signal.

Receiving Signals

As a member of a virtual machine group, your virtual machine can receive three types of signals. These are signal-in, signal-out, and a normal signal sent by another group member using an IUCV SEND. The Signal system service passes these signals to your virtual machine through an IUCV SEND using a 1-way message with the signal specified in the parameter list data.

Specify the source's signal ID and the flag settings in the target class (TRGCLS) with the following format:

Flags
is a set of bits defining the type of signal sent. Only three bits are defined and the others are unused. The defined bits are:

Code	Meaning
X'80'	Signal-In
X'40'	Signal-Out
X'10'	Broadcast signal
X'2F'	Unused

Signal ID
is the source's signal ID.

The Signal system service sets all unused fields to zero.

If the signal-in flag is on, this signal was specified in the user data of the user's IUCV CONNECT to the Signal system service.

If the signal-out flag is on, this signal was specified in the user data of the user's IUCV SEVER to the Signal system service.

If the broadcast flag or no flags are on, this signal was specified in the parameter list data of the user's IUCV SEND to the Signal system service.

If you want to stop receiving signals from the Signal system service, you can use IUCV QUIESCE. However, the Signal system service does not queue signals, and the signals from other members of the group will be lost until you issue an IUCV RESUME.

Leaving the Signal System Service

When all communications with the Signal system service are completed, you can terminate communication by issuing either an IUCV SEVER or an IUCV RETRIEVE BUFFER. The user data field on a SEVER must have the following format:

Signal data
is the 8-byte signal you want passed to other members of the group. Only group members that have specified the signal-out flag when they connected to the Signal system service will receive the data.

If a SEVER is generated by the CP system, as on a RETRIEVE BUFFER or a virtual machine reset, the signal data is set to all zeros.

Chapter 21. Spool System Service (*SPL)

The Spool System Service is a CP system service that provides an interface using IUCV for communication between CP and virtual machines that provide print services for advanced function printers, and reader services for reader notification.

*SPL also functions as a generic spool file interface facility that you can use instead of DIAGNOSE code X'14' to select and read files from the CP spool system.

When the virtual machine connects to the spool system service, for printer service it passes an identifier, *prtid*, which is used as the external name of that virtual machine. When the virtual machine connects to the spool system service, for reader service the IPUSER area (first 8 bytes) must be blank (X'40'), and the 9th byte defines a flag byte (IPUFLAG1). A bit is defined in this byte (IPRDR = X'80') to indicate that the connect request is for the reader notification function. For *SPL to act as a generic spool file interface, the virtual machine must pass the keyword VIRTUAL in place of the *prtid*.

The term *prtid* is synonymous with print server virtual machine in this document. The virtual machine has access to the system printer spool files but these printers, of course, are not supported directly by CP.

Once communication between the *prtid* and the spool system service has been established, a *prtid* can:

- Select a spool file from the print chain for processing
- Close a selected file
- Send messages or command responses to the operator or other users
- Read the spool file descriptor (SFBLOK) (see note “1” on page 751)
- Read the spool file records (SPLINKs) (see note “2” on page 751)
- Read an external attribute buffer (XAB) (see note “3” on page 751).

Notes:

1. For more information about SFBLOK, refer to Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969.
2. For more information about SPLINK, refer to Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969.
3. The external attribute buffer (XAB) is a control block that contains data the user creates to specify additional information about a print file. Each print file has its own XAB, and CP has the facilities to maintain the XABs. For more information about XABs, refer to “External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'” on page 979.
4. For more information on the IUCV functions mentioned in this chapter, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

Similarly, the spool system service communicates with the *prtid* and can:

- Send commands to the *prtid*
- Notify the *prtid* that a print file is available for processing
- Cancel processing on a print file (PURGE function).

For the second option, you can enter the word VIRTUAL in the field normally used for the *prtid* that the PSF machine represents. VIRTUAL indicates that you want the spool system service to process files in the virtual reader, printer, or punch of a given user. This option enables an application to:

- Read files out of its virtual reader, virtual printer, or virtual punch
- Given the right privilege classes, read files out of another user's virtual reader, virtual printer, or virtual punch

© Copyright IBM Corp. 1991, 2020

- Allow for selection of files based on the setting of *seen* bits and allow for the *seen* bits to be reset on files.

This function of *SPL does not require any change in existing applications, such as PSF, that use *SPL.

Depending on whether you specify VIRTUAL or a *prtid* in the field, all subsequent IUCV SEND, RECEIVE and SEVER functions issued by the application take on different meanings. For instance, the IUCV transmission class used by an application for selecting a spool file will differ depending on the type of IUCV connection established. IUCV connections established using the VIRTUAL keyword will be referred to as the generic *SPL interface, while connections established using a *prtid* will be referred to as the Advance Function Printers (APF) printing *SPL interface. The pages that follow describe each of these communication functions in more detail.

The AFP Printing *SPL Interface

The AFP Printing *SPL Interface provides print services.

Establishing Communication with the Spool System Service

The spool system service uses IUCV to communicate with the print server virtual machine, and reader server virtual machine. For you to use the spool system service, the directory entry for the virtual machines must specify the IUCV *SPL option. *SPL is the SPOOL system service user ID. The IUCV macro checks the validity of all IUCV parameters. All IUCV errors are handled according to IUCV specifications. The spool system service checks the validity of all required parameters and handles the errors. IUCV requires the virtual machine to do a DECLARE BUFFER to initialize the virtual machine for IUCV communication. The virtual machine does an IUCV CONNECT to the spool system service. The connection lets the *prtid* use the spool system service to obtain spool print files for processing. Operator and user commands can be sent to the server virtual machines through the IUCV connection by specifying the *prtid* defined by the virtual machine at CONNECT time.

IUCV CONNECT to the Spool System Service

For Printer Service

An IUCV CONNECT is issued by the virtual machine with USERID=*SPL, PRTY=NO, PRMDATA=YES, and USERDTA is an address containing the *prtid* name.

A *prtid* name is a 1- to 8-character alphanumeric name your installation assigns. If you specify a *prtid* name of four or fewer characters, then the name *cannot* contain all hexadecimal characters. For example, FFFF is **not** a valid *prtid* name, whereas FFFFF is. BLD is another example of a valid name. A *prtid* name cannot be ALL. A *prtid* name must **not** contain imbedded blanks; however, trailing blanks are permitted. It is the installation's responsibility to control the names assigned to *prtids*. The name should not conflict with any other names, options, or notation the installation uses.

Example

It is recommended that a *prtid* name should not:

- Be the same as a user ID on the system
- Be *PROC* or *PROCESSOR*
- Include a dash (-)
- Be a name that could be confused with a logical device number, for example, L1234.

The spool system service allows multiple virtual machines to be connected with the same *prtid* name. This way, when you use the spool system service SEND function, each virtual machine of the specified *prtid* name is sent a message.

If you want to create your own IUCV parameter list, then the IPUSER user data field must have the following data:

```
         ORG   IPUSER
SPLNAME  DC    CL8       PRTID
```

For Reader Service

An IUCV CONNECT is issued by the virtual machine with USERID=*SPL, PRTY=NO, PRMDATA=YES, and USERDTA=IPUSER(1:9). The IPUSER field from bytes 1 to 9 contains the following information:

```
RDRID      DS   CL8      Set to blanks

Bits Defined in IPRFLAG1

IPRDR      EQU  X'80' Connection for the reader notify function
```

If all the parameters required by Spool system service are valid, the Spool system service issues an IUCV ACCEPT to complete the connection with the virtual machine. All other Spool system service functions depend on the IUCV CONNECT and ACCEPT being complete.

Reasons the CONNECT Function May Sever

If any of the parameters passed to Spool system service are invalid, Spool system service issues an IUCV SEVER. When the *prtid* is ready to end its connection with *SPL system service, the *prtid* then issues an IUCV SEVER. Any spool files being processed at the time of the SEVER remain unchanged.

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to the *prtid*:

- A CONNECT issued was with QUIESCE, return code X'80'
- A CONNECT issued was with PRTY, return code X'80'
- A CONNECT issued was with PRMDATA=NO, return code X'80'
- A CONNECT issued was with a *prtid* name, that is:
  - ALL
  - Contains imbedded blanks
  - Is four or less hexadecimal digits.

When the Spool system service CONNECT issues a SEVER, the format of the IPUSER field is:

- IPUSER+0: 1-byte error code of X'80'
- IPUSER+1: 1-byte reserved field of X'00'
- IPUSER+2: 2-bytes, containing IPMSGID from the IUCV parameter list that caused the error.

When the reader service issues a CONNECT, and the Spool system service has 8 (maximum) connections with reader service machines, then *SPL issues a SEVER, the format of the IPUSER field is:

- IPUSER+0: 1-byte error code of X'80'.

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to the reader service machine:

- A CONNECT issued was with QUIESCE, return code X'80'
- A CONNECT issued was with PRTY, return code X'80'
- A CONNECT issued was with PRMDATA=NO, return code X'80'.

When the Spool system service CONNECT issues a SEVER, the format of the IPUSER field is:

- IPUSER+0: 1-byte error code of X'80'.

Virtual Machine Communication to the Spool System Service

After the Spool system service accepts the connection (using an IUCV ACCEPT), the virtual machine can send the following types of requests to the Spool system service with an IUCV SEND:

- SELECT
- CLOSE
- MESSAGE
- READ
  - READ SFBLOK (see Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969)
  - READ XAB (see “External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'” on page 979)
  - READ SPLINK (see Appendix A, “Data Areas Used by DIAGNOSE Codes,” on page 969)
- ENABLE
- DISABLE.

All calls from the virtual machine to *SPL are 2-way sends. The material that follows describes these functions:

The SELECT Function

Through a file-select, a *prtid* can obtain a spool print file for processing. To select a file to process, the *prtid* sends a SELECT (through an IUCV SEND specifying TYPE=2WAY, DATA=PRMMSG or DATA=BUFFER, the value of 4 in the address specified by TRGCLS) to the Spool system service.

The SELECT function does not require that a previously selected file be closed before another file is selected.

The spool print file can be selected either by specific file (user ID or spool ID), by the generic selection criteria, or by default. In other words, select by:

- Specific criteria (user ID and spool ID), DATA=BUFFER
- Generic criteria, DATA=BUFFER
- Previously specified (default) criteria, DATA=PRMMSG.

SELECT with Specific or Generic Criteria (DATA=BUFFER)

When DATA=BUFFER (the user data is in the buffer and IUCV RECEIVE transfers the data), a length of 60 in the address specified by BUFLEN, and the buffer contains the following information:

Note: Field names used in the following descriptions are used for reference only. You may copy these names, or use any other names, in your own code.

```
SPLSEFLG   DS   1X       Selection flag
SPLSETYP   DS   1X       Type of file eligible for selection
SPLSECLS   DS   4CL1     CLASSes
SPLSEFSH   DS   CL4      FLASH
SPLSEFRM   DS   CL8      FORM (Operator)
SPLSEDES   DS   4CL8     Destinations
SPLSUSID   DS   CL8      User ID (if flag SPLSESID is ON)
SPLSPID    DS   H        Spool ID (if flag SPLSESID is ON)

Bits Defined in SPLSEFLG

SPLSECHG   EQU  X'80'    Change default selection criteria
                         to the new selection criteria sent in the
                         BUFFER.
                         Note:  All selection criteria is changed
                         so all data must be specified in the buffer.
SPLSECON   EQU  X'40'    Select spool file for "convert"
                         process.
```

```
Bits Defined in SPLSETYP

SPLSE5AC   EQU  X'80'    Select files using X'5A' CCWs
SPLSEN5A   EQU  X'40'    Select files not using X'5A' CCWs
SPLSECNV   EQU  X'20'    Select files that are converted
SPLSENCV   EQU  X'10'    Select files that are not converted
SPLSN38L   EQU  X'08'    Select files not using 3800 load CCWs
SPLSBE38   EQU  X'04'    Select files containing 3800 load CCWs
                         at the beginning of the file
SPLSAN38   EQU  X'02'    Select files containing 3800 load CCWs
                         anywhere within the file
SPLSESID   EQU  X'01'    Select a specific spool file by ID
                         (used in conjunction with the SPLSPID
                         and SPLUSID)
```

If the three flags—SPLSN38L, SPLSBE38, and SPLSAN38— are all OFF, then the 3800 load CCW characteristics of the spool file do not affect whether the file is selected (the tests are bypassed).

Notes:

1. If the SPLSESID bit is off, then select the next available file matching the default selection criteria.
2. If the user ID is less than 8 characters, it must be left-justified and padded with blanks.
3. If the select is by spool ID, the selection criteria are not changed.
4. If the SPLSECON bit is off, then select a spool file to start the print process.

   A file that is selected for print processing is similar to files being handled by CP-driven system printers. The file is marked as being open for system use.

   A file that is selected for conversion processing is special because the *prtid* for which the file was selected is notified when CP CHANGE, CP PURGE, and CP TRANSFER commands are issued for the file. See the Spool system service PURGE function ("The PURGE Function" on page 765.)
5. When the select is not by spool ID, you must specify at least one each of CLASS, FLASH, FORM, DESTination, either SPLSE5AC or SPLSEN5A, and either SPLSECNV or SPLSENCV in SPLSETYP.

   If you specify a value of blank for CLASSes, FLASH, FORM, or DESTinations, then no file is selected.
6. Any unused length of the CLASSes, FLASH, FORM, or DESTinations fields must be padded with blanks.
   - An asterisk (*) specified for a CLASS, means that any CLASS file can be selected.
   - An asterisk (*) specified for FORM (or DESTination) means that any FORM (or DESTination) file can be selected.
   - An asterisk (*) specified for FLASH means that any FLASH file can be selected.
   - OFF specified for FORM means that a file is eligible for selection if it has either been assigned a FORM of OFF or if it has been assigned the default printer FORM. For more information about setting default forms, see the PRINTER operand on the FORM_DEFAULT statement in the system configuration file. The FORM_DEFAULT statement is described in *z/VM: CP Planning and Administration*.
   - OFF specified for DESTination means that files which do not have a destination assigned to them, or have been assigned OFF, can be selected.
   - OFF specified for FLASH means that files which do not have a FLASH assigned to them, or have been assigned OFF, can be selected.
7. If SPLSESID is specified, then no other bits in SPLSETYP may be specified.

REPLY Information

If a file is selected, the Spool system service always responds using an IUCV REPLY with DATA=BUFFER and a length of 14 in the address specified by ANSLEN.

ANSBUF=*address*, where *address* points to a buffer that contains the following information:

```
IPUSER + 0:   DS   H    Spool ID
IPUSER + 2:   DS   H    COPY count for the file or zero
IPUSER + 4:   DS   XL1  Primary flag = X'00'
IPUSER + 5:   DS   XL1  Secondary flag
```

```
IPUSER + 6:   DS   CL8  User ID

Meaning of Bits in Secondary Flag
EQU  X'80'   SFBLOK marked to be purged
EQU  X'40'   File has been converted
EQU  X'20'   File contains X'5A' CCWs
EQU  X'10'   Reserved
EQU  X'08'   Reserved
EQU  X'04'   Reserved
EQU  X'02'   Reserved
EQU  X'01'   Reserved
```

If no file is selected, the Spool system service always responds using an IUCV REPLY with DATA=PRMMSG. PRMMSG contains the following information:

```
IPRMMSG1   Bytes 0 to 3       Reserved
IPRMMSG2   Byte 0             Primary flag = X'04'
           Byte 1             Secondary flag
           Bytes 2 to 3       Reserved

Meaning of Bits in Secondary Flag
EQU  X'80'    Reserved
EQU  X'40'    Reserved
EQU  X'20'    Reserved
EQU  X'10'    Specified file not found on PRINT queue
EQU  X'08'    Specified file in USER HOLD status
EQU  X'04'    Specified file in SYSTEM HOLD status
EQU  X'02'    File is in the process of being printed
EQU  X'01'    Reserved
```

The select was by spool ID; the file was not selected and has one of the characteristics shown in the secondary flag.

SELECT by Previously-Specified Criteria (DATA=PRMMSG)

When DATA=PRMMSG (select using default selection criteria), then PRMMSG contains the following information:

```
IPRMMSG1   Bytes 0 to 3       Reserved
IPRMMSG2   Byte 0             Flag
           Bytes 1 to 3       Reserved

Bits Defined in Byte 0 of IPRMMSG2
           EQU  X'80' Reserved
SPLSECON   EQU  X'40' If set ON, select spool file for "convert" process
                      If set OFF, select file for "print" processing
```

All other bits are reserved and must be set to zero.

Notes:

1. A file that is selected for print processing is similar to files that are being handled by CP-driven system printers. The file is marked as being open for system use.

   When a file is selected for convert processing, the *prtid* for which the file was selected is notified if the CP CHANGE, CP PURGE, and CP TRANSFER commands are issued for the file. This is so the print server can discard the current conversion and redo it in light of the new characteristics. See the Spool system service PURGE function (“The PURGE Function” on page 765.)
2. Default selection criteria is defined by the last issued SELECT.

Reasons the SELECT Function May Sever

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to a *prtid*:

Return code=X'80':

- Use of TYPE=1WAY communication
- Incorrect buffer length. The specified buffer length on the SEND must be 60 bytes.
- Conflicting selection criteria options:

  - DATA=PRMMSG, and default selection criteria are not defined.
  - DATA=BUFFER, and both the SPLSE5AC and SPLSEN5A flags are OFF.
  - DATA=BUFFER and both the SPLSECNV and SPLSENCV flags are OFF.
- Incorrect flag specification. No other bit in SPLSETYP can be ON if SPLSESID is ON.

When the Spool system service SELECT does a SEVER, the format of the IPUSER field length is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'04' -  Identifies the type of IUCV
              SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
              that caused the error
```

The CLOSE Function

A *prtid* uses the Spool system service CLOSE function to take one of these actions:

1. End the processing of a selected file:

   The CLOSE functions are performed if a new copy count is not requested.

   In this case, when the CLOSE is complete, the *prtid* can specify to delete the print file or leave it on the VM spool.
2. Change the spool file copy count:

   If SPLCCOPY is greater than 0 and less than X'00FF', the change function is performed. (If SPLCCOPY is greater than X'00FF', then the Spool system service severs the IUCV path.)

To use the CLOSE function, the *prtid* sends a CLOSE request (using an IUCV specifying SEND TYPE=2WAY, DATA=BUFFER, a length of 30 in the address specified by BUFLEN, and a value of 5 in the address specified by TRGCLS) to the Spool system service.

The buffer contains the following information:

```
SPLCSPID   DS   1H       File spool ID
SPLCFLCL   DS   1X       Flag (If the function is to change the copy
                         count, then this flag is ignored.)
SPLCCLAS   DS   1X       CLASS
SPLCFORM   DS   CL8      FORM
SPLCDEST   DS   CL8      DESTination
SPLCCOPY   DS   1H       Copy count
SPLCUSID   DS   CL8      User ID of the owner of the spool file

Bits Defined in SPLSFLCL
SPLCLPUR   EQU  X'80' Close and purge file
SPLCLUHO   EQU  X'40' Close and requeue in user hold
SPLCLSHO   EQU  X'20' Close and requeue in system hold
SPLCLCON   EQU  X'10' Close and requeue converted
SPLCLCLA   EQU  X'08' Close and requeue with new CLASS
SPLCLFRM   EQU  X'04' Close and requeue with new FORM
SPLCLDES   EQU  X'02' Close and requeue with new DESTination
SPLCLUNC   EQU  X'01' Close and requeue nonconverted
```

Notes:

1. If the user ID is less than 8 characters, it must be left-justified and padded with blanks.
2. When you specify SPLCLPUR, it is invalid to specify any other option in SPLCFLCL.
3. It is invalid to specify both SPLCLCON and SPLCLUNC.
4. If SPLCLPUR is not specified, either SPLCLCON or SPLCLUNC must be specified.
5. When SPLCCOPY is not zero, CLOSE updates the copy count, but the file remains in SELECT status; no other changes occur.
6. SPLCCOPY cannot be greater than 255.
7. When you specify SPLCLPUR and the file has the KEEP attribute, the file is closed in USER hold. When you specify SPLCLPUR and the file has the NOKEEP attribute, the file is purged.

8. When you specify either SPLCLCON or SPLCLUNC, the file is closed according to the settings of SPLCLUHO and SPLCLSHO. If neither SPLCLUHO or SPLCLSHO is specified, the file is closed NOHOLD regardless of the KEEP attribute of the file.

REPLY Information

The Spool system service responds to the CLOSE request using an IUCV REPLY with DATA=PRMMSG, and PRMMSG containing the following information:

```
IPRMMSG1  Bytes 0 to 3      Reserved
IPRMMSG2  Byte  0           Flag
          Bytes 1 to 3      Reserved

Bit Defined in Byte 0 of IPRMMSG2
EQU  X'04'        File not selected; message already sent to the
                  prtid that selected the file.
```

Reasons the CLOSE Function May Sever

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to a *prtid*:

Return code=X'80':

- Use of TYPE=1WAY communication
- Incorrect buffer length. The specified buffer length must be 30-bytes.
- Conflicting file-disposition options. For example, specifying close and purge file at the same time you specify close and requeue in user hold.
- Use of DATA=PRMMSG in the IUCV SEND
- SPLCCOPY is greater than 255.

Return code=X'10': I/O pending

When the Spool system service CLOSE does a SEVER, the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'05' -  Identifies the type of IUCV
                   SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                   that caused the error
```

The MESSAGE Function

A *prtid* uses the Spool system service MESSAGE function to send messages or command responses to the operator or other users. To do this, the *prtid* uses an IUCV SEND specifying TYPE=2WAY, DATA=BUFFER, BUFFER=*address*, and BUFLEN=*address*, a value of 6 in the address specified by TRGCLS to the Spool system service.

When a *prtid* sends a message, it sends the following information to the Spool system service in a buffer:

```
SPLMSUID   DS   CL8      User ID to receive the message
SPLMSTYP   DS   CL1      TYPE of message:
                              S - Sent via CP SMSG
                              M - Sent via CP MSG
                              W - Sent via CP MSGNOH and
                                  console alarm is sounded
SPLMSARE   DS   0XL224   Message area (maximum size is 224 bytes)
                         - First 2 bytes:  Message area length
                         - Next 'n' bytes: Message text
SPLMSLEN   DS   XL2      Message area length
                           (actual size of SPLMSARE)
SPLMSTXT   DS   CL222    Message text (maximum size is 222.)
```

Note: When a message is sent to a remote node, the issuer is responsible for placing the necessary networking control sequence and the actual message.

To send a message, the issuer uses the following format:

```
RSCScmd    VMnode    userid    message
```

For example, if a user wants to send a message (HI from OVERHERE) to a remote VM node (RCVNODE) user ID OVERTHER, then the issuer specifies the message text be sent to the RSCS machine as follows.

```
MSG        RCVNODE   OVERTHER   HI from OVERHERE
```

REPLY Information

When the Spool system service MESSAGE finishes processing the issuer's request, the Spool system service MESSAGE replies to the issuer with return codes indicating the success or failure of the function. The reply is sent using an IUCV REPLY, DATA=PRMMSG, and PRMMSG=return code.

Return codes which do *not* sever the path are:

Code
Meaning

X'00'
Successful message

X'03'
USERID=ALL is invalid, when message type is S

X'2D'
User ID not logged on

X'39'
User ID not receiving.

Reasons the MESSAGE Function May Sever

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to a *prtid*:

Return code=X'80':

- Use of TYPE=1WAY communication
- Use of DATA=PRMMSG. The Spool system service MESSAGE allows only DATA=BUFFER.
- Target user ID is all blanks. The Spool system service MESSAGE does not allow character blanks for the target user ID.
- Invalid message type.
- Invalid data buffer length. The Spool system service allows only message types S, M, and W. The Spool system service requires that the data buffer length be greater than or equal to 12 and less than or equal to 240.
- Invalid message area format. The length of the message area must be greater than 2 but less than or equal to 224.

When the Spool system service MESSAGE does a SEVER, the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'06'—Identifies the type of IUCV
                   SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                   that caused the error
```

The READ-SFBLOK Function

With this function, a *prtid* can read the information contained in a selected spool file's SFBLOK.

When a *prtid* does a READ-SFBLOK, it sends the following information to the Spool system service:

- The spool ID of the file

- The size of the buffer
- The address of the receiving buffer
- The user ID of the owner of the file.

The READ-SFBLOK function uses the IUCV SEND specifying TYPE=2WAY, DATA=BUFFER (all the user data is in a buffer), and a value of 2 in the address specified by TRGCLS. BUFFER=*address* and a value of 16 in the address specified by BUFLEN are also specified.

The buffer contains the following information:

```
SPLRFSID   DS   1H       Spool ID
SPLRFSIZ   DS   1H       Buffer size
SPLRFADR   DS   1F       Buffer address
SPLRFUID   DS   CL8      User ID
```

Note: If the user ID is less than 8 characters, it must be left-justified and padded with blanks.

When READ-SFBLOK is complete, the Spool system service replies using an IUCV REPLY with DATA=PRMMSG, and PRMMSG containing the following information:

```
IPRMMSG1  Bytes 0 and 1     Reserved
          Bytes 2 and 3     Length of data
IPRMMSG2  Byte  0           Flag
          Bytes 1 to 3      Reserved

Bits Defined in Byte 0 of IPRMMSG2
EQU  X'20'        CP paging error
EQU  X'08'        User buffer not large enough to
                  hold requested data
EQU  X'04'        File not selected, or Spool system
                  service processing for the file stopped,
                  and a Spool system service PURGE
                  message already sent to the prtid
                  that selected the file.
```

Notes:

1. If the READ-SFBLOK was successful, then bytes 2 and 3 of IPRMMSG1 contain the actual length of the SFBLOK.
2. If the READ-SFBLOK was unsuccessful, then bytes 2 and 3 of IPRMMSG1 contain zeros.

Reasons the READ-SFBLOK Function May Sever

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to a *prtid*:

Return code=X'80':

- The use of TYPE=1WAY communication.
- The use of DATA=PRMMSG. The Spool system service READ-SFBLOK only allows DATA=BUFFER on the IUCV SEND.
- Incorrect buffer length. The message buffer length must be 16-bytes.

Return code=X'08':

- Protection or addressing violation for the user buffer. Either there is a storage protection violation involving the user buffer, or the address range specified for the buffer is not addressable.

When the Spool system service READ-SFBLOK does a SEVER, the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'02'—Identifies the type of IUCV
                   SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                   that caused the error
```

The READ-XAB Function

Use the READ-XAB function to allow a *prtid* to read the information contained in a selected spool file's external attribute buffer (XAB).

When a *prtid* does a READ-XAB, it sends the following information to the Spool system service:

- The spool ID of the file
- The size of the buffer
- The address of the receiving buffer
- The user ID of the owner of the file.

The READ-XAB function uses the IUCV SEND, specifying TYPE=2WAY, DATA=BUFFER (all the user data is in a buffer), and a value of 3 in the address specified by TRGCLS. BUFFER=*address* and a value of 16 in the address specified by BUFLEN are also specified. The buffer contains the following information:

```
SPLRXSID   DS   1H       Spool ID
SPLRXSIZ   DS   1H       Buffer size
SPLRXADR   DS   1F       Buffer address
SPLRXUID   DS   CL8      User ID
```

Note: If the user ID is less than 8 characters, it must be left-justified and padded with blanks.

REPLY Information

When the Spool system service READ-XAB is complete, the Spool system service replies using an IUCV REPLY, with DATA=PRMMSG, and PRMMSG containing the following information:

```
IPRMMSG1  Bytes 0 and 1     Reserved
          Bytes 2 and 3     Length of data
IPRMMSG2  Byte  0           Flag
          Bytes 1 to 3      Reserved

Bits Defined in Byte 0 of IPRMMSG2
EQU  X'20'        CP I/O error
EQU  X'08'        User buffer not large enough to
                  hold requested data
EQU  X'04'        File not selected or Spool system
                  service processing for the file stopped
                  and a Spool system service PURGE
                  message already sent to the prtid
                  that selected the file.
```

Note: If there is no XAB, then the size returned is zero. If the size returned is greater than the size of the buffer provided, then no data was put in the buffer. In this case, the requester must get a buffer large enough to hold the XAB and repeat the READ-XAB function.

Reasons the READ-XAB Function May Sever

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to a *prtid*:

Return code=X'80':

- The use of TYPE=1WAY communication.
- The use of DATA=PRMMSG on the IUCV SEND. The Spool system service READ XAB only allows DATA=BUFFER.
- Incorrect buffer length. The specified buffer length must be 16 bytes.

Return code=X'08':

- An invalid user buffer length. The buffer length cannot be greater than 32,767 bytes.
- A protection or addressing violation for the user buffer. There is either a storage protection violation involving the user buffer, or the address range specified for the buffer is not addressable.

When the Spool system service READ-XAB does a SEVER, the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'03'—Identifies the type of IUCV
                   SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                   that caused the error
```

The READ-SPLINK Function

You use this function to read print data from a file. With this function, a *prtid* can read the CP DASD records (SPLINKs) for a selected spool file.

When a *prtid* requests a READ-SPLINK, it sends the following information to the Spool system service:

- The spool ID of the file
- The user ID of the owner of the file
- The number of SPLINKs to be read
- The address of the receiving buffer, which must be on a 4KB boundary.

The READ-SPLINK function uses an IUCV SEND specifying TYPE=2WAY, DATA=BUFFER, BUFFER=*address*, a length of 16 in the address specified by BUFLEN, and a value of 1 in the address specified by TRGCLS. The buffer contains the following information:

```
SPLRPSID   DS   1H       Spool ID
SPLRPSIZ   DS   1H       n data pages to be read
SPLRPADR   DS   1F       Buffer address
SPLRPUID   DS   CL8      User ID
```

Notes:

1. *n* is the number of data pages to be read (each page is 4KB). SPLINKs are read in sequential order. The first READ for a SPLINK after the file has been selected, reads the first *n* SPLINKs for that file. Following READS read the next *n* SPLINKs for the file. Reads can be continued until the end-of-file is reached.
2. If the user ID is less than 8 characters, it must be left-justified and padded with blanks.

REPLY Information

When the READ-SPLINK is complete, the Spool system service replies using an IUCV REPLY with DATA=PRMMSG, and PRMMSG containing the following:

```
IPRMMSG1  Bytes 0 and 1     Reserved
          Bytes 2 and 3     Number of SPLINKs read
IPRMMSG2  Byte  0           Flag
          Bytes 1 to 3      Reserved

Bits Defined in Byte 0 of IPRMMSG2
EQU  X'20'        CP I/O error
EQU  X'04'        File not selected or Spool system
                  service processing for the file stopped
                  and a Spool system service PURGE
                  message already sent to the prtid
                  that selected the file.
EQU  X'02'        End-of-file for READ-SPLINK
```

Reasons the READ-SPLINK Function May Sever

The following situations are considered user errors and cause the Spool system service to sever the IUCV path to a *prtid*:

Return code=X'80':

- The use of TYPE=1WAY communication.
- The use of DATA=PRMMSG on the IUCV SEND. The Spool system service READ SPLINK only allows DATA=BUFFER on the IUCV SEND.
- Incorrect buffer length. The message buffer length must be 16 bytes.

Return code=X'08':

- A protection or addressing violation for the user buffer

  Any of the following has happened:

  – The address of the user buffer is not on a 4KB boundary.
  – There is a storage protection violation involving the user buffer.
  – The address range specified for the buffer is not addressable.
- The number of SPLINKs specified was zero.

Return code=X'10': I/O pending

The READ-SPLINK or READ-XAB function was done while a previous READ-SPLINK/READ-XAB request was outstanding for the file. The IUCV REPLY for the previous request must be received before the next READ can be done.

When the Spool system service READ-SPLINK does a sever (using the SEVER function), the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'01' -  Identifies the type of IUCV
                   SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                   that caused the error
```

The ENABLE Function

This function enables reader notification for a specified user ID. It sends an ENABLE (though an IUCV SEND specifying TYPE=2WAY, DATA=PRMMSG PRMMSG=user ID (to be enabled) and a value of 10 in the address specified by TRGCLS) to the Spool system service.

REPLY Information

When the Spool system service ENABLE is completed, the Spool system service replies using an IUCV REPLY, with DATA=PRMMSG, and PRMMSG containing the following information:

```
IPRMMSG2  Byte  0          X'00' function request has been
                           completed successfully
```

When the user ID does not exist, the Spool system service always responds using an IUCV REPLY with DATA=PRMMSG, and PRMMSG containing the following information:

```
IPRMMSG2  Byte  0          X'04' Userid not found
```

When the Spool system service ENABLE does a SEVER, the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'0A' -  Identifies the type of IUCV
                       SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                       that caused the error
```

The DISABLE Function

This function disables reader notification for a specified user ID. It sends a DISABLE (though an IUCV SEND specifying TYPE=2WAY communication, DATA=PRMMSG, PRMMSG=user ID (to be enabled), and a value of 11 in the address specified by TRGCLS) to the Spool system service.

REPLY Information

When the Spool system service DISABLE is completed, the Spool system service replies using an IUCV REPLY, with DATA=PRMMSG, and PRMMSG containing the following information:

```
IPRMMSG2  Byte  0          X'00' function request has been
                           completed successfully
```

When the Spool system service DISABLE does a SEVER, the format of the IPUSER field is:

```
IPUSER + 0:   Return code
IPUSER + 1:   X'0B' -  Identifies the type of IUCV
                       SEND that caused the sever
IPUSER + 2:   IPMSGID from the IUCV parameter list
                       that caused the error
```

Spool System Service Communication to a Virtual Machine

When the Spool system service accepts the connection to a *prtid*, the Spool system service can send the following types of requests to a *prtid*:

- SEND
- NOTIFY
- PURGE.

The pages that follow describe these functions.

The SEND Function

The Spool system service uses the Spool system service SEND function to pass printer type commands directly to a *prtid*.

The Spool system service SEND function is used in two ways:

1. To pass data from a non-privileged user to a *prtid* by specifically issuing the CP SEND command.
2. To route operator commands dealing with printer internals to *prtids*. The commands in this group are those existing class D, spool operator, and CP commands that control physical system printers. These commands are:
   - DRAIN
   - FLUSH
   - QUERY
   - START.

   The Spool system service routes any of the commands issued from the virtual console which specifies a *prtid* in place of an *rdev*.

The Spool system service SEND function is an IUCV SEND specifying TYPE=1WAY, DATA=BUFFER, BUFFER=*address*, a length of 253 in the address specified by BUFLEN, and a value of 9 in the address specified by TRGCLS.

Note: When the ALL parameter is specified on the DRAIN, QUERY, or START command, the Spool system service SEND function sends the text to every CONNECTed *prtid*. Also, when *prtids* have the same name, the Spool system service SEND function is done to each of them independently.

It is the responsibility of the print server to do the necessary privilege class and authorization checks for all commands sent to it using the Spool system service SEND function. The *prtid* uses the Spool system service MESSAGE function to respond to the issuer of the command.

With Spool system service SEND, the Spool system service passes the following information to a *prtid*:

```
SPLSNUID   DS   CL8      User ID of the issuer of the command
SPLSNPCS   DS   XL4      Privilege classes of the issuer
SPLSNTXT   DS   CL240    Text of the command
           DC   X'15'    End-of-text indicator
```

The end-of-text indicator can occur anywhere within the SPLSNTXT field or in the byte following this field.

The NOTIFY Function

The Spool system service NOTIFY function informs an idle *prtid* that a print file is available for processing. It also informs the reader service machine that a reader file has been added or deleted to the specified user's queue.

The Spool system service notifies the *prtid* through an IUCV SEND specifying TYPE=1WAY, DATA=PRMMSG, and a value of 8 in the address specified by TRGCLS. No information is sent to the *prtid* concerning the spool files associated with the NOTIFY.

Notes:

1. A *prtid* is considered to be *idle* when it has done a SELECT and received a response of *file not available* from the Spool system service.
2. When a *prtid* obtains a file for processing by issuing a SELECT request, the Spool system service stops sending notifications to the *prtid* because the *prtid* is no longer idle.

The Spool system service notifies the reader service machine through an IUCV SEND specifying TYPE=1WAY, DATA=BUFFER, a length of 9 in the address specified by BUFLEN, and a value of 8 in the address specified by TRGCLS. The 9-bytes of data in the buffer consist of the user ID whose reader queue changed, the user's spool ID of the reader file, and the number of reader files currently on the user's queue.

The PURGE Function

The Spool system service PURGE function signals a *prtid* to immediately stop processing a print file that the *prtid* has selected and is in the process of converting. The Spool system service PURGE is done when a CP PURGE, CP CHANGE, or CP TRANSFER command is issued for a file being converted.

For the PURGE function, the Spool system service does an IUCV SEND specifying (TYPE=1WAY, DATA=BUFFER, a length of 10 in the address specified by BUFLEN, and a value of 7 in the address specified by TRGCLS) to the *prtid*. The 10-bytes of data in the buffer consist of the spool ID of the file to be *purged* and the user ID that owns the file. The buffer is formatted as follows:

```
+ 0:   DS  XL2  Spool ID
+ 2:   DS  CL8  User ID
```

Note: If the user ID is less than 8 characters, it is left-justified and padded with blanks.

The Generic *SPL Interface

The *SPL also functions as a generic spool file interface facility.

Establishing Communication with the Spool System Service

The spool system service uses IUCV to communicate with an application. To create a path to be used in conjunction with the generic *SPL interface, the application must issue an IUCV CONNECT with *userid* set to *SPL, PRTY=NO, PRMDATA=YES, and USERDTA is an address containing the word *VIRTUAL* followed by a blank in the first 8 bytes.

Processing a File

Each generic *SPL interface path created between the application and the spool system service can be used to read data from only one file at a time. The process of reading the contents of a file can be summarized as follows:

1. IUCV CONNECT to *SPL with the first 8 bytes of IPUSER set to VIRTUAL (followed by a blank).
2. IUCV SEND a SELECT request to *SPL specifying a file to be selected for reading.

   Note: You can have the SFBLOK, 3800 information, any number of SPLINKs and the XAB data of the selected file to be transmitted back to you using the SELECT request.

The data becomes available in the specified buffers when the message complete interrupt occurs.

3. IUCV SEND a GET request for SFBLOK, 3800 information, SPLINKs, XAB, or any combination thereof.

   The data becomes available in the specified buffers when the message complete interrupt occurs.

4. IUCV SEND a CLOSE request for the file.

At this point the IUCV path may be reused or SEVERed. To reuse, simply send another SELECT request to *SPL informing it of the next file to be selected.

Selecting a File To Be Read

To select a file to be made active on an established generic *SPL connection, you use IUCV SEND to send a SELECT request to the *SPL System Service.

A SELECT request involves issuing an IUCV SEND request TYPE=2WAY, DATA=BUFFER, BUFFER = address of 48 byte parameter list (mapped by SPGBK), BUFLEN = address of field containing length of the parameter list (SPGSIZEB), a value of 1 in the address specified by TRGCLS, ANSBUF = address of 16 byte reply buffer (mapped by SPRBK), ANSLEN = address of field containing length of the reply buffer (SPRSIZEB). The parameter list sent to *SPL for a SELECT request is mapped by SPGBK. The reply buffer is mapped by SPRBK. SPGBK DSECT and SPRBK DSECT are both in the HCPGPI MACLIB.

A SELECT request is required to activate a file on an IUCV path. SELECTs fall into two possible categories:

- Select Specific - selecting by user ID, spool ID and spool queue
- Select Next - select next unseen file by user ID and spool queue

Thus, the SPGSPEC and SPGNEXT bit settings in the SPGOPTN byte are mutually exclusive. Once a file is selected on a given *SPL connection no reference needs to be made to the spoolid of the file; all transactions that follow refer specifically to the "attached" file. To request files from spool queues belonging to a user other than that of the virtual machine the application is running on, the virtual machine running the application must have class D privileges.

```
SPGBK    DSECT

SPGUSER  DS    CL8           Userid whose file is to be selected
SPGSPID  DS    H             Spoolid of file to be selected
SPGQUEUE DS    X             Queue (RDR|PRT|PUNCH) file is on
*
*        Settings for SPGQUEUE
*
SPGRDR   EQU   1             Select file from reader queue
SPGPRT   EQU   2             Select file from printer queue
SPGPUN   EQU   3             Select file from punch queue

SPGOPTN  DS    X             flag byte
*
*        Bits defined in SPGOPTN for SELECT and GET operations
*
SPGSPEC  EQU   X'80'         Select specific (SPGUSER, SPGSPID
                               and SPGQUEUE)
SPGNEXT  EQU   X'40'         Select next unseen (SPGUSER & SPGQUEUE)
SPGRESET EQU   X'20'         Reset all seen bits (SPGUSER &SPGQUEUE)
SPGGSFB  EQU   X'10'         Transfer SFBLOK information
SPGG3800 EQU   X'08'         Transfer 3800 section information
SPGGSPLK EQU   X'04'         Transfer SPLINK(s) (SPGSPLKS & SPGSPLKN)
SPGGXAB  EQU   X'02'         Transfer XAB
SPGNOSEL EQU   X'01'         Request for data only,
                               file remains unselected
*
*        Bits defined in SPGOPTN for CLOSE operation
*
SPGPURGE EQU   X'80'         Purge the file
SPGUHOLD EQU   X'40'         Put the file in user hold
SPGSHOLD EQU   X'20'         Put the file in system hold
SPGDECR  EQU   X'10'         Decrement the copy count

         DS    F             Reserved for future IBM use

SPGSPLKS DS    F             Start transferring SPLINK(s) from here
SPGSPLKN DS    F             Number of SPLINKs to transfer

SPGSFBA  DS    A             Address to place SFBLOK info
SPG3800A DS    A             Address to place 3800 info (SPLINK)
SPGSPLKA DS    A             Address to place SPLINK info
SPGXABA  DS    A             Address to place XAB info

SPGSFBL  DS    X             Len of SFBLOK info requested (d-words)
SPG3800L DS    X             Len of 3800 info requested (bytes)
SPGXABL  DS    H             Len of XAB requested (bytes)
         DS    F             Reserved for future IBM Use

SPGSIZEB EQU   (*-SPGBK)     Size in bytes
SPGSIZED EQU   (*-SPGBK+7)/8 Size in d-words
```

Figure 88. SPGBK DSECT

The *SPL System Service will respond with 16 bytes of information at the address specified on the ANSBUF parameter of the IUCV SEND. The buffer will contain the following information:

```
SPRBK    DSECT

SPRGENRC DS    X             General return code
SPRSFBRC DS    X             SFBLOK related return code
SPRSPKRC DS    X             SPLINK related return code
SPRXABRC DS    X             XAB related return code
SPR38RC  DS    X             3800 related return code

SPRSPLKN DS    AL3           Number of SPLINKs transferred

SPRSFBL  DS    X             Len of SFBLOK info returned (d-words)
SPR3800L DS    X             Len of 3800 info returned (bytes)
SPRXABL  DS    H             Len of XAB data returned (bytes)

         DS    F             Reserved for future IBM Use

SPRSIZEB EQU   (*-SPRBK)     Size in bytes
SPRSIZED EQU   (*-SPRBK+7)/8 Size in d-words
```

Figure 89. SPRBK DSECT

In the case of a SELECT that does not request any additional information to be transferred, the only field set in the SPRBK is SPRGENRC:

Code
Meaning

0
Request completed successfully

4
No file meeting requested criteria found

8
Previously selected file still active

12
Not authorized to select another user's files

16
Value specified for queue (SPGQUEUE) is not valid.

20
Specfied userid (SPGUSER) not found.

24
ESM check failed.

255
Invalid parameter list encountered:

Both Select Specific (SPGSPEC) and Select Next (SPGNEXT) were specified
Neither Select Specific (SPGSPEC) or Select Next (SPGNEXT) were specified and there was not already an active file on the path

The SELECT request may be combined with a GET request to select a file and get information such as SFBLOK, 3800 information (in SPLINK), SPLINK(s) and XAB information as part of the initial reply. This is accomplished by setting transfer options bits in the SPGOPTN option byte. See the following section for details of the GET request.

Note: The SPGRESET option bit is used to indicate to *SPL that all files in the user's selected queue are to have their seen bits reset. This action will supersede any other selection request. Thus, if SPGRESET and SPGNEXT are set, all seen bits will be reset and the application will end up attaching the first file in the selected queue (based on SPGQUEUE) to the *SPL IUCV connection.

Transferring Information About a Selected File

As mentioned before, the GET request may be combined with the SELECT request described in the previous section. As with SELECT requests, GET requests are made by sending the *SPL System service a SPGBK parameter list using TYPE=2WAY and a value of 1 in the address specified by TRGCLS. Combined SELECT and GET requests are recognized through the bit settings in the SPGOPTN byte. A pure GET request will fail if no file is currently active on the chosen *SPL path.

With a GET request or a combined SELECT and GET (assuming the SELECT request does not fail), the application can expect additional data to accompany the REPLY from the *SPL System Service. The GET request is formulated as follows:

- If the SPGGSFB bit is set, the first SPGSFBL double words of the SFBLOK of the attached file is placed at the address specified in SPGSFBA. The minimum length that can be specified by SPGSFBL is 24 d-words.
- If the SPGG3800 bit is set, the first SPG3800L bytes of the 3800 section of the attached file is placed at the address specified in SPG3800A.
- If the SPGGSPLK bit is set, the number of SPLINKs specified in SPGSPLKN is placed at the address specified in SPGSPLKA.

The SPGSPLKS and SPGSPLKN settings determine what SPLINKs of the file are placed at the specified address. The value of SPGSPLKS determines which SPLINK the transfer will start from. The value of SPGSPLKN determines how many SPLINKs will be transferred with this request.

For example, a value of 2 in SPGPLKS and a value of 3 in SPGSPLKN would indicate that the transfer of SPLINK data should start with the second SPLINK in the file and that a total of three SPLINKS would be placed into storage starting at the address specified in SPGSPLKA.

If the value of SPGPLKS is 0, the read will start from wherever the file is currently positioned. Multiple SPLINKs are copied into contiguous storage starting at the address specified in SPGSPLKA.

All write CCWs in the SPLINKs transferred by *SPL remain as the original opcode. This is different from Diagnose X'14', which converts all write CCWs in punch files to X'41' opcodes.

- If the SPGGXAB bit is set, the first SPGXABL bytes of the XAB of the attached file are placed at the address specified in SPGXABA.

If the settings of the SPGSFBL, SPG3800L, SPGSPLKN or SPGXABL fields exceed the length of the corresponding areas for the file, only the data available is transferred to the buffer. The remainder of the buffer is unchanged. If the settings of the SPG3800L, SPGSPLKN or SPGXABL fields are less than amount of data available, then the amount of data specified in the length field is transferred. If SFBLOK data is being requested, SPGSFBL must be set to a minimum of 24 double words. The SPRSPLKN, SPRSFBL, SPR3800L or SPRXABL fields will contain actual number of transferred splinks, double words or bytes. If the amount of data transferred is not equal to the amount requested, the SPRSFBRC, SPRSPKRC, SPRXABRC or SPR38RC fields will contain a return code to indicate this. If SPGSFBL, SPG3800L, SPGSPLKL or SPGXABL are equal to 0, then the corresponding return code in the SPRBK (SPRSFBRC, SPRSPKRC, SPRXABRC or SPR38RC) will be 0 and the corresponding length field in the SPRBK (SPRSPLKN, SPRSFBL, SPR3800L or SPRXABL) will be 0.

The GET request may cause the following return codes to be reflected in the SPRBK. For a combined SELECT and GET request, these return codes can be set in addition to those already outlined in the SELECT section above.

The settings for SPRGENRC can be the following:

Code
Meaning

0
Request completed successfully.

255
Invalid parameter list encountered:

SPGNOSEL was specified with Transfer SPLINK (SPGGSPLK) and/or Transfer XAB (SPGGXAB)

The settings for SPRSFBRC can be the following:

Code
Meaning

0
Requested SFBLOK information transferred successfully.

4
Paging error encountered when attempting to transfer SFBLOK information to your virtual machine.

8
Addressing or protection exception encountered due to invalid SFBLOK data area address specified on request.

12
Data area address length that was specified was smaller than size of the SFBLOK (192 bytes).

16
The amount of data requested was greater than the amount of data available. SPRSFBL contains the number of double words of data returned.

The settings for SPRSPKRC can be the following:

Code
Meaning

0
Requested SPLINK information transferred successfully.

4
Paging error encountered attempting to access SPLINKs or transfer SPLINK information to your virtual machine.

8
Addressing or protection exception encountered due to invalid SPLINK address specified on request.

12
Starting SPLINK number specified in SPGSPLKS exceeded the number of SPLINKs contained in the file.

16
The amount of data requested is not equal to the amount available. SPRSPLKN contains the actual number of SPLINKs returned. If SPRSPLKN is less than the number of SPLINKs requested, end of file was reached.

20
There was already an outstanding read for this file

The settings for SPRXABRC can be the following:

Code
Meaning

0
Requested XAB information transferred successfully.

4
Paging error encountered when attempting to access XAB or transfer XAB information to your virtual machine.

8
Addressing or protection exception encountered due to invalid XAB data area address specified on request or length of data requested (SPGXABL) was greater than 32,767 bytes.

12
File does not have an XAB defined.

16
The amount of data requested is not equal to the amount available. SPRXABL contains the actual number of bytes of XAB data returned. If SPRXABL is less than the amount of XAB data requested (SPGXABL), then all of the available XAB data was returned. If SPRXABL is equal to the amount of XAB data requested, there is more data available.

20
There was already an outstanding read for this file

The settings for SPR38RC can be the following:

Code
Meaning

0
Requested 3800 information transferred successfully.

4
Paging error encountered when attempting to access 3800 data or transfer 3800 information to your virtual machine.

8
Addressing or protection exception encountered due to invalid 3800 data area address specified on request.

16
The amount of data requested is not equal to the amount available. SPR3800L contains the actual number of bytes of 3800 data returned. If SPR3800L is less than the amount of 3800 data requested (SPG3800L), then all of the available 3800 data was returned. If SPR3800L is equal to the amount of 3800 data requested, there is more data available.

If more than one of the transfer bits is set in the GET request, the operations are handled by *SPL independently. For example, if an invalid SPLINK start counter is specified in the SPGSPLKS field, no SPLINK information will be transferred to the user's virtual machine, but any other information requested will be placed at the requested locations in storage.

Closing a File

Once all needed information for a given file has been obtained, the file can be closed by issuing a CLOSE request to the *SPL System Service.

A CLOSE request involves issuing an IUCV SEND request TYPE=2WAY, DATA=BUFFER, BUFFER = address of 48 byte parameter list (mapped by SPGBK), BUFLEN = address of field containing length of the parameter list (SPGSIZEB), a value of 2 in the address specified by TRGCLS, ANSBUF = address of 16 byte reply buffer (mapped by SPRBK), ANSLEN = address of field containing length of the reply buffer (SPRSIZEB). The parameter list sent to *SPL for a CLOSE request is mapped by SPGBK. Required fields are SPGUSER and SPGSPID. SPGOPTN can be used to specify whether the file should be closed or the copy count decremented. SPGOPTN is defined as follows for the CLOSE function:

X'80'
Purge the file

X'40'
Place file in user hold

X'20'
Place file in system hold

X'10'
Decrement the copy count of file. This option does not close the file. If the copy count is already set to 1, purge the file.

The purge option cannot be used in conjunction with the other three options. If the purge option is specified with any of the other three options, the path is severed. No other restrictions exist. If none of the bits are set, the file is closed and its characteristics are not changed.

The *SPL System Service replies to the CLOSE request by sending an SPRBK back in the address specified by the ANSBUF parameter on the IUCV SEND.

The possible settings for SPRGENRC are as follows:

Code
Meaning

0
File closed and options set as requested

4
No file active on this *SPL connection

8
File purged (either because of a purge request or a decremented copy count)

Once the file has been closed, the path to *SPL need not be severed; instead the path may be reused by a new SELECT request attaching a new file to the path.

Clearing an Existing Connection

Should the application lose track of what file is active on a path or should it wish to clear out any file attached to the path, it can issue a CLEAR request for the *SPL path.

A CLEAR request involves issuing an IUCV SEND request TYPE=2WAY, DATA=BUFFER, BUFLEN = address of field containing a value of 0, a value of 3 in the address specified by TRGCLS, ANSBUF = address of 16 byte reply buffer (mapped by SPRBK), ANSLEN = address of field containing length of the reply buffer (16).

The *SPL System Service replies to a CLEAR request by returning an SPRBK in the address specified on the ANSBUF parameter of the IUCV SEND. The only significant field in the SPRBK is the SPRGENRC field, which may be set as follows:

Code
Meaning

0
Active file on path has been closed

4
No file was active on the path

Chapter 22. Symptom System Service (*SYMPTOM)

An application may be written by an installation to run in a guest virtual machine that has been authorized to use the IUCV interface to collect symptom records from the z/VM control program supporting it. This IUCV authorization is defined within the IUCV Directory Control Statement of the guest virtual machine. The IUCV control statement must name *SYMPTOM as the CP system service to which a communication path will be established. The user ID of the guest virtual machine may also be identified to the control program during system generation or in the system configuration file so that records can be accumulated for the virtual machine before it has connected to the system service. For more information on the required IUCV authorization and identifying the virtual machine that contains a data retrieval application to the control program (the SYSTEM_USERIDS statement in the system configuration file), see *z/VM: CP Planning and Administration*.

For more information on the IUCV functions mentioned in this chapter, see Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

The Symptom system service (*SYMPTOM) in CP supports both 1-way and 2-way IUCV protocols when sending records to authorized virtual machines. When a **1-way** IUCV SEND is issued by z/VM, the virtual machine to which the symptom record is sent cannot issue an IUCV REPLY but must issue a RECEIVE. When a **2-way** IUCV SEND is issued by z/VM, the virtual machine to which the symptom record is sent must issue an IUCV REPLY. Response data may not be sent on an IUCV REPLY. The reply buffer length field in the IUCV parameter list, IPBFLN2F, must contain zeros. This can be accomplished by setting a register to zero and coding ANSLEN=(*reg*) on the IUCV REPLY macro.

Connecting to the Symptom System Service

Before issuing the IUCV CONNECT to the CP *SYMPTOM system service, a virtual machine must issue a DECLARE BUFFER request to IUCV to provide an external interrupt buffer. The virtual machine must be enabled for IUCV interrupts in Control Register 0, and the PSW must be set to enable external interrupts.

The connection with the Symptom system service is established by issuing an IUCV CONNECT, specifying USERID=*SYMPTOM. The use of the IUCV 1-way or 2-way protocol for gathering symptom records from the CP Symptom system service is specified by the virtual machine in the IPUSER data area when it issues an IUCV CONNECT to the Symptom System Service. This area must contain a X'02' at offset 8 to indicate 2-way protocol. Otherwise, the 1-way protocol is used by default. The CONNECT parameter list must also indicate that you do not want to receive messages with data in the parameter list. This is indicated by specifying or defaulting to the PRMDATA=NO option on the IUCV CONNECT.

A virtual machine is not allowed to issue an IUCV SEND to the Symptom system service (the path is QUIESCED by the CP Symptom system service). A virtual machine may only have one communication path to the Symptom system service. The Symptom system service only sends records with the PRTY=NO option.

When an application running in an authorized virtual machine issues a CONNECT to the CP Symptom system service, the connection is either completed successfully (by ACCEPT) or rejected (by SEVER). If the connection is accepted, IUCV returns a PATHID to the virtual machine, which must be specified on all subsequent IUCV requests to the system service. Only one CONNECT can be issued by a virtual machine to the Symptom system service.

If the connection is severed, the Symptom system service places a 1-byte code at offset 9 of the IPUSER field of the IPARML to indicate why. A code of X'04' indicates that the virtual machine already has a connection to the Symptom system service. A code of X'08' indicates that the virtual machine made a protocol error on the CONNECT request. The PRMDATA=YES option was specified but should not have been. A code of X'0C' indicates that the maximum number of connections to the Symptom system service has been exceeded.

© Copyright IBM Corp. 1991, 2020

The data format of a symptom record is identical to records recorded using the RETRIEVE SYMPTOM command. More than one user ID may be authorized to connect to this service.

Receiving Symptom Records

To obtain a symptom record, when the application is notified by an external interrupt that one is available, an IUCV RECEIVE must be issued. The Symptom system service does not send another record until either a response (when the application indicates that data is to be sent to it using the 1-way protocol) or a REPLY (when an application indicates that data is to be sent to it using the 2-way protocol) is received by the CP Symptom system service to the previous record sent.

The CP Symptom system service maintains a threshold limit which indicates when to notify the system operator and the receiving virtual machine that uncollected records are accumulating in host storage. The default value is 2 for symptom records. The system operator may change this value by issuing the RECORDING SYMPTOM command.

To stop receiving records temporarily, the application must issue an IUCV SEVER. CP continues to queue records for your virtual machine until a CP RECORDING SYMPTOM OFF command is issued for your user ID. To resume receiving records, the system operator may issue an IUCV CONNECT specifying USERID as *SYMPTOM.

If z/VM abends while a virtual machine is collecting symptom record data, symptom records not received by the virtual machine are checkpointed and requeued to the virtual machine on a subsequent warm or force start of the control program.

The virtual machine is logged on the system automatically by CP if it is identified on the SYSTEM_USERIDS statement in the system configuration file as part of your system generation.

Disconnecting from the Symptom System Service

You can terminate collection of symptom records by issuing IUCV SEVER or IUCV RETRIEVE BUFFER for your *SYMPTOM path. A SEVER may be initiated by the system due to virtual machine reset or an IUCV RETRIEVE BUFFER request.

Chapter 23. VM Event System Service (*VMEVENT)

The VM Event system service (*VMEVENT) is a CP system service that notifies you about certain events that occur in the VM system. When you connect to the VM Event system service, you receive notification of all events that occur from that point forward. For a virtual machine to receive notifications about VM system events, it must first be authorized to connect to the VM Event system service, *VMEVENT. Your system administrator is the person who can authorize a virtual machine to establish a connection. To do this, the administrator must specify a special IUCV *VMEVENT statement in the virtual machine's directory entry. For more information on how a system administrator authorizes virtual machines to connect to IUCV system services, see *z/VM: CP Planning and Administration*.

For more information on the IUCV functions mentioned in this section, refer to Chapter 5, “IUCV Function Descriptions,” on page 317 and Chapter 8, “IUCV Macro Functions for Use in APPC/VM,” on page 521.

Establishing Communication with the VM Event System Service

The VM Event system service uses IUCV to communicate with a virtual machine. The IUCV macro checks the validity of the IUCV parameters and any errors are handled according to IUCV specifications. Your first step in establishing IUCV communications with the VM Event system service is to issue an IUCV DECLARE BUFFER. This function initializes the virtual machine for IUCV communication. This function also specifies a buffer where IUCV can store external interruption information.

Connecting to the VM Event System Service

After you initialize for IUCV communication, you must issue an IUCV CONNECT and specify USERID=*VMEVENT in the IUCV CONNECT parameter list (IPARML).

Receiving Events

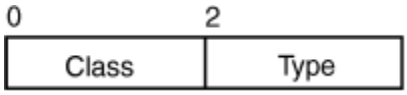
Upon notification of a successful connection, your virtual machine is ready to receive events. The VM Event system service passes events to your virtual machine one at a time by issuing a 1-way IUCV SEND. The target class (TRGCLS) identifies the class and type of the event, which indicates the nature of the associated data, as follows:

Class	Type	Data	Event
0	0	user ID	LOGON
0	1	user ID	LOGOFF
0	2	user ID, reason	Failure condition detected
0	3	user ID, interval	LOGOFF timeout started
0	4	user ID	Forced SLEEP started
0	5	user ID	Runnable state entered
0	6	user ID	Free storage limit exceeded
0	9	user ID, destination system	Virtual machine outbound relocation started
0	10	user ID, source system	Virtual machine inbound relocation started
0	11	user ID, destination system	Virtual machine outbound relocation complete
0	12	user ID, source system	Virtual machine inbound relocation complete

© Copyright IBM Corp. 1991, 2020

Class	Type	Data	Event
0	13	user ID, reason code	Virtual machine outbound relocation terminated
0	14	user ID, reason code	Virtual machine inbound relocation terminated
0	15	user ID	Timebomb exploded
0	26	user ID, operating system type	Operating system type
0	27	user ID	Virtual system reset

The target class field has the following format:



In all cases, the user ID is an eight-character value.

- For a "Failure condition detected" (Type 2) event, the user ID is followed by a one-byte reason code, which is a binary value that corresponds to the error codes that can be produced by Diagnose Code X'B0'. See “DIAGNOSE Code X'B0' – Access Re-IPL Data” on page 149.
- For a "LOGOFF timeout started" (Type 3) event, the user ID is followed by a two-byte timeout interval, which is a binary value in seconds.
- A "Runnable state entered" (Type 5) event is sent when a virtual machine becomes runnable after a previous event of some other type was produced for it.
- If either of the following events occur, a logoff timeout is started and that VM event is reported:
  - A forced disconnection occurred
  - A console read was placed on a disconnected guest that had no functional secondary user

  However, if the Disconnect_Timeout value in the system configuration file is zero, a forced sleep occurs and that VM event is reported instead. For more information about Disconnect_Timeout, see FEATURES Statement in *z/VM: CP Planning and Administration*.
- For Type 9 and Type 11, the user ID is followed by an eight-byte destination system.
- For Type 10 and Type 12, the user ID is followed by an eight-byte source system.
- For Type 13 and Type 14, the user ID is followed by an one-byte reason code that can be one of the following:

  1
  Canceled by VMRELOCATE CANCEL command

  2
  Canceled by CPHX command

  3
  Canceled due to lost ISFC connection

  4
  Canceled due to MAXTOTAL time limit exceeded

  5
  Canceled due to MAXQUIESCE time limit exceeded

  6
  Canceled due to eligibility violation

  7
  Canceled due to virtual machine action

  8
  Canceled due to an internal processing error

9
Canceled because the CP exit rejected the command

11
Canceled because the CP exit gave a return code that is not valid

- For Type 26, the user ID is followed by a 1-byte operating system type that can be one of the following:

X'00'
Unknown

X'04'
Inferred Linux

X'08'
Explicitly Linux

When you first connect to the VM Event system service, it generates events to report to you the current status of all logged-on virtual machines that have some sort of exception. The target class identifies the class and type of the status, which indicates the nature of the associated data, as follows:

Class	Type	Data	Status
1	2	user ID, reason	Failure condition exists
1	3	user ID, interval	LOGOFF timeout service
1	4	user ID	Forced SLEEP active
1	6	user ID	Free storage limit exceeded
1	9	user ID, destination system	Virtual machine outbound relocation active
1	10	user ID, source system	Virtual machine inbound relocation active

The message data correspond to those for Class 0 messages of the same type. For a "LOGOFF timeout started" (Type 3) event, the interval is the amount of time remaining in the timeout and will be zero if it has expired.

The following events are reported when the SSI mode or an SSI member state changes:

Class	Type	Data	Status
2	7	SSI name, previous mode, new mode (10 bytes)	SSI mode
2	8	SSI name, member system name, previous state, new state (18 bytes)	SSI member state

For Type 7, the SSI name is eight bytes followed by two one-byte values for mode that can be one of the following:

X'01' - Stable
X'02' - Influx
X'03' - Safe

For Type 8, the SSI name is eight bytes followed by an eight-byte system name and two one-byte values for state that can be one of the following:

X'00' - Down
X'01' - Joining
X'02' - Joined
X'03' - Leaving
X'04' - Isolated
X'05' - Suspended
X'80' - Unknown

Information describing different SSI modes and system states can be found within the QUERY SSI command documentation in *z/VM: CP Commands and Utilities Reference*.

The following events are reported when a guest on an SSI member connects to *VMEVENT:

Class	Type	Data	Status
3	7	SSI name, previous mode, current mode (10 bytes); previous mode and current mode have the same value	SSI mode
3	8	SSI name, member system name, previous state, current state (18 bytes); previous state and current state have the same value	SSI member state

Type 7 and 8 values for Class 3 are in the same format as those for Class 2.

The following networking events are reported:

Class	Type	Data	Status
4	16	Virtual Switch Name, Port Group Name, RDEV Number, Port Number, Port Status, Port Status Reason, Vswitch Status, RDEV Error Status	The device has been activated. The connection to the real hardware LAN is now operational.
4	17	Virtual Switch Name, Port Group Name, RDEV Number, Port Number, Port Status, Port Status Reason, Vswitch Status, RDEV Error Status	An additional device has been activated for the existing real hardware LAN.
4	18	Virtual Switch Name, Port Group Name, RDEV Number, Port Number, Port Status, Port Status Reason, Vswitch Status, RDEV Error Status	The device has been deactivated. The connection to the existing hardware LAN is still operational.
4	19	Virtual Switch Name, Port Group Name, RDEV Number, Port Number, Port Status, Port Status Reason, Vswitch Status, RDEV Error Status	The device has been deactivated. The connection to the existing hardware LAN is no longer operational.
4	20	Virtual Switch Name, Bridge Port RDEV Number, Bridge Port Status, Bridge Port RDEV Error Status	The device has been activated. The connection to the virtual switch Bridge Port is now operational.
4	21	Virtual Switch Name, Bridge Port RDEV Number, Bridge Port Status, Bridge Port RDEV Error Status	The device has been deactivated. The connection to the virtual switch Bridge Port is no longer operational.

Class	Type	Data	Status
4	22	Virtual Switch Name, System Identifier, Local Virtual Switch Name, Local System Identifier, MAC Address, Port Group Name, PCHID Number, Flags, Port Number	A Global virtual switch member had a connectivity problem and the local system using the same global virtual switch and shared port group took over the MAC address to continue to provide connectivity. The MAC Address assigned to Virtual Switch Name (with associated Port Group Name, PCHID Number, Port Number, and System Identifier) has been assigned to Local Virtual Switch Name on Local System Identifier.
4	23	Virtual Switch Name, System Identifier, Local Virtual Switch Name, Local System Identifier, MAC Address, Port Group Name, PCHID Number, Flags, Port Number	A connectivity problem affecting a global virtual switch and shared port group member has been repaired or a different global virtual switch member has taken over the MAC Address. The MAC Address previously assigned to Local Virtual Switch Name and Local System Identifier is now assigned to Virtual Switch Name with associated Port Group Name, PCHID Number, Port Number, and System Identifier.
4	24	Local System Identifier, IVL Domain, Flags	The System has successfully joined the IVL Domain.
4	25	Local System Identifier, IVL Domain, Flags	The System has left the IVL Domain.

In all cases, the Virtual Switch Name is an eight-character value.

- For Types 16-19:
  - Port Group Name (Link Aggregation Group name) is an eight-character value
  - RDEV Number (base device address) is a two-byte hexadecimal value
  - Port Number is a one-byte decimal value
  - Port Status is a one-byte hexadecimal value
  - Port Status Reason contains two-bytes of hexadecimal data
  - Vswitch Status (Virtual Switch UPLINK Port Status), is a one-byte hexadecimal value
  - RDEV Error Status is a one-byte hexadecimal value

  For more information on these status fields, see DIAGNOSE Code X'26C', "Subcode X'00000020'—Return Virtual Switch Information" on page 239 in Table 47 on page 240 and Table 48 on page 246.
- For Types 20-21:
  - Bridge Port RDEV number (base device address) is a two-byte hexadecimal value
  - Bridge Port Status is a one-byte hexadecimal value
  - Bridge Port RDEV Error Status is a one-byte hexadecimal value

  For more information on these status fields, see DIAGNOSE Code X'26C', "Subcode X'00000020'—Return Virtual Switch Information" on page 239 in Table 48 on page 246.
- For Types 22-23:
  - Virtual Switch Name and Local Virtual Switch Name are eight-character values
  - System Identifier and Local System Identifier are eight-character values. If the System Identifier for the remote system can not be retrieved, the field will contain zeros.
  - MAC Address is a six-byte hexadecimal value
  - Port Group Name (Link Aggregation Group name) is an eight-character value
  - PCHID Number (Physical CHannel ID) is a two-byte hexadecimal value

– Flags is a one-byte field. If Flags is X'80', this indicates that the PCHID Number is a valid value. If the Flag is X'00', the PCHID Number could not be retrieved, and is not valid.
– Port Number is a one-byte decimal value

The format of the data for types 22-23 is:

```
****     Format for events 22 and 23:
*
*
*   0 |                    Virtual Switch Name                    |
*
*   8 |                    System Identifier                      |
*
*  10 |                    Local Virtual Switch Name              |
*
*  18 |                    Local System Identifier                |
*
*  20 | /////////////  |          MAC Address                      |
*
*  28 |                    Port Group Name                        |
*
*  30 | PCHID Number | Flags | Port Number | ///////////////////////////////|
*
*
*  38
*
*
****     BITS DEFINED IN Flags (AT HEX DISPLACEMENT: 32)
*
*   VALUE DEFINITION DESCRIPTION
*   ---------- ---------------- ----------------------------------------
*    X'80'   1... ....        PCHID Number is valid
*
****
```

- For Types 24-25:
  – The Local System Identifier is an 8 character value
  – IVL Domain is a one-byte character value
  – IVL Flag is a one-byte field. If the IVL Flag is X'80', this indicates that the Local System has Joined the IVL Domain indicated. If the IVL Flag is X'40', the Local System has left the IVL Domain indicated.

  The format of the data for types 24-25 is:

```
   0 |                    Local System Identifier                |
   8 | IVL Domain | Flags | ///////////////////////////////////////////|

****     BITS DEFINED IN Flags (AT HEX DISPLACEMENT: 9)
*
*   VALUE DEFINITION DESCRIPTION
*   ---------- ---------------- ----------------------------------------
*    X'80'   1... ....        Local System Joined IVL Domain
*    X'40'   .1.. ....        Local System Left IVL Domain
*
****
```

If you want to stop receiving events from the VM Event system service temporarily, you can use IUCV QUIESCE. The VM Event system service queues events while a path is quiescent. The queued events will be available when you issue an IUCV RESUME.

Disconnecting from the VM Event System Service

When all communications with the VM Event system service are completed, you can terminate communication by issuing either an IUCV SEVER or an IUCV RETRIEVE BUFFER.

Part 5. CP Macros for VM Data Spaces and Other Services

This part contains the following chapters:

- Chapter 24, “VM Data Spaces Overview,” on page 783
- Chapter 25, “CP Macros,” on page 793

© Copyright IBM Corp. 1991, 2020

Chapter 24. VM Data Spaces Overview

Using ESA/390 extensions to the interpretive-execution facility, z/VM enables applications to take advantage of hardware support for multiple address spaces introduced by Enterprise Systems Architecture. This support is available to applications that run in ESA/Extended Configuration (XC) virtual machines.

This chapter introduces VM Data Spaces support and briefly outlines the function provided by seven CP macros available to programmers working with virtual machines. These macros, described in detail in the next chapter, are:

ADRSPACE
Provides services for creating, sharing, destroying, and managing address spaces

ALSERV
Provides services for obtaining and relinquishing access to an address space

DEFWORKA
Defines the work areas required for one or more CP macros contained within your program

MAPMDISK
Provides services for mapping data on a direct access storage device (DASD) to pages of an address space

PFAULT
Provides for asynchronous handling of page faults

REFPAGE
Provides notification to CP of expected page reference patterns.

VMUDQ
Provides an application interface for making specified queries against the CP User Directory

For information about the addition of commands and command operands, and a directory control statement and operands related to these CP macros and VM Data Spaces, see *z/VM: CP Commands and Utilities Reference* and *z/VM: CP Planning and Administration*.

What Are Data Spaces?

The guest main storage of an XC virtual machine consists of one or more extents of storage known as absolute storage address spaces, or simply address spaces. Each virtual machine has an initial extent of main storage, called its primary address space, at logon. An application running in an XC virtual machine can create multiple address spaces of up to 2 gigabytes (2GB) each for data storage in addition to its primary address space, as shown in Figure 90 on page 783. These address spaces are also called data spaces because they can be used only for data storage and manipulation. An authorized application can share its data spaces with other virtual machines on the same z/VM system. In the figure, the User1 virtual machine can share one or both of its data spaces as well as its primary address space with the User2 virtual machine.

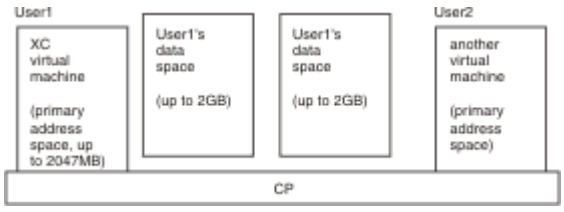


Figure 90. Created Data Spaces Being Shared With Other Users

© Copyright IBM Corp. 1991, 2020

In addition, data space support allows an application executing on a non-XC virtual machine to share its primary address space as well as copy data from other virtual machines' address spaces into its primary address space.

Uses for Data Spaces

Certain applications require vast amounts of storage to work efficiently. One such class of applications includes database managers, which can map an entire database into storage at one time rather than explicitly managing I/O. Another class of applications requires large storage buffers, which now can reside outside the virtual machine's primary address space. The graphic representation of three-dimensional objects, image processing, and numerically-intensive computation all require such large storage areas.

Applications in an XC virtual machine can use data spaces to:

- Dynamically obtain more storage than is available in its virtual machine's primary address space
- Isolate data from other programs that may execute in the same virtual machine
- Share data located in a data space among programs executing in the same or other virtual machines
- Isolate data by its particular usage and then share that data only among related users (this is an alternative to using a common area that may contain data for various uses)
- Share data located in its virtual machine primary address space with programs executing in other virtual machines.

In addition, applications (particularly those that provide database management services) can use data spaces with the minidisk mapping services. This can improve overall system performance by replacing virtual machine I/O, such as DIAGNOSE I/O, with paging I/O, which is more efficient.

ESA/XC Architecture

VM Data Space support is tailored to the ESA/XC architecture virtual machine environment. ESA/XC architecture is a derivative of ESA/390 architecture, so it shares many features of that architecture. However, ESA/XC architecture differs from ESA/390 architecture in several important ways:

- ESA/XC architecture is a virtual machine architecture designed to satisfy CMS application program requirements. Other virtual machine architectures are simulations of architectures implemented on real hardware. For example, an ESA/390 virtual machine is the functional equivalent of a real ESA/390 processor.
- ESA/XC services are available specifically for guests, such as CMS, that run with dynamic address translation (DAT) off.
- The interpretive-execution facility is not provided in ESA/XC architecture. This facility is intended for use by a host for the emulation of virtual machines. ESA/XC architecture is intended for application programs, not hosts like z/VM, so the interpretive execution facility is not needed.
- CP manages the address spaces created by XC virtual machines. As a result, address spaces created from one virtual machine can be shared with other virtual machines on the same z/VM system. In contrast, ESA/390 architecture requires that the virtual machine operating system run with DAT on and manage its own data spaces, thus precluding the virtual machine from sharing its data spaces with other users.

Other features of ESA/390 architecture, such as channel subsystem I/O and bimodal addressing, continue to exist in ESA/XC architecture. See *z/VM: ESA/XC Principles of Operation* for details of ESA/XC architecture.

Address Space Support

As the previous section indicates, ESA/XC architecture support for multiple address spaces differs from that of ESA/390 architecture and z/Architecture. The capabilities available to applications running in an XC virtual machine, however, are similar to those provided by ESA/390 architecture and z/Architecture. The combination of ESA/XC architecture and the services provided by CP macros enable applications

running in an XC virtual machine to own multiple address spaces. In addition, ESA/XC architecture allows sharing of address spaces among multiple virtual machines.

Every XC virtual machine owns at least one address space, the primary address space, given to the user by CP when the user logs on to the z/VM system. The size of this primary address space is determined from the entry describing that user ID in the user directory, or from a subsequent `DEFINE STORAGE` command.

In addition to the primary address space, a user whose CP directory contains authorization can create multiple data spaces and share them with other users. Both creating and sharing require separate authorization in the user's directory. A data space exists until it is either explicitly destroyed by the owner or until the owning virtual machine goes through a virtual machine reset operation. Virtual machine reset occurs as part of the processing of the CP LOGOFF, SYSTEM RESET, and IPL commands.

Address spaces created by programs running in an XC virtual machine are maintained by CP. The structures used to control access to address spaces thus reside in CP-controlled storage. CP also maintains a host access list for each virtual machine logged on to the z/VM system. This list contains information that designates the address spaces the associated virtual machine is allowed to access. By default, each access list can hold 62 entries, and thus can identify 62 address spaces accessible to the virtual machine in addition to the primary address space. The XCONFIG ACCESSLIST directory control statement can be used to increase the number of entries allowed on the access list.

In addition to the access lists, CP maintains a list of permitted users that is associated with each address space. To access or manipulate data in an address or data space, an application must own (or have been permitted access to) the address space.

The ability to own multiple address spaces brings to mind the question of how a program specifies which address space should be referenced by an instruction. The answer lies in ESA/390 hardware support in the form of a set of access registers. Each access register is paired with a general register. The access registers indicate which address space contains the data being referenced by an address in the associated general register.

The address translation mode determines whether the access registers are used when resolving an address. When in access-register mode (AR mode), assembler instructions (such as load, store, and move character) can move data in and out of a data space and manipulate data within it. Assembler instructions can also perform arithmetic operations on the data.

The access registers are used for addressing only when a program running on an XC virtual machine is executing in AR mode. The usual address translation mode is called primary-space mode. The non-privileged instruction called SET ADDRESS SPACE CONTROL (SAC) is used by an application program to set a bit in the program status word (PSW) that indicates which address translation mode is in effect.

Summary of Data Space Operations

To exploit data space support, use the CP macros described in detail in Chapter 25, “CP Macros,” on page 793. These macros provide system services for creating, controlling, and deleting data spaces.

An application running in a virtual machine that is allowed to **own** a data space can use the following CP macros to perform various address space functions:

ADRSPACE CREATE
Create a data space

ADRSPACE DESTROY
Destroy a data space

ADRSPACE PERMIT
Permit other users (virtual machines) or applications to access an address space

ADRSPACE ISOLATE
Isolate an address space from other users

ADRSPACE QUERY
Ask for the identification token and size of any address space it owns or is permitted to access

ALSERV ADD
Add an entry to the host access list

ALSERV REMOVE
Remove an entry from the host access list

A data space can be created and deleted only from an XC virtual machine that has been authorized using an XCONFIG ADDRSPACE directory statement. With the SHARE option on that directory statement, your application can share address spaces it created with applications running on other virtual machines.

An application running in a non-XC virtual machine that is allowed to access an address space belonging to another virtual machine can use the following operations to perform certain address space functions:

ADRSPACE QUERY	Ask for the identification token of the address space it needs to access
ALSERV ADD	Add an entry to the host access list
ALSERV REMOVE	Remove an entry from the host access list
DIAGNOSE code X'248'	Copy from another address space into primary address space (for non-XC virtual machines)

Note: Although data space management also can be performed using callable services library (CSL) routines in CMS, do not mix that programming interface with this one. CMS relies on the CP interface and any mixing of the interfaces may inhibit CMS from performing its error checking. Thus, performing data space management directly using data space CP macros or DIAGNOSE codes along with the CMS interface may cause unexpected results.

Using Data Spaces in Your Applications

This section describes how to use the CP macros in your applications to set up and manage data spaces. Before you assemble a program that uses these macros, however, you must issue the GLOBAL command for HCPGPI MACLIB.

Creating and Using Data Spaces

A program's ability to create, delete, and access data spaces depends on whether the virtual machine that it executes in has been authorized to do so through CP directory control statements. Because the use of data spaces consumes system resources, such as real and auxiliary storage, their use is controlled by the installation. System programmers responsible for tuning and maintaining z/VM use the following directory control statements to control these resources:

- XCONFIG ADDRSPACE directory control statement — This statement authorizes an XC virtual machine to create and delete data spaces, specifies the maximum size of the data spaces, and indicates whether they can be shared. For non-XC virtual machines, it allows sharing of the primary address space.
- XCONFIG ACCESSLIST directory control statement — This statement allows any type of virtual machine to access more than 62 address spaces (the number allowed without an XCONFIG ACCESSLIST statement), specifying the number of data spaces that can be accessed at any given time. The access list is maintained for the virtual machine by CP and is used to keep track of address space addressability. Note that a non-XC virtual machine can reference data in another virtual machine's address space only by using the DIAGNOSE code X'248' (copy-to-primary service).

As an application developer, you should be aware of these installation-established limits and how they relate to return codes associated with the ADRSPACE and ALSERV macros.

Creating a Data Space

When a program executing in an XC virtual machine uses the ADRSPACE CREATE macro to create a data space, it needs to provide the name and size (number of pages) of the data space.

Upon return from ADRSPACE CREATE, the address provided on the ASIT operand contains the ***address space identification token*** (ASIT) of the newly-created data space. This ASIT is a system-wide unique identifier for this data space. It is not used again during the current system IPL. The application must retain the ASIT for subsequent address space related service calls. An application can obtain the ASIT of an existing data (or address) space by calling the QUERY function with the name and owner of the address space.

An ASIT identifies an **instance** of a data space. An instance of a data space is a temporal concept that means a particular *version* of the data in a data space. For example, when a data space is deleted and a new data space with the same name is created, the new data space can be considered a new instance of the original data space. Even though the name is the same, re-creating a data space results in a new ASIT being assigned. Thus, the ASIT can be used to ensure that only users authorized for a particular instance of a data space can access it.

Permitting Another User to Access an Address Space

Recall that in addition to access lists, CP maintains a list of permitted users for each address space on the z/VM system. To access an address space that it does not own, a user must have been granted access by the owner and thus be listed on the permitted-users list for that address space by CP. The owner of an address space can grant either read-only or read/write access to one or more other users.

To grant another user access to an address space, call the ADRSPACE PERMIT macro, supplying the ASIT of the address space, the identity of the virtual machine that is to be given access, and the type of permission (read-only or read/write). Although the PERMIT function grants the access, it does not establish it. The user authorized to access the address space must then call the ALSERV ADD macro to establish the access. Figure 91 on page 787 shows that User1 has created a data space, DataSpaceA, and permitted User2 read-only access to it.

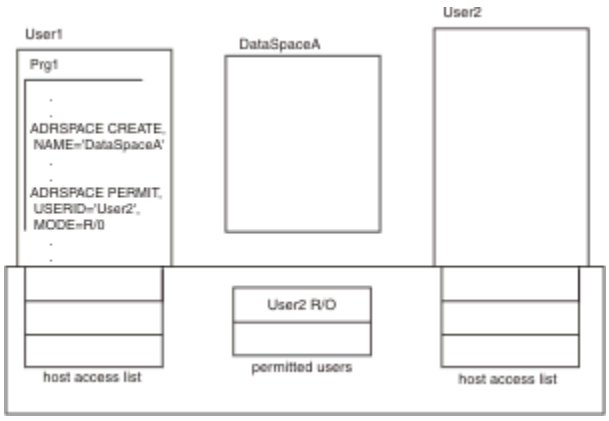


Figure 91. Granting Another User Access to an Address Space

The permitted user needs the ASIT of the address space to establish addressability. Typically, the owner informs the permitted user through standard z/VM communications facilities, such as APPC/VM. The owner of the address space always has read/write permission. It is an error for an owner to call ADRSPACE PERMIT to permit itself.

Permission to access an address space remains in effect until the owner either isolates or destroys the address space, or until a virtual machine reset occurs on either the owning or the permitted virtual machine.

Accessing Data Space Storage

To gain access to the data space, the program calls the ALSERV ADD macro, specifying the ASIT of the data space, which was returned on the ADRSPACE CREATE or QUERY call.

In the example of Figure 91 on page 787, both User1 and User2 must call ALSERV ADD to establish addressability to DataSpaceA. User1 can establish its access as read/write, because as the owner, it automatically has read/write access, while User2 can establish read-only access. ALSERV ADD fills in an unused entry in the access list as requested by the program and returns an ***access-list-entry token***

(***ALET***). The ALET is then used by the program to select the access list entry (ALE) that identifies the newly-created address space.

An ALE, as depicted graphically in Figure 92 on page 788, can be thought of as comprising several components:

A state

An ALE is always in one of three states: **unused**, **valid**, or **revoked**. An unused ALE is one that either has never been used or has become available as a result of the REMOVE function. A valid ALE is one that has been filled in by the ADD function. A valid ALE enters the revoked state as a result of the address space owner's destroying the address space or explicitly isolating it from other users. The revoked state is different from the unused state in that an ALE can enter the revoked state without any action by the owner of the access list.

An ALET

The access list entry token that selects the ALE when it is in a valid or revoked state.

An ASIT

The address space identification token designates the address space represented by the ALE.

A read/write authorization

When a program calls the ALSERV ADD macro, it stipulates whether accesses using the resulting ALET are to be read-only, or whether they can be read/write. The program can call ALSERV ADD more than once with the same ASIT to obtain different ALETs for a particular address space, for instance, one for read-only and one for read/write access.

A fault resolution flag

To use storage efficiently, CP might temporarily page out data from an address space to auxiliary storage. When an application program references a paged-out portion of an address space, CP receives notification of this as a page fault. CP then reads the data back into storage and reruns the virtual machine. Aside from a slight time delay, the virtual machine is unaware of this synchronous activity. If this slight delay in execution is a problem for an application, the program can specify that CP resolve page faults asynchronously. A program that specifies asynchronous page-fault resolution is not delayed by page faults, but must contain additional logic to complete the *handshaking* required to use this function. See the section entitled “PFAULT Macro -- Page-Fault Handshaking Services” on page 852 for more information.

Here is a graphic representation of an access-list entry:

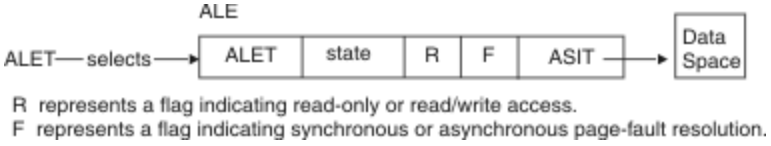


Figure 92. Graphic Representation of an ALET and an ALE

Once the application has the ALET, it can access an address space by placing the ALET in the access register associated with the general register being used for the storage operand of the instruction. Before executing instructions that use access register references, the application must enter AR mode (available only to XC virtual machines). When in AR mode, the access register corresponding to the general register specified in the base register field of the instruction is examined to determine which address space is being referenced.

The ALET in the access register indicates the address space containing the operand and the address in the general register (possibly modified by a displacement or index specified in the instruction) indicates the location of the operand within the address space, as shown in Figure 93 on page 789. For instructions such as MOVE (MVC) that have multiple storage operands, each operand can be designated by a different access-and-general register pair, so the operands can reside in different address spaces.

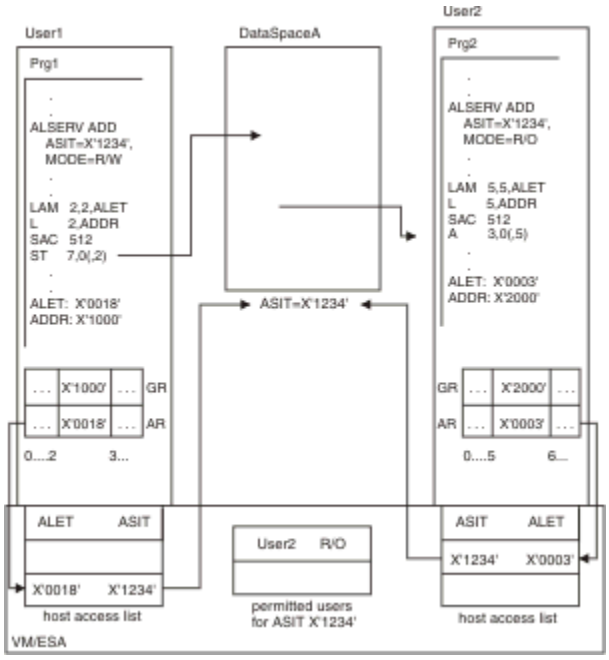


Figure 93. Accessing an Address Space

An access register should contain either zeros or an ALET that was obtained by the application when adding an ALE to its access list. If the access register contains zeros, the reference is to the primary address space. If the access register contains an ALET selecting a valid access-list entry, the reference is to the address space designated by that entry. If the selected ALE is in the unused or revoked state, a program interruption results.

Dropping Addressability to an Address Space

When access to an address space is no longer required, a virtual machine can call the ALSERV REMOVE macro to drop addressability to it. This function changes a valid (or revoked) access-list entry's state to unused so the entry can be used again later. Removing addressability does not change the virtual machine's permission to access the address space. As long as the owning virtual machine does not revoke permission, the permitted user can establish addressability again.

Maintaining addressability to an address space longer than necessary is not generally harmful, so most applications would need to remove addressability only during termination. Long-running applications, such as resource managers in server machines or very complex applications requiring access to more address spaces than the number of entries available, may need to actively manage the contents of the access list. Such applications could temporarily remove addressability to one address space to free up an access-list entry for another address space.

Isolating a Shared Address Space

If widespread changes need to be made to the data stored in a shared address space, these changes may need to be made with the assurance that no user references are in process. In this case, the owner can return the address space to a private state by calling the ADRSPACE ISOLATE macro. This function rescinds all permissions to the address space previously granted using the ADRSPACE PERMIT macro and cancels any current addressability that other virtual machines may have to the address space. When the ISOLATE function completes, only the owning virtual machine is able to access the isolated address space.

The current addressability of other virtual machines is cancelled by changing applicable entries in their access lists from the valid to the revoked state. The sharing virtual machines find out that their access has been revoked when their next attempt to reference the address space results in an addressing-capability exception.

After updating the data in the address space, the owner must repeatedly call the ADRSPACE PERMIT macro to restore permission for each virtual machine that it wants to allow access to the address space again. The permitted virtual machines then must call ALSERV ADD to re-establish their addressability to the address space.

To reduce the cost in overall system performance, an application requiring a shared address space that periodically needs to be isolated for refresh activity may want to implement its own locking protocol instead of using the isolate service.

Destroying a Data Space

When a data space is no longer needed, the owner should call the ADRSPACE DESTROY macro to destroy it. Only the owner of a data space can destroy it. Because a data space always requires some host real storage, it is a good practice to destroy data spaces when finished with them.

The DESTROY function discards the current contents of the data space and frees all control structures associated with it. In addition, this function changes the state of the access-list entry designating the data space to revoked. If the data space was shared with other virtual machines, an implicit ISOLATE function is performed to terminate the other users' access to the data space. The owner, as well as any other virtual machines permitted to access the data space, should issue ALSERV REMOVE to change the state of the access-list entry designating the destroyed data space from revoked to unused.

Mapping Minidisks to Address Spaces

Applications that require access to very large amounts of data may be able to derive a performance benefit by establishing a mapping between the minidisks in its virtual machine's I/O configuration and a data space owned by the virtual machine, in effect using the data space as a huge buffer for DASD blocks. By temporarily placing the minidisk data in a data space, the application can use ordinary CPU instructions to access the data rather than I/O instructions.

The MAPMDISK macro lets a program establish an association between a collection of minidisk blocks on one or more minidisks and a collection of address space pages in one or more address spaces. The address spaces used for mapping can be the virtual machine's primary address space as well as data spaces that it owns. When a mapping exists, an image of the data that resides on the mapped minidisk blocks appears in the associated address space pages without requiring the application to issue I/O instructions to load the data. The I/O operations are performed implicitly by CP as references are made to the mapped address spaces.

An application executing in an XC virtual machine can invoke the following mapping service functions using the MAPMDISK macro:

IDENTIFY
Identify the minidisk pool and the pool-relative block numbers of the minidisk blocks within the pool.

DEFINE
Establish a mapping between a range of pages in an address space and a set of blocks residing in the minidisk pool.

SAVE
Initiate a request to write ranges of mapped pages to their corresponding minidisks.

REMOVE
Remove a mapping between a range of pages in an address space and a set of minidisk blocks residing in the minidisk pool.

An application using these mapping services must be in primary-space mode.

The application's first step in preparation for mapping minidisk storage to an address space is to call the MAPMDISK IDENTIFY macro to identify the collection of minidisks in its virtual machine's I/O configuration that will be mapped. This collection is known as a minidisk pool. The IDENTIFY function assigns a pool-relative block number to each block within the minidisk pool. The result of this operation is an image of a single large minidisk comprising a consecutive range of 4KB blocks of storage. The pool-

relative block numbers are used by the mapping services to establish the association between address-space pages and blocks within the minidisk pool.

Once the minidisk pool has been identified, the application uses the DEFINE function of the MAPMDISK macro to establish the mapping. When a mapping exists between an address space page and a minidisk block, a correlation is maintained between the contents of the page and the contents of the block such that data contained in the minidisk block is available in the page where it can be manipulated with CPU instructions. Upon completion of the DEFINE operation, the data that is contained on the minidisk blocks is available in the newly-mapped pages. Actual movement of the data from a minidisk block to the address space page, however, does not occur until the first reference is made to the page.

A page for which a current mapping is defined is known as a **mapped** page. A page that has never had a mapping defined or one whose mapping was subsequently removed, is known as an **unmapped** page.

When an application wants to store changes back on DASD, it uses the SAVE function of the MAPMDISK macro. The SAVE function initiates an asynchronous operation that writes mapped pages to the associated minidisk blocks. These mapped pages are those changed since they were written out to DASD.

When an application is finished referencing the mapped data, it can use the REMOVE function to discontinue the mapping association. This operation changes the status of the address space pages to unmapped, so they can be used as ordinary storage.

Notifying CP of Future Reference Patterns

If an application accesses certain data in a regular or predictable way, the tuning function provided by the REFPAGE macro may be helpful in reducing page faults and thus improving overall performance of the application. The REFPAGE macro lets you specify a range of pages that will be referenced in sequential order within a specified address space. These pages comprise a logical block. With this information, CP can pre-fetch pages into its dynamic paging area (DPA) allowing the program immediate access to its pages.

Reference patterns specified on REFPAGE can be regular or irregular. A regular pattern might occur during the processing of large arrays. Such patterns could be specified in block form. An irregular pattern might be encountered during a database server's indexed scan of data. Although the individual references may appear unrelated, they are predictable based on the contents of the index. Such apparently irregular patterns can be specified in list form, because the order of page reference is known ahead of time for a list of unordered pages.

As with any tuning function, incorrect application of the REFPAGE macro can degrade rather than improve performance. If the page ranges specified on the macro do not closely match the actual reference pattern, increased paging may result.

Chapter 25. CP Macros

This chapter contains information on the following:

- Coding CP macros
- ADRSPACE – address space services
- ALSERV – access list services
- DEFWORKA – define macro work area
- MAPMDISK – mapping services
- PFAULT – page-fault handshaking services
- REFPAGE – page reference services.
- VMUDQ – VM user directory query

If you are unfamiliar with reading syntax diagrams, see "Syntax, Message, and Response Conventions" on page xxxv.

Using the Online HELP Facility for CP Macros

You can receive online information about the CP and VM data space macros by using the z/VM HELP Facility. For example, to display a menu of the CP and data space macros, enter:

```
help vmds menu
```

To display information about a specific macro (PFAULT in this example), enter:

```
help vmds pfault
```

For more information about using the HELP Facility, see *z/VM: CMS User's Guide*. To display the main HELP task panel, enter:

```
help
```

For more information about the HELP command, see *z/VM: CMS Commands and Utilities Reference* or enter:

```
help cms help
```

Coding CP Macros

This section describes both the preferred way to code CP and VM data spaces macros and alternative methods.

Preferred Use

To simplify CP macros in both nonreentrant and reentrant environments, the macros can *automatically* control and define the work area required to execute the macro function. You must code the DEFWORKA macro once, with no operands, after the invocation of the CP macros prior to the assembler END statement (see examples). The DEFWORKA macro does not generate any executable code. It collects and defines, with proper alignment, any CP macro work areas required in your program.

Usage Notes:

1. For a reentrant program, code the DEFWORKA macro within a DSECT for which storage is acquired and released within the program. This makes a separate instance of the storage available to each

© Copyright IBM Corp. 1991, 2020

dispatchable unit executing the same reenterable program. Addressability to the storage is required prior to the invocation of the CP macros using the work area.

2. If you are coding in a nonreentrant environment, the invocation of DEFWORKA may appear within the program's control section (CSECT). This defines the macro work area as part of the program's storage.
3. If you code any CP or VM Data Spaces macro within your program and fail to code the DEFWORKA macro, you receive error messages from the assembly of your program indicating undefined references.

The use of DEFWORKA can be seen in the following examples.

Example 1: A Reentrant Program

You can reduce the complexity of coding within a reentrant environment by deferring the definition of the macro work areas to the DEFWORKA macro coded at the end of your program.

The following segments of a reentrant program outline a method for managing the CP macro work area storage. The DEFWORKA macro is coded within a DSECT, called MYDATA, containing other variables modified within this program. The program obtains CMS free storage for the area reserved by the MYDATA DSECT and then releases this storage before exiting. Two CP macros, ADRSPACE and ALSERV, are used within the program; however, the same implementation can be applied to any CP macro that relies on the DEFWORKA macro for work area definition.

```
MYCSECT  CSECT
         .
         CMSSTOR OBTAIN,BYTES=MYDATALN,...       ACQUIRE FREE STORAGE
         LR    R10,R1                    SAVE THE ADDR OF THE STORAGE
         USING MYDATA,R10                ESTABLISH ADDRESSABILITY
         .
         .
         ADRSPACE CREATE,NAME='MYSPACE',SIZE=SPACESIZ,                X
               ASIT=MYASIT               INVOKE A CP MACRO
         .
         .
         ALSERV ADD,ASIT=MYASIT,                                      X
               ALET=MYALET               INVOKE ANOTHER CP MACRO
         .
         .
         CMSSTOR RELEASE,BYTES=MYDATALN,                               X
                 ADDR=(R10),...          RELEASE THE STORAGE
         .
         .
         BR    R14                       EXIT THE PROGRAM
R1       EQU   1
R10      EQU   10
R14      EQU   14
SPACESIZ DC    F'10'          PARAMETER FOR ADRSPACE
MYDATA   DSECT
MYASIT   DS    D              PARAMETER FOR ADRSPACE
MYALET   DS    F              PARAMETER FOR ALSERV
         DEFWORKA              DEFINE CP MACRO WORK AREA
+HCPWORKA DS   (0)D,64X
MYDATALN EQU   *-MYDATA
         END                   END OF MYCSECT
```

Figure 94. Using DEFWORKA within a Reentrant Program

Example 2: A Non-Reentrant Program

In this example, a nonreentrant program invokes the DEFWORKA macro within the program's CSECT. This causes the work area for the macro to reside within the storage for the program.

```
MYCSECT  CSECT
         .
         .
         .
         ADRSPACE CREATE,NAME='MYSPACE',SIZE=SPACESIZ,                X
               ASIT=MYASIT               INVOKE A CP MACRO
         .
         .
         ALSERV ADD,ASIT=MYASIT,                                      X
               ALET=MYALET               INVOKE ANOTHER CP MACRO
         .
         .
         .
         BR    R14
R14      EQU   14
SPACESIZ DC    F'10'          PARAMETER FOR ADRSPACE
MYASIT   DS    D              PARAMETER FOR ADRSPACE
MYALET   DS    F              PARAMETER FOR ALSERV
         DEFWORKA             DEFINE CP MACRO WORK AREA
+HCPWORKA DS   (0)D,64X
         END                  END OF MYCSECT
```

Figure 95. Using DEFWORKA within a Non-reentrant Program

Alternative Methods

In most cases you will find that DEFWORKA's management of the CP macro work areas satisfies the needs of your program. However, there may be instances when you do not want to give DEFWORKA complete control over the definition of the CP macro work areas. For instance, you may want to embed the work area within a control block that persists beyond the execution of a particular program. This can be done through the CP macro's DECLARE function. DEFWORKA expands any CP macro work areas that have not yet been defined within the program by this or a preceding DEFWORKA invocation, or by the DECLARE function of the CP macros contained within the program.

The examples that follow show alternate ways to manage the macro work area should you have a requirement to do so. These examples use the CP ADRSPACE macro; any CP macro which relies on the DEFWORKA macro can be used in the same manner.

Example 1: Forcing Unique Work Areas

The following program assigns a unique workarea to each invocation of the ADRSPACE macro by using the WORKAREA operand on the ADRSPACE macro invocations. You may choose to force unique macro work areas in order to preserve the data within the work area for problem determination. The WORKAREA operand is an optional operand that can be used to identify the label associated with the work area storage.

```
 MYCSECT  CSECT
          .
          .
          ADRSPACE CREATE,...,WORKAREA=WORK1
          .
          .
          ADRSPACE CREATE,...,WORKAREA=WORK2
          .
          .
          ADRSPACE CREATE,...
          .
          .
          BR    R14                 EXIT THE PROGRAM
 R14      EQU   14
          DEFWORKA                  DEFINE CP MACRO WORK AREA
+WORK1    DS    (0)D,64X
+WORK2    DS    (0)D,64X
+HCPWORKA DS    (0)D,64X
          END                       END OF MYCSECT
```

Figure 96. Using DEFWORKA to Force Unique Macro Work Areas

In this program the ADRSPACE macro is invoked three times: the first invocation uses area WORK1 as its work area; the second invocation uses area WORK2; and, since the third invocation of ADRSPACE does not include the WORKAREA operand, the default area assigned by the DEFWORKA macro is used. DEFWORKA expands the required storage definitions for all three work areas.

Example 2: The DECLARE Function - Creating a Remote Work Area

You may have the requirement to embed the CP macro workarea in another data area (or control block) to be used by several programs. Suppose you have coded a macro, called CBLOCK, that generates a DSECT in which you want to embed the CP macro workarea. The CBLOCK macro may look like this:

```
          MACRO
          CBLOCK
 CBLOCK   DSECT
 CBLONE   DS    F                  A FIELD WITHIN THIS CONTROL BLOCK
 CBLTWO   DS    F                  ANOTHER FIELD
 CBLTHREE DS    F                  ANOTHER FIELD
 CBLMACWA ADRSPACE DECLARE         DEFINE THE WORKAREA FOR ADRSPACE
 CBLLENTH EQU   *-CBLOCK           LENGTH OF THE CONTROL BLOCK
          MEND
```

The CBLOCK macro expands a DSECT defining a data area called CBLOCK. The labels CBLONE, CBLTWO, and CBLTHREE are other fields within the CBLOCK that you define; they are shown to indicate that the control block contains data other than the CP macro workarea. You can use the DECLARE function of the ADRSPACE macro to define the storage required for the executable functions of the ADRSPACE macro. To see how the storage is defined, look at the expansion of the CBLOCK macro in the following program. Assume that storage for the CBLOCK was obtained within the calling program, and its address is provided as input to this program in register 10.

```
 MYCSECT  CSECT
          .
          USING CBLOCK,R10
          .
          .
          ADRSPACE CREATE,...,WORKAREA=CBLMACWA
          .
          .
          .
          BR    R14
 R14      EQU   14
          CBLOCK
+CBLOCK   DSECT
+CBLONE   DS    F                 A FIELD WITHIN THIS CONTROL BLOCK
+CBLTWO   DS    F                 ANOTHER FIELD
+CBLTHREE DS    F                 ANOTHER FIELD
+CBLMACWA DS    (0)D,64X
 CBLLENTH EQU   *-CBLOCK          LENGTH OF THE CONTROL BLOCK
          END                     END OF MYCSECT
```

Figure 97. Using a Remote Macro Work Area

The ADRSPACE DECLARE function within the CBLOCK macro defines the storage required for the CP macro workarea. The DEFWORKA macro is not required within this program. If you were to code DEFWORKA, the expansion of the ADRSPACE DECLARE function would indicate to DEFWORKA that workarea CBLMACWA was already defined, and DEFWORKA would not define any additional work areas.

In this example, a label is coded on the invocation of the ADRSPACE DECLARE function. This label is assigned to the work area storage and is used as the value of the WORKAREA operand.

When using a scheme such as this, you must ensure compatibility between the definition and usage of the macro work area. Any time you reassemble one of the programs using the macro work area with a new level of CP, you must also reassemble the program which uses the DECLARE function to allocate that work area.

ADRSPACE — Address Space Services

Purpose

Use the ADRSPACE macro to create or destroy address spaces and to control other virtual machine accesses to the address spaces owned by your virtual machine. You can also use the ADRSPACE macro to return information about an address space that your virtual machine owns or is authorized to access.

The following address-space service functions can be invoked using this macro:

CREATE
Create a new address space

DECLARE
Define the macro work area

DESTROY
Destroy an address space

ISOLATE
Restore an address space to the private state

PERMIT
Authorize another virtual machine for access to an address space

QUERY
Return information for an address space

Usage Notes

ESA/XC Address Spaces: The guest main storage of an XC virtual machine consists of one or more extents of storage known as absolute-storage address spaces, or simply address spaces. Each virtual machine is provided with an initial extent of main storage, called its host-primary address space, at logon. Using this macro, a virtual machine can request that CP provide additional address spaces, each from 64 kilobytes to 2 gigabytes in size. These additional address spaces are sometimes called data spaces because only data can be within the address space; a program cannot execute in a data space.

The host-primary address space of a virtual machine is always directly addressable by that virtual machine. An XC virtual machine can directly access other address spaces when the virtual machine is in the access-register mode, provided that it has a valid host access-list entry designating the address space. An XA, ESA, or Z virtual machine can indirectly access other address spaces by using DIAGNOSE code X'248' (Copy-to-primary service), provided that it has a valid host access-list entry designating the address space.

A virtual machine can use the ALSERV macro to add a host access-list entry to the virtual machine's host access list.

Address Space States: An address space is considered to be in one of two states: private or shareable.

When an address space is created by CP, it is initially in the private state. While in this state, it can only be accessed by the virtual machine that owns the address space.

An address space is changed from the private to the shareable state when the owning virtual machine uses the ADRSPACE PERMIT function to authorize another virtual machine to access the address space. When an address space is in the shareable state, it is possible for the address space to be accessed by other virtual machines (using CP commands, a program's use of access-register mode, or DIAGNOSE code X'248').

Certain DIAGNOSE code X'64' operations are prohibited when the host-primary address space is in the shareable state.

Once an address space is placed in the shareable state, it remains in this state until the owning virtual machine subsequently uses the ADRSPACE ISOLATE function to return it to the private state, or until a subsystem-reset operation is performed on the owning virtual machine.

Address Space Identification: Two identifying values are associated with each address space: an address-space name, and an address space identification token (ASIT).

An address-space name is a string of 1 to 24 characters that is assigned to an address space. The name can contain only uppercase letters (A through Z), numbers (0 through 9), and certain special characters (# $ @ _ -). Each address space owned by your virtual machine must have a different address-space name. CP assigns the name *BASE* to the host-primary address space of your virtual machine; your virtual machine assigns a name to each address space that it creates using the ADRSPACE CREATE function. Although all of the address spaces owned by your virtual machine must have different address-space names, there is no requirement that the names of your virtual machine's address spaces be different from the names of another virtual machine's address space.

The address-space name is a component of a larger identifier called a *spaceid*. A space ID is a fully-qualified name for an address space consisting of an address-space name prefixed by the user ID of the virtual machine that owns the address space. A space ID is used as a parameter for some operations, for example the DISPLAY STORAGE command, to identify a particular address space in the system.

Also associated with each address space is an address space identification token (ASIT). An ASIT is an 8-byte value assigned to the address space by CP when the address space is created. When an address space is created by the ADRSPACE CREATE function, the ASIT is returned by that function. The ADRSPACE QUERY function can be used to return the ASIT associated with the host-primary address space of your virtual machine. The ASIT is an input to functions of the ADRSPACE and other macros to designate the address space for those other operations.

An ASIT is a system-wide token that is unique for the scope of a z/VM (CP) IPL. That is, once a particular ASIT value has been assigned by CP to an address space, it will not be reassigned to another address space created by CP during the life of the current CP IPL. In addition, when CP returns the ASIT

associated with an address space (for example, as a result of the ADRSPACE QUERY function), it always returns the same value regardless of the virtual machine to which the ASIT is being returned. These characteristics let a program determine if two address spaces are the same by testing the ASITs associated with the address spaces for equality.

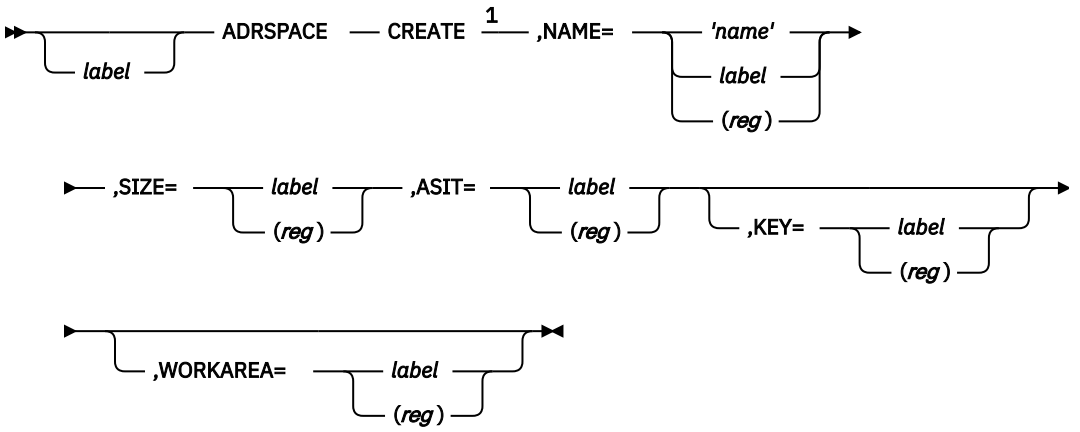
The ASIT associated with the host-primary address space of a virtual machine is sometimes used as an alternative to a userid as a means of identifying a virtual machine. When the primary-space ASIT is used in this way, it is known as a virtual configuration identification token (VCIT). Because a VCIT is in fact an ASIT, VCITs share the same characteristics as ASITs; they are system-wide tokens unique for the life of the CP IPL.

Program Exceptions

The ADRSPACE macro may result in one of the following program exceptions:

Problem Encountered	Cause
Access exception (See page “Access Exceptions” on page 8.)	An error occurred trying to • Fetch or store the macro parameter list (in macro work area) • Fetch a macro operand • Store the ASIT operand (CREATE or QUERY functions) • Store the SIZE operand (QUERY function)
Specification exception	• The CREATE or DESTROY function was requested and your virtual machine is an XA, ESA, or Z virtual machine. • The macro work area is not doubleword aligned. • The parameter list generated by the macro is in error.

ADRSPACE CREATE



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function creates a new address space and returns the ASIT associated with the new address space. All pages of the newly created address space will contain binary zeros and have the same storage key.

The maximum number of address spaces, and the maximum total size of all address spaces that your virtual machine can create is specified by the XCONFIG ADDRSPACE statement in your virtual machine's CP directory entry. If your CP directory entry does not contain a XCONFIG ADDRSPACE statement, then your virtual machine is not authorized to use this function to create any address spaces.

Address spaces that you create with this function exist until either your virtual machine explicitly destroys them, or until a subsystem-reset operation is performed on your virtual machine, for example by using the SYSTEM CLEAR, SYSTEM RESET, IPL, or LOGOFF command. When an address space is destroyed, if the host access list for your virtual machine or some other user's virtual machine has host access-list entries designating the address space, then those host access-list entries are set to the revoked state.

Your virtual machine must be an XC virtual machine to use this function. If your virtual machine is an XA, ESA, or Z virtual machine, then a specification exception is recognized.

Parameters

label
is an optional assembler language label on the macro.

NAME=
specifies the address-space name to be assigned to the new address space. This operand may be specified as one of the following:

- A character string of up to 24 characters within single quotation marks.
- A label identifying a 24-byte real storage area containing the address-space name, left-justified in the 24-byte field, and the remainder of the field padded with spaces. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label is used to determine the address space containing the storage area.
- A register (in the range of 2-12, inclusive) containing the real address of a 24-byte storage area defining the address-space name, left-justified in the 24-byte field, and the remainder of the field padded with spaces. For a virtual machine in the access-register mode, the access register

corresponding to the general register is used to determine the address space containing the storage area.

In all cases, the address-space name must contain only uppercase letters (A through Z), numbers (0 through 9), and certain special characters (# $ @ _ -). If the name contains characters that are not allowed, then no new address space is created and return code 16 is given.

Each address space owned by your virtual machine must have a different address-space name. If the address-space name specified matches the name assigned to an address space that your virtual machine previously created, or if it matches the name *BASE* which is preassigned by CP for your virtual machine's host-primary address space, then no new address space is created and return code 4 is given.

This operand is required.

SIZE=
specifies the address of a 4-byte area in real storage that contains the size for the new address space in pages; each page is 4096 bytes. Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The minimum size of an address space is 1 page (4096 bytes), and the maximum is 524,288 pages (2 gigabytes). If the size is not in this range, then return code 20 is given. If the requested size would cause the total size of all address spaces created by your virtual machine to exceed the maximum specified for your virtual machine, then no new address space is created and return code 12 is given.

The amount of storage that CP allocates for an address space must be a multiple of 256 pages (1 megabyte). Therefore, if the value of the SIZE operand is not a multiple of 256, the value is rounded up to the next multiple of 256.

This operand is required.

ASIT=
specifies the address of an 8-byte real storage area that is set by this function to be the ASIT associated with the newly created address space. Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

This operand is required.

KEY=
specifies the address of a byte in real storage containing the storage key to be assigned to every page in the new address space. Specify this operand as a label associated with the byte in real storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the byte in real storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

Bits 0-3 of the byte contain the access-control bits of the storage key and bit 4 of the byte is the fetch-protection bit. Bits 5-7 of the byte are ignored. The reference and change bits of the storage key are always set to zero.

This operand is optional. If it is not specified, the access-control and fetch-protection bits of all pages in the newly created address space will contain zeros.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ADRSPACE DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. The name specified by the NAME operand is part of the space ID for the new address space. The full space ID is *userid*:*name* where *userid* is the user ID for your virtual machine, and *name* is the address-space name specified by the NAME operand.

Condition Codes and Return Codes

On return from the CREATE function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	CREATE function status
0	The address space has been created as requested. The ASIT associated with the new address space has been returned.
4	The specified address-space name matches the name of another address space owned by your virtual machine. No new address space has been created.
8	Creating the new address space would cause the maximum number of address spaces allowed for your virtual machine to be exceeded. No new address space has been created.
12	Creating the new address space would cause the total size of all address spaces owned by your virtual machine to exceed the maximum permitted. No new address space has been created.
16	The specified address-space name contains invalid characters. No new address space has been created.
20	The specified size for the new address space is out of range. No new address space has been created.

ADRSPACE DECLARE

Purpose

This function defines the storage required for the WORKAREA operand of the ADRSPACE macro. You must code this function only if you need to define the macro work area outside of the program requiring it. To simplify the use of the ADRSPACE macro, do not code the ADRSPACE DECLARE function and omit the WORKAREA operand on the executable functions of the ADRSPACE macro. This allows the ADRSPACE macro to control and define the necessary data expansion through the DEFWORKA macro coded at the end of your program. See “Coding CP Macros” on page 793 for additional details.

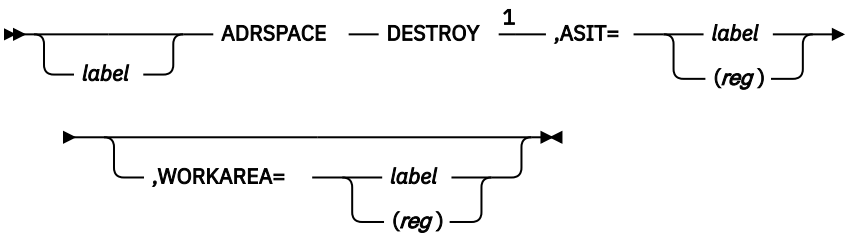
Parameters

label
is an optional assembler language label on the macro to be assigned to the defined storage.

Usage Notes

1. The DECLARE function does not generate any executable code. You may code it within a dummy section (DSECT) or a control section (CSECT).
2. Although a label is not required on the invocation of the DECLARE function, it may be necessary to identify the work area using the WORKAREA operand.

ADRSPACE DESTROY



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function destroys an address space previously created by your virtual machine.

If the address space to be destroyed is designated by any host access-list entries, either in your virtual machine's access list or the host access list of other virtual machines, those host access-list entries are set to the revoked state. If, when in the access-register mode, your virtual machine attempts to use a revoked host access-list entry, an addressing-capability exception will be recognized.

Your virtual machine must be an XC virtual machine to use this function. If your virtual machine is an XA, ESA, or Z virtual machine, then a specification exception is recognized.

Parameters

label
is an optional assembler language label on the macro.

ASIT=
specifies the address of an 8-byte real storage area that contains the ASIT identifying the address space to be deleted. Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The specified ASIT must match the ASIT associated with an address space that your virtual machine previously created, otherwise return code 4 is given.

This operand is required.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ADRSPACE DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding the use of this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

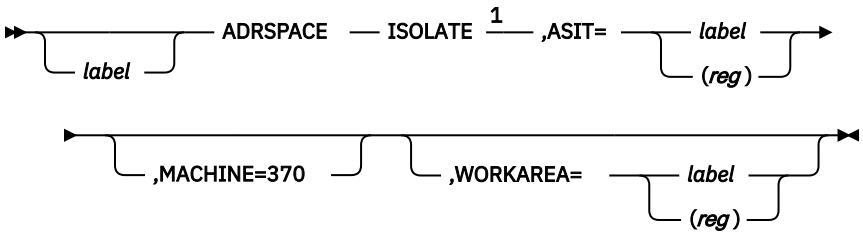
1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. The DESTROY function cannot be used to destroy the host-primary address space of your virtual machine.

Condition Codes and Return Codes

On return from the DESTROY function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	DESTROY Status
0	The address space has been destroyed as requested.
4	The specified ASIT does not identify an existing address space that your virtual machine created. This error also includes the case of the ASIT identifying the host-primary address space of your virtual machine in an attempt to destroy the host-primary address space.

ADRSPACE ISOLATE



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function restores to a private state an address space owned by your virtual machine by removing all references by other virtual machines to the address space.

If the address space to be isolated is designated by any entries in the host access lists of any virtual machine other than your own, those host access-list entries are set to the revoked state. If, when in the access-register mode, a virtual machine attempts to use a revoked host access-list entry, an addressing-capability exception will be recognized.

The execution of the ISOLATE request does not complete until all affected host access-list entries are set to the revoked state, and all virtual machines have completed any current storage accesses to the address space which is being isolated.

If your virtual machine is not an XC virtual machine, it can use this function to isolate its host-primary address space only.

Parameters

label
is an optional assembler language label on the macro.

ASIT=
specifies the address of an 8-byte real storage area that contains the ASIT identifying the address space to be isolated. Specify this operand as a *label* associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The specified ASIT must match the ASIT associated with an address space that your virtual machine owns, otherwise return code 4 is given.

This operand is required.

MACHINE=370
has no supported function, because 370 virtual machines are not supported. Results are undefined if this operand is specified.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the *label* associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for

the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ADRSPACE DECLARE function. See "Coding CP Macros" on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

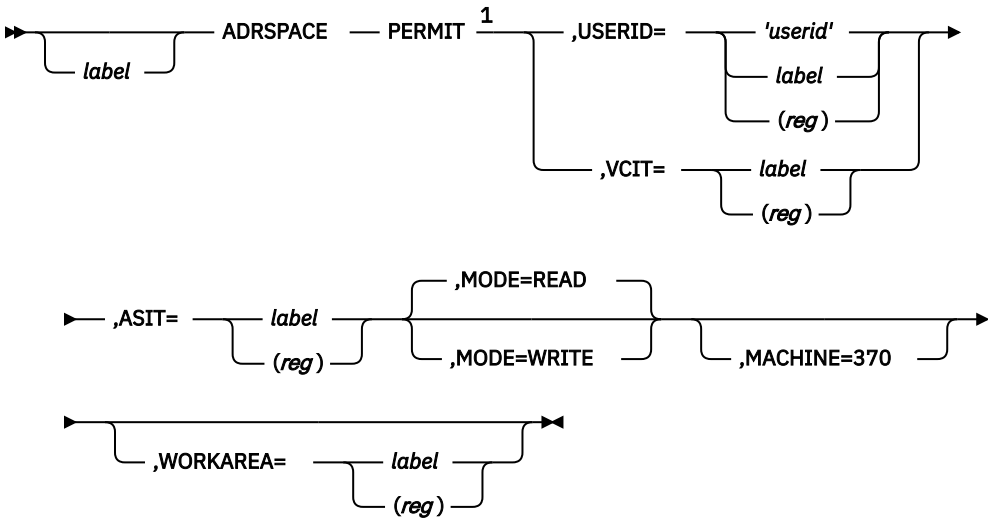
1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. The ISOLATE function may take significant time and processing to complete. Its use should be minimized.
4. The ISOLATE function does not affect any entries in your virtual machine's host access list.
5. When the ISOLATE function completes, it is guaranteed that the only references to the isolated address space will be those being made by your virtual machine.
6. If ADRSPACE ISOLATE is used specifying the ASIT of a space owned by a user that has been relocated by the VMRELOCATE command prior to a subsequent reset clear function, the request fails and return code 4 is given. To avoid this situation, issue a command that initiates a reset clear function on the user on the relocation target system. Then obtain the new ASIT value using the QUERY SPACES command or the ADRSPACE QUERY API from a program. Commands that result in a reset clear include:
   - SYSTEM CLEAR
   - IPL by NSS name
   - IPL by device with the CLear option
   - SET MACHine (to a different machine architecture)

Condition Codes and Return Codes

On return from the ISOLATE function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	ISOLATE function status
0	The address space has been isolated as requested.
4	The specified ASIT does not identify an existing address space that your virtual machine created, nor does it identify the host-primary address space of your virtual machine, or your virtual machine has been relocated through the VMRELOCATE command and no subsequent reset clear function has been invoked.

ADRSPACE PERMIT



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function authorizes a virtual machine for access to an address space that your virtual machine owns. The address space for which authorization is granted can be an address space created by your virtual machine or it can be your virtual machine's host-primary address space. You can authorize other virtual machines for either read-only, or read-write access to the address space.

When another virtual machine has authorization to access one of your virtual machine's address spaces, it can use the ALSERV macro to add to its host access list an entry designating the address space, and thereby access the address space when in access-register mode. It can also access the address space using CP commands such as CP DISPLAY, CP DUMP and CP STORE (if authorized for read-write access).

The authorization granted by using this function persists until one of the following occurs:

- Your virtual machine converts the address space for which access authorization was given to a private address space by using the ISOLATE function of this macro.
- Your virtual machine destroys the address space for which authorization was granted. The address space can be explicitly destroyed through the DESTROY function of this macro, or implicitly destroyed as a part of a subsystem-reset performed on your virtual machine, for example by the SYSTEM CLEAR, SYSTEM RESET, IPL, or LOGOFF commands.
- A subsystem reset is performed on the virtual machine to which authorization was granted. For example, a subsystem reset is performed by the SYSTEM CLEAR, SYSTEM RESET, IPL, or LOGOFF commands.

The SHARE parameter must be specified on an XCONFIG ADDRSPACE directory control statement in your virtual machine's CP directory entry to authorize you to use this function, otherwise return code 32 is given.

Parameters

label
is an optional assembler language label on the macro.

USERID=
specifies the user ID of the virtual machine to be given access authorization. This operand may be specified as one of the following:

- A character string of up to 8 characters within single quotation marks.
- A label associated with an 8-byte real storage area containing the user ID, left-justified in the 8-byte field, and the remainder of the field padded with spaces. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label is used to determine the address space containing the storage area.
- The number of a register (in the range of 2-12, inclusive) containing the real address of an 8-byte storage area defining the user ID, left-justified in the 8-byte field, and the remainder of the field padded with spaces. For an XC virtual machine in the access-register mode, the access register corresponding to the general register is used to determine the address space containing the storage area.

The user ID specified by this operand must designate a currently logged-on or disconnected virtual machine that does not already have authorization for the address space. If the user ID is not the user ID of a currently logged-on or disconnected virtual machine, then return code 28 is given. If the user ID designates a virtual machine that already has authorization for the address space, then return code 24 is given.

The USERID and VCIT operands are mutually exclusive. However, one of them is required to identify the virtual machine which is to be given access authorization.

VCIT=
specifies the address of an 8-byte real storage area that contains the VCIT identifying the virtual machine to be given access authorization. Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The VCIT for a virtual machine is the ASIT value assigned to that virtual machine's host-primary address space.

The VCIT specified by this operand must designate a currently logged-on or disconnected virtual machine that does not already have authorization for the address space. If the VCIT does not identify a currently logged-on or disconnected virtual machine, then return code 28 is given. If the VCIT designates a virtual machine that already has authorization for the address space, then return code 24 is given.

The USERID and VCIT operands are mutually exclusive. However, one of them is required to identify the virtual machine which is to be given access authorization.

ASIT=
specifies the address of an 8-byte real storage area that contains the ASIT identifying the address space for which access authorization is to be given. Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The ASIT specified by this operand must designate an address space owned by your virtual machine, that is, either an address space that your virtual machine created, or the host-primary address space for your virtual machine. If it does not designate an address space owned by your virtual machine, return code 4 is given.

This operand is required.

MODE=
specifies whether the virtual machine indicated by the USERID operand is to be given read-only, or read-write access authority. If MODE=READ is specified, the virtual machine is given authority for read-only access to the address space. If MODE=WRITE is specified, the virtual machine is given authority for read-write access to the address space.

This operand is optional; the default is MODE=READ.

MACHINE=370
has no supported function, because 370 virtual machines are not supported. Results are undefined if this operand is specified.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ADRSPACE DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. The first time that your virtual machine uses the PERMIT function to grant authorization for access to a particular address space, that address space is changed from a private address space to a shareable address space. The address space remains in the shareable state until your virtual machine uses the ISOLATE function of this macro to change it back to the private state.

   When the host-primary address space of your virtual machine is in the shareable state, certain operations of DIAGNOSE code X'64' are prohibited. See the usage notes for DIAGNOSE code X'64' for details.
4. Your virtual machine is implicitly authorized for read-write access to its host-primary address space and to all address spaces that it creates. This authorization cannot be changed by using PERMIT function (with the MODE=READ operand). Return code 24 is given if the PERMIT function is invoked with a USERID or VCIT designating your virtual machine.
5. The address space for which authorization is to be granted is specified by ASIT. When an address space is created using the CREATE function of this macro, the ASIT associated with the address space is returned. If authorization is to be granted to your virtual machine's host-primary address space, the ASIT associated with the host-primary space can be obtained using the QUERY function of this macro.
6. Your virtual machine cannot simply reissue the PERMIT function to change the type of access authorization that was previously granted to a particular virtual machine. To change previously-granted access authorization, your virtual machine must first use the ISOLATE function to revoke all users' access authority to the address space, and then reestablish the desired access authorization for each virtual machine by a series of PERMIT invocations.
7. If an external security manager is installed on your system, you may not be authorized to use the PERMIT function of this macro. For additional information, contact your security administrator.
8. If ADRSPACE PERMIT is used specifying the ASIT of a space owned by a user that has been relocated by the VMRELOCATE command prior to a subsequent reset clear function, the request fails and return code 4 is given. To avoid this situation, issue a command that initiates a reset clear function on the user on the relocation target system. Then obtain the new ASIT value using the QUERY SPACES

command or the ADRSPACE QUERY API from a program. Commands that result in a reset clear include:

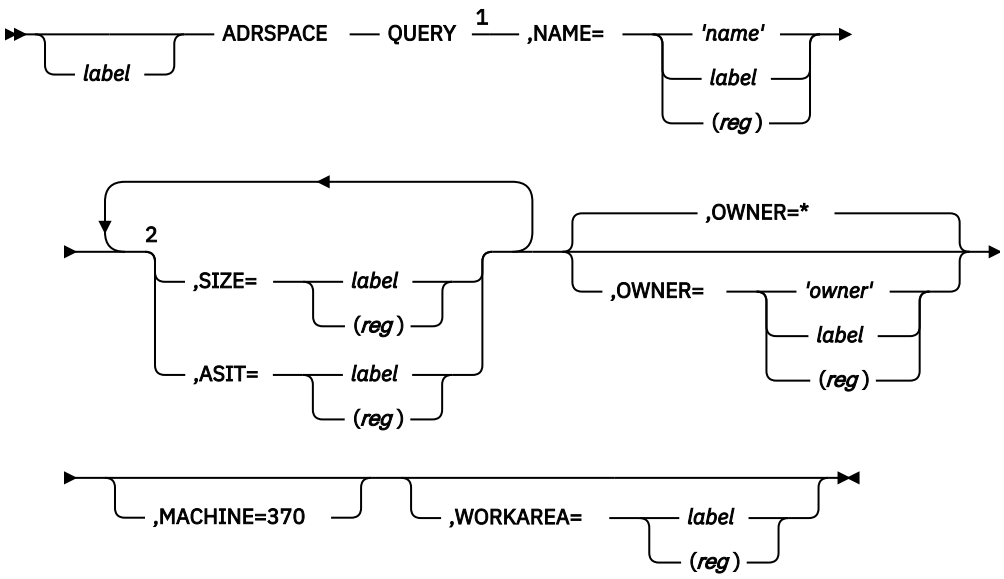
- SYSTEM CLEAR
- IPL by NSS name
- IPL by device with the CLear option
- SET MACHine (to a different machine architecture)

Condition Codes and Return Codes

On return from the PERMIT function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	PERMIT function status
0	The virtual machine specified by the USERID or VCIT operand has been authorized for access to the specified address space.
4	The specified ASIT does not identify a currently existing address space that your virtual machine owns, or your virtual machine has been relocated through the VMRELOCATE command and no subsequent reset clear function has been invoked. No authorization has been granted.
24	The specified virtual machine was already authorized for access to the specified address space. Authorization for the address space has not been changed.
28	The specified user ID or VCIT does not designate a currently logged-on or disconnected virtual machine. No authorization has been granted.
32	Your virtual machine is not authorized to use the PERMIT function of the ADRSPACE macro.

ADRSPACE QUERY



Notes:

[1] The keyword parameters that follow can be specified in any order.
[2] Each parameter can be specified only once.

Purpose

This function returns the ASIT and size associated with an address space your virtual machine owns or is authorized to access.

Parameters

label
is an optional assembler language label on the macro.

NAME=
specifies the name of the address space for which information is to be returned. This operand may be specified as one of the following:

- A character string of up to 24 characters within single quotation marks.
- A label associated with a 24-byte real storage area containing the address-space name, left-justified in the 24-byte field, and the remainder of the field padded with spaces. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label is used to determine the address space containing the storage area.
- The number of a register (in the range of 2-12, inclusive) containing the real address of a 24-byte storage area defining the address-space name, left-justified in the 24-byte field, and the remainder of the field padded with spaces. For an XC virtual machine in the access-register mode, the access register corresponding to the general register is used to determine the address space containing the storage area.

In all cases the address-space name must contain only uppercase letters (A through Z), numbers (0 through 9), and certain special characters (# $ @ _ -). If the name contains characters that are not allowed, no information is returned, and return code 16 is given.

If the OWNER operand is not specified, then this name must match the address-space name of an address space owned by your virtual machine. If it does not, then no information is returned and return code 4 is given.

If the OWNER operand is specified, then this name must match the address-space name of an address space owned by the user ID identified by the OWNER operand. Further, if the OWNER operand does not identify your virtual machine, then the owning virtual machine must have already granted authorization to your virtual machine to access the address space. If either of these conditions is not met, then no information is returned and return code 4 is given.

This operand is required.

SIZE=
specifies the address of a 4-byte real storage area that is set by this function to the size of the named address space in pages; each page is 4096 bytes.

For a host-primary address space, this includes discontiguous storage (that is, NSS or saved segment storage located above the virtual machine's defined size). Note, if discontiguous storage is used, there may be non-addressable areas within this storage.

Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

You must specify SIZE or ASIT or both to identify the address space information to be returned.

ASIT=
specifies the address of an 8-byte real storage area that is set by this function to the ASIT associated with the named address space. Specify this operand as a label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

You must specify SIZE or ASIT or both to identify the address space information to be returned.

OWNER=
specifies the user ID of the virtual machine owning the address space for which information is requested. This operand may be specified as one of the following:

- An asterisk (*) to indicate the user ID of your virtual machine.
- A character string of up to 8 characters within single quotation marks.
- A label associated with an 8-byte real storage area containing the user ID, left-justified in the 8-byte field, and the remainder of the field padded with spaces. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label is used to determine the address space containing the storage area.
- The number of a register (in the range of 2-12, inclusive) containing the real address of an 8-byte storage area defining the user ID, left-justified in the 8-byte field, and the remainder of the field padded with spaces. For an XC virtual machine in the access-register mode, the access register corresponding to general register is used to determine the address space containing the storage area.

If the OWNER operand does not identify your virtual machine, then your virtual machine must be authorized to access the specified address space; otherwise, no information is returned and return code 4 is given. The virtual machine identified by the OWNER operand authorizes access to the address space by using the PERMIT function of this macro.

This operand is optional; the default is the user ID for your virtual machine.

MACHINE=370
has no supported function, because 370 virtual machines are not supported. Results are undefined if this operand is specified.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ADRSPACE DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding the use of this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. Your virtual machine can use the QUERY function, with the operand NAME=‘BASE’ to determine the ASIT associated with your virtual machine's host-primary address space.

Condition Codes and Return Codes

On return from the QUERY function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	QUERY function status
0	The ASIT and size for the specified address space have been supplied in the locations indicated by the ASIT and the SIZE operands, respectively.
4	The specified address-space name does not identify either an address space that your virtual machine owns, or an address space owned by another virtual machine for which your virtual machine has been granted access authorization. No information has been returned.
16	The specified address-space-name contains invalid characters. No information has been returned.

ALSERV — Access List Services

Purpose

Use the ALSERV macro to add entries to, or remove entries from your virtual machine's host access list. If your virtual machine is an XC virtual machine, the host access list for your virtual machine specifies those address spaces your virtual machine can access when it is in the access-register mode. If your virtual machine is an XA, ESA, or Z virtual machine, the host access list for your virtual machine specifies those address spaces your virtual machine can access indirectly through DIAGNOSE code X'248' (Copy-to-primary service).

The following access-list services can be invoked using this macro:

ADD
Establish a valid host access-list entry

DECLARE
Define the macro work area.

REMOVE
Invalidate a host access-list entry

Usage Notes

Access Lists and Access-List Entry States: Each virtual machine has associated with it a host-managed table called a host access list. A host access list defines the set of address spaces that are directly addressable by the virtual machine when it is in the access-register mode (for XC virtual machines) or that can be accessed using DIAGNOSE code X'248' (for XA or ESA virtual machines). A host access list contains a directory-specified number of host access-list entries (ALEs), each of which is considered to be in one of three states: unused, valid or revoked. These states have an effect on the use of the entry for addressing, and the operation of the functions of the ALSERV macro, as follows:

- An unused ALE does not designate any address space and cannot be successfully used for addressing. An unused ALE can be selected by the ADD function and then set to the valid state by that function.
- A valid ALE designates a currently-existing address space and can be successfully used for addressing when your virtual machine is in the access-register mode (or when your virtual machine uses DIAGNOSE code X'248'). A valid ALE will not be selected by the ADD function.
- A revoked ALE is an ALE that was previously valid, but now designates an address space for which your virtual machine's access has been revoked. A revoked ALE cannot be successfully used for addressing. A revoked ALE will not be selected by the ADD function.

When a virtual machine is first logged on, or after a subsystem-reset operation has been performed on a virtual machine, all of the ALEs in its host access list are in the unused state.

Transitions from one ALE state to another happen as a result of the ALSERV macro, the ADRSPACE macro, and virtual-machine subsystem reset, as described below:

- An unused ALE is set to the valid state with the ADD function of this macro. When the ALE is made valid, it is set to designate a particular address space. In addition, the ALE is assigned an ALET value that is used when in the access-register mode to select this particular ALE for addressing. The ALET value remains uniquely associated with the ALE until the ALE is placed in the unused state.

  The ALET value that is assigned to the host access-list entry is returned by the ADD function of this macro. When your virtual machine is in the access-register mode (or when your virtual machine uses DIAGNOSE code X'248') and uses the ALET that selects the ALE, the storage operand associated with the ALET is considered to reside within the address space that the ALE designates.
- A valid ALE is set to the revoked state when the address space designated by the ALE is destroyed, or when access to an address space is revoked by the virtual machine that owns the address space. An address space is destroyed when its owner uses the DESTROY function of the ADRSPACE macro, or it

may be destroyed when a subsystem-reset operation is performed on the virtual machine that owns the address space. A subsystem-reset is performed, for example, by the SYSTEM CLEAR, SYSTEM RESET, LOGOFF, or IPL commands. Access to an address space can be revoked by the address space owner using the ISOLATE function of the ADRSPACE macro.

- An ALE in either the valid or revoked state is set to the unused state when the entry is removed using the REMOVE function of this macro, or when a subsystem-reset operation is performed. A subsystem-reset is performed, for example, by the SYSTEM CLEAR, SYSTEM RESET, LOGOFF, or IPL commands.

Host access lists can range in size from 6 to 1022 entries. The XCONFIG ACCESSLIST directory statement controls the size of the host access list provided for a virtual machine. If a virtual machine's CP directory entry does not contain an XCONFIG ACCESSLIST statement, then the default is that a 62-entry host access list is provided for the virtual machine.

Access-List-Entry Tokens: When a virtual machine is in the access-register mode, (or when a virtual machine uses DIAGNOSE code X'248'), the address space in which an operand resides is specified indirectly by an access-list-entry token (ALET). An ALET selects the host access-list entry to use; the host access-list entry in turn designates the particular address space containing the operand. The process of translating an ALET and determining which, if any, host access-list entry it selects (and hence which address space the ALET represents) is called the host-access-register translation process.

A host access-list entry that is in either the valid or revoked states is selected by a single ALET value for as long as that entry remains in either of these two states. A host access-list entry that is in the unused state is not selected by any ALET value.

When the host-access-register translation process attempts to translate an ALET to determine the host access-list entry that it selects, there are four possible results. The ALET:

- Selects a host access-list entry that is in the valid state.

  This is normal completion of the host-access-register translation process. The storage operand associated with the ALET is considered to reside within the address space that the host access-list entry designates.
- Selects a host access-list entry that is in the revoked state.

  A revoked host access-list entry is no longer usable for addressing; an addressing-capability exception is recognized.
- Does not select a host access-list entry

  The ALET is not usable for addressing; an ALEN-translation exception is recognized.
- Is not well formed, for example, it contains invalid bit settings.

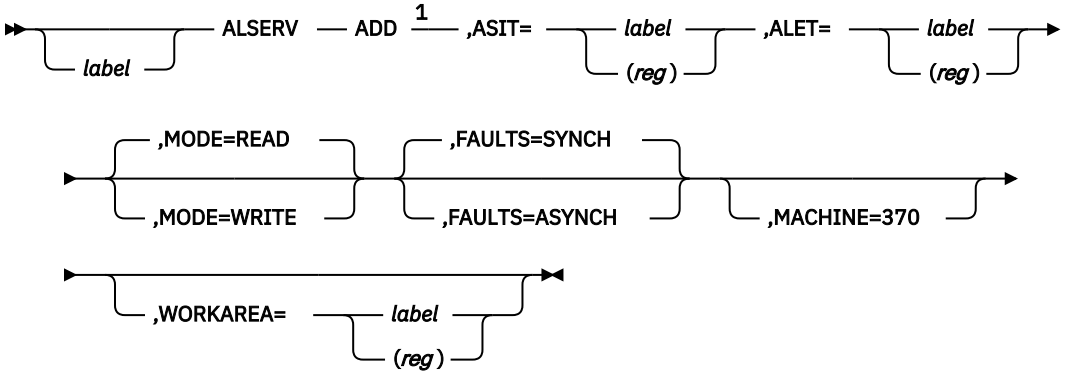
  The ALET is not usable for addressing; an ALET-specification exception is recognized.

Program Exceptions

The ALSERV macro may result in one of the following program exceptions:

Problem Encountered	Cause
Access exception (See page “Access Exceptions” on page 8.)	An error occurred trying to • Fetch or store the macro parameter list (in macro work area) • Fetch a macro operand • Store the ALET operand (ADD function)
Specification exception	• The macro work area is not doubleword aligned. • The parameter list generated by the macro is in error.

ALSERV ADD



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function establishes a valid ALE in your virtual machine's host access list. It selects an unused entry in the host access list, places that entry in the valid state designating a specified address space, and returns the ALET that your virtual machine can use to address the specified address space.

Your virtual machine must be authorized for the requested type of access (read-only or read-write) to the specified address space, otherwise return code 8 is given. Your virtual machine is always authorized for read-write access to the address spaces that it owns. Another virtual machine can authorize your virtual machine to access address spaces that it owns by using the PERMIT function of the ADRSPACE macro.

Parameters

label
is an optional assembler language label on the macro.

ASIT=
specifies the address of an 8-byte area in real storage that contains the ASIT identifying the address space to be designated by the validated ALE. Specify this operand as a label associated with the area in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the area in storage. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

This ASIT must designate a currently-existing address space that your virtual machine is authorized to access, otherwise return code 8 is given.

This operand is required.

ALET=
specifies the address of a 4-byte real storage area that is set by this function to be the ALET that your virtual machine can use to reference the address space identified by the ASIT operand. Specify the ALET operand as a label associated with the area in real storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the area in storage. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

This operand is required.

MODE=
specifies whether the new ALE allows read-only or read-write access to the designated address space. If MODE=READ is specified, then the ALE is set to provide read-only access to the address space; an attempt to modify the designated address space using this ALE will result in a protection exception.

If MODE=WRITE is specified, then the ALE is set to provide read-write access to the address space.

Your virtual machine must be authorized for the requested type of access to the address space specified by the ASIT operand. If you request either type of ALE when you have no access authorization, or a read-write ALE when you have read-only access authorization, then return code 8 is given. Your virtual machine is always authorized for read-write access to an address spaces it owns. Your virtual machine's authorization to an address space owned by another virtual machines is established by the owning virtual machine.

This operand is optional; the default is MODE=READ.

FAULTS=
specifies whether storage references that use the ALE are eligible for page-fault handshaking.

If FAULTS=ASYNCH is specified then storage references that use the ALE are eligible for page-fault handshaking, provided that the other conditions necessary for page-fault handshaking are satisfied. See Page-Fault Handshaking for details on these other conditions and the page-fault-handshaking process.

If FAULTS=SYNCH is specified then storage references that use the ALE are not eligible for page-fault handshaking.

This operand is optional; the default is FAULTS=SYNCH.

MACHINE=370
has no supported function, because 370 virtual machines are not supported. Results are undefined if this operand is specified.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ALSERV DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. The address space to be designated by the new ALE is specified by ASIT. When an address space is created using the CREATE function of the ADRSPACE macro, the ASIT associated with the address space is returned. The ASIT associated with an address space can also be obtained using the QUERY function of the ADRSPACE macro.
4. If your virtual machine invokes the ADD function more than once for the same address space (with the same or a different MODE specifications), your virtual machine's host access list will contain multiple ALEs designating that address space.

5. It is not necessary to have established the location of a page-fault handshaking token prior to establishing an ALE with the FAULTS=ASYNCH option. However, page-fault-handshaking actions will not occur for the ALE until the page-fault handshaking token is established for the virtual CPU. See Page-Fault Handshaking for details on the page-fault-handshaking process.
6. If ALSERV ADD is used specifying the ASIT of a space owned by a user that has been relocated by the VMRELOCATE command prior to a subsequent reset clear function, the request fails and return code 8 is given. To avoid this situation, issue a command that initiates a reset clear function on the user on the relocation target system. Then obtain the new ASIT value using the QUERY SPACES command or the ADRSPACE QUERY API from a program. Commands that result in a reset clear include:
   - SYSTEM CLEAR
   - IPL by NSS name
   - IPL by device with the CLear option
   - SET MACHine (to a different machine architecture)

Condition Codes and Return Codes

On return from the ADD function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	ADD function status
0	An ALE has been established to designate the specified address space. The ALET to use to reference the address space has been returned.
4	There are no unused entries in the host access list that can be used to establish the new ALE.
8	The specified ASIT does not identify a currently-existing address space for which your virtual machine is authorized for the requested type of access, or your virtual machine has been relocated through the VMRELOCATE command and no subsequent reset clear function has been invoked.

ALSERV DECLARE

Purpose

ALSERV DECLARE defines the storage required for the WORKAREA operand of the ALSERV macro. You must code this function only if you need to define the macro work area outside of the program requiring it. To simplify the use of the ALSERV macro, don't code the ALSERV DECLARE function and omit the WORKAREA operand on the executable functions of the ALSERV macro. This allows the ALSERV macro to control and define the necessary data expansion through the DEFWORKA macro coded at the end of your program. See “Coding CP Macros” on page 793 for additional details.

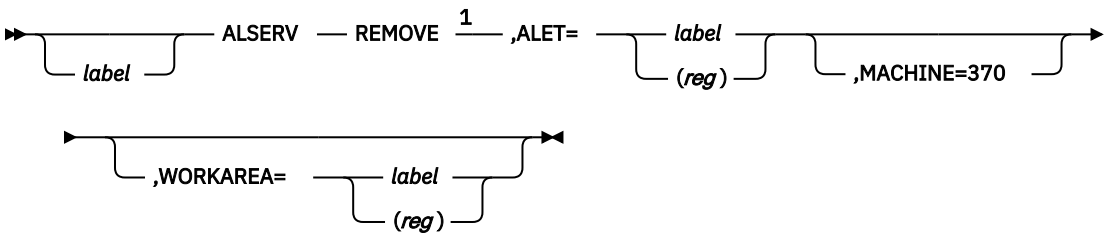
Parameters

label
is an optional assembler language label on the macro to be assigned to the defined storage.

Usage Notes

1. The DECLARE function does not generate any executable code. You may code it within a dummy section (DSECT) or a control section (CSECT).
2. Although a label is not required on the invocation of the DECLARE function, it may be necessary in order to identify the work area on the WORKAREA operand.

ALSERV REMOVE



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function sets a specified ALE in your virtual machine's host access list to the unused state.

The ALE to be modified is specified by the ALET value which selects that ALE; the specified ALE can originally be in either the valid or revoked state. After the REMOVE operation, the ALE cannot be used for addressing when your virtual machine is in the access-register mode (or cannot be used by your virtual machine through DIAGNOSE code X'248'). The ALE is available for reassignment as a valid ALE by a subsequent invocation of the ADD function.

Parameters

label
is an optional assembler language label on the macro.

ALET=
specifies the address of a 4-byte area in real storage that contains the ALET designating the ALE to be set to the unused state. Specify this operand as a label associated with the area in real storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the area in real storage. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

This ALET must designate either a valid, or a revoked entry in your virtual machine's host access list, otherwise return code 4 is given. If the ALET contains invalid bit settings such that it would cause an ALET-specification exception to be recognized if it were used in the access-register mode, then return code 12 is given.

This operand is required.

MACHINE=370
has no supported function, because 370 virtual machines are not supported. Results are undefined if this operand is specified.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For an XC virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the ALSERV DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. An ALET of X'00000000' designates the host-primary address space, and does not designate any ALE. An attempt to use the REMOVE function with an ALET of X'00000000' will result in return code 12.

Condition Codes and Return Codes

On return from the REMOVE function, register 15 contains one of the following return codes:

Return Code In Register 15 (Decimal)	REMOVE function status
0	The ALE designated by the ALET operand has been removed as requested, and the ALE is now in the unused state.
4	The ALET specified by the ALET operand does not designate an ALE in either the valid or revoked states. No ALE states have been changed.
12	The ALET specified by the ALET operand contains invalid bit settings. No ALE states have been changed.

DEFWORKA — Define Macro Work Area

Purpose

Use the DEFWORKA macro to define the work areas required for one or more CP macros contained within your program. The description of each CP macro indicates whether it supports the definition of the macro's work area through DEFWORKA.

Code DEFWORKA after the invocation of the CP macros which rely on DEFWORKA. DEFWORKA collects and defines, with proper alignment, the CP macro work areas required within your program. The macros which use DEFWORKA also support a WORKAREA operand for their executable functions, and a DECLARE function. DEFWORKA generates the work area for a CP macro which is invoked without a WORKAREA specification or a CP macro which has a WORKAREA=*label* specification, where *label* has not yet been defined through DEFWORKA or the macro's DECLARE function.

Parameters

label
is an optional assembler language label on the macro.

Usage Notes

1. The DEFWORKA macro does not generate any executable code. You may code it within a dummy section (DSECT) or the control section (CSECT) of your program.
2. A label coded on the DEFWORKA invocation is not used anywhere within the macro expansion.
3. For additional details on this macro see “Coding CP Macros” on page 793.

MAPMDISK — Mapping Services

Purpose

Use the MAPMDISK macro to establish and remove mappings between minidisks in your virtual machine's I/O configurations and address spaces owned by your virtual machine. A mapping is an association between a set of 4K minidisk blocks and a set of address space pages that permits a virtual machine to access data residing on those 4K minidisk blocks using normal CPU instructions, such as MOVE LONG, rather than by using I/O operations such as START SUBCHANNEL or DIAGNOSE code X'A4'.

Within this description of the MAPMDISK macro, the term "block", in the phrases minidisk block, pool-relative block number, minidisk-relative block number, and device-relative block number refer to a 4K byte area of data starting on a 4K block boundary. This use of the term "block" differs from a block on a fixed block architecture (FBA) DASD. Eight contiguous blocks of size 512 bytes on an FBA DASD make one minidisk 4K block. An FBA block starts on a 4K block boundary if the number of the FBA block is evenly divisible by 8.

The following mapping-service functions can be invoked using this macro:

DECLARE
Define the macro work area

DEFINE
Establish a mapping between a range of pages in an address space, and a set of 4K blocks residing in the minidisk pool

IDENTIFY
Identify the minidisk pool and the pool-relative block numbers of the 4K blocks within the pool

REMOVE
Remove a mapping between a range of pages in an address space, and a set of 4K blocks residing in the minidisk pool

SAVE
Initiate a request to write ranges of mapped pages to their corresponding minidisks

Usage Notes

Your virtual machine must be an XC virtual machine to successfully use any of the functions of this macro. If your virtual machine is an XA, ESA, or Z virtual machine, then a specification exception is recognized.

Pool-Relative Block Numbers: A minidisk pool is the collection of minidisks in your virtual machine's I/O configuration that will participate in mappings. For defining mappings, a 4K DASD block within the minidisk pool is designated by a 32-bit unsigned value called a pool-relative block number. The IDENTIFY function of this macro identifies the minidisk pool for your virtual machine and defines the assignment of pool-relative block numbers to 4K minidisk blocks contained in the minidisk pool.

Each pool-relative block number value is considered either assigned or unassigned. An assigned pool-relative block number is associated with a single 4K minidisk block for mapping-related operations. An unassigned pool-relative block number is not associated with any 4K minidisk block within the minidisk pool. The assigned or unassigned state of a particular pool-relative block number may change as a result of invoking the IDENTIFY function, or modifying your virtual machine's I/O configuration. See “MAPMDISK IDENTIFY” on page 837 for details.

Each 4K minidisk block that can be the target of a mapping is designated by at least one pool-relative block number. Although not normally the case, a 4K minidisk block may be designated by multiple pool-relative block numbers.

Effects of Mapping: A mapping is an association between a page in an address space and, indirectly through a pool-relative block number, a 4K block on a minidisk in a minidisk pool. When a mapping exists between a page and a 4K block, a correlation is maintained between the contents of the page and the contents of the 4K block so data contained in the 4K block is available in the page where it can be

manipulated by using normal CPU instructions. Mappings are established by using the DEFINE function of this macro, and are removed using the REMOVE function of this macro. MAPMDISK REMOVE must be issued before exiting guest applications that use MAPMDISK; otherwise, unpredictable results might occur.

A page for which a current mapping is defined is known as a mapped page. A page that has never had a mapping defined, or one that had a mapping defined and subsequently removed, is known as an unmapped page.

A mapping is established with the DEFINE function by specifying a pool-relative block number to be assigned to a page; this pool-relative block number indirectly specifies the 4K minidisk block that is associated with the mapped page. When a mapping is established for a page, the current contents of that page may be discarded and either the contents of the associated 4K minidisk block, or binary zeros, may be made available in the mapped page. Alternatively, the current contents of the page may be retained. The specific operation depends on the options used in establishing the mapping.

Subsequently, for as long as the page remains mapped, the contents of the page can be refreshed from the associated 4K minidisk block, and the contents of the page can be stored on the 4K minidisk block as follows:

- The contents of the mapped page can be refreshed by CP at any time, except that if the mapped page has been changed, then it will not be refreshed unless the store operation, whose description follows, has been successfully performed on the page at least as recently as the time of the last change.

  The refresh operation performed by CP consists of translating (through the then-current minidisk extent list) the pool-relative block number assigned to the page to determine the location of the associated 4K minidisk block and fetching the contents of the 4K minidisk block into the mapped page. As viewed by virtual machines referencing the mapped page, this refreshing operation updates the contents of the page in a page-concurrent manner.

  If an error is encountered when fetching data from the 4K minidisk block, the mapped page is marked *in error* so on a subsequent reference to the page a machine check indicating a storage-error condition is recognized. The storage-error condition is identified by a machine-check-interruption code (MCIC) specifying storage error uncorrected (bit 16 of the MCIC) and storage-key error uncorrected (bit 18 of the MCIC). Usually, the condition will be presented by a processing backup machine check in which all validity bits in the MCIC are set to one and a failing-storage address and ASIT are stored. However the condition may be presented by a more severe machine check with other machine-check conditions.
- The contents of the mapped page can be stored by CP at any time. In addition, a program can use the SAVE function of this macro to request that the contents of a changed mapped page be stored.

  The store operation performed by CP consists of translating (through the then-current minidisk extent list) the pool-relative block number assigned to the page to determine the location of the associated 4K minidisk block and storing the contents of the mapped page on the 4K minidisk block. The store operation is performed in a manner that maintains change integrity for the contents of the page. As viewed by virtual machines accessing the 4K minidisk block, this store operation alters the contents of the block in a block-concurrent manner.

  If an error is encountered when storing data on the 4K minidisk block, a machine check indicating a storage-degradation condition is recognized if the store operation was initiated automatically by CP, or an error completion code is presented if the store operation was initiated through the SAVE function. In both cases, the contents of the mapped page are unaffected and the page is still considered to be a changed page. The storage-degradation condition is identified by a machine-check-interruption code (MCIC) specifying storage-error corrected (bit 17 of the MCIC) and storage degradation (bit 19 of the MCIC). Usually, the condition will be presented by a system recovery machine check in which all validity bits in the MCIC are set to one and a failing-storage address and ASIT are stored. However the condition may be presented by a more severe machine check with other machine-check conditions.

Except as defined above, it is unpredictable if, and when, the refresh and store operations are performed. Changes made to a 4K minidisk block that is the target of a mapping may not have an immediate, if any, effect on the data available in the corresponding mapped pages. Likewise, changes made to a mapped page may not have an immediate, if any, effect on the contents of the associated 4K minidisk block.

Because of this unpredictability, care must be taken in using I/O instructions, or I/O DIAGNOSE functions to either read from or write to a 4K minidisk block that is the target of a mapping.

If, while a page is mapped, the pool-relative block number assigned to the page becomes an unassigned pool-relative block number, then the contents of the page will eventually become unavailable. When the contents are unavailable, the mapped page is marked *in error* such that on a subsequent reference to the page a storage-error machine check will be recognized.

Programming Notes:

1. Although data on 4K minidisk blocks is said to be available in the associated pages after some forms of the DEFINE operation are completed, no actual movement of data from the minidisk into storage occurs at the time the mapping is defined. Movement of data from the 4K minidisk block into the mapped pages is performed by CP's paging subsystem when a virtual machine makes the first reference to the mapped page.

   After a mapped page is in storage, CP's paging subsystem may or may not steal the frame assigned to the mapped page. If the frame is stolen and had been changed, the changed contents of the frame are stored back on the associated 4K minidisk block. When a virtual machine makes another reference to the page, the page will be reread from the 4K minidisk block.

   Since it is unpredictable if or when CP's paging subsystem steals a frame, your program cannot depend on this action for correct operation. In particular, your program cannot depend on CP's page stealing to cause changed data to be written to the associated 4K minidisk blocks; the SAVE function of this macro can be used to ensure predictable saving of changed data. It also cannot depend on CP refetching data from the minidisk to cause the data in a changed 4K block to be visible in a mapped page.
2. While a mapping exists, some of the pool-relative block numbers associated with mapped pages may become unassigned, for example if the minidisk containing the 4K block identified by the pool-relative block number is detached from the virtual machine's I/O configuration. When a pool-relative block number used in a mapping becomes unassigned, it is possible that the data contained on the 4K minidisk block that was identified by the (now-unassigned) pool-relative block number may remain visible in a mapped page for an unpredictable amount of time. This will occur if CP is currently backing the mapped page with a real frame. Eventually, any changes made on the mapped page will be discarded and the mapped page will become unaddressable when CP steals the frame backing the mapped page. If a subsequent reference is made to the mapped page, a storage-error machine check will be recognized.
3. Once a storage-degradation machine check or error completion code from the SAVE function is received, future storage-degradation machine checks for the same page are possible as long as the page remains mapped to the same 4K minidisk block. This is because the contents of the page are unaffected by the error and are still considered to be changed, so CP may try the store operation again on the page at any time. To eliminate the possibility of future storage-degradation machine checks, the mapping for the page should be removed (using the REMOVE function) or changed (using the DEFINE function) after a storage-degradation machine check or error completion code from the SAVE function is received.
4. Although the preceding section defines that the contents of a mapped page can be stored by CP on the 4K minidisk block at any time, this action will not normally occur unless the page has been changed. However, there are unusual situations that can cause CP to store the contents of a page that was never changed. Because of these unusual situations, your program cannot assume that unchanged pages will not be stored.
5. Minidisk cache should be turned off for minidisks that are only read and written via MAPMDISK and the corresponding data space. Minidisks which use a combination of data space access/MAPMDISK and virtual I/O (Diagnose and channel program) should be evaluated on an individual basis to determine if minidisk cache should remain enabled.

Program Exceptions

The MAPMDISK macro may result in one of the following program exceptions:

Problem Encountered	Cause
Access exception (See page "Access Exceptions" on page 8.)	An error occurred trying to • Fetch or store the macro parameter list (in macro work area) • Fetch a macro operand • Fetch an extent-list-definition block (IDENTIFY function) • Fetch an mapping-list-definition block (DEFINE function) • Access (as if a fetch) the range of pages to be mapped (DEFINE function); key-controlled protection does not apply • Fetch a save-list-definition block (SAVE function) • Store the error status buffer (SAVE function); key-controlled protection does not apply
Specification exception	• Your virtual machine is an ESA virtual machine. • The macro work area is not aligned on a doubleword boundary. • The parameter list generated by the macro is in error. • An extent-list definition block, a mapping-list definition block or a save-list definition block is not aligned on the required boundary. • The ERRBA field in a save-list definition block does not specify a nonzero address on a doubleword boundary. • The PAGEADDR parameter does not specify an address on a page boundary. • The code in bits 24-31 of the register indicated by PAGEVIEW=(reg) is invalid.

MAPMDISK DECLARE

Purpose

This function defines the storage required for the WORKAREA operand of the MAPMDISK macro. You must code this function only if you need to define the macro work area outside of the program requiring it. To simplify using the MAPMDISK macro, do not code the MAPMDISK DECLARE function and omit the WORKAREA operand on the executable functions of the MAPMDISK macro. This allows the MAPMDISK macro to control and define the necessary data expansion through the DEFWORKA macro coded at the end of your program. See “Coding CP Macros” on page 793 for additional details.

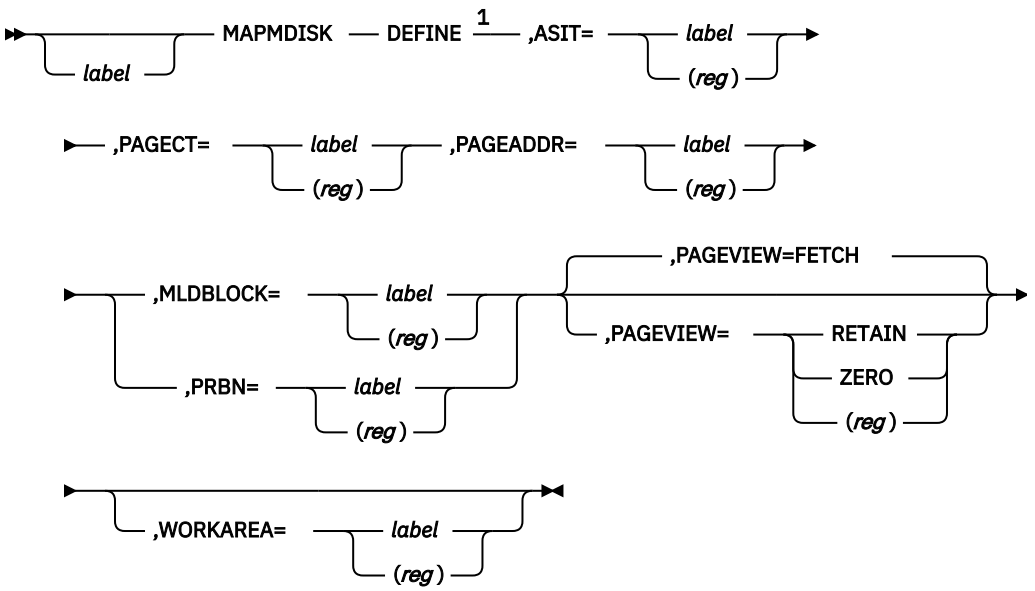
Parameters

label
is an optional assembler language label on the macro to be assigned to the defined storage.

Usage Notes

1. The DECLARE function does not generate any executable code. You may code it within a dummy section (DSECT) or a control section (CSECT).
2. Although a label is not required on the invocation of the DECLARE function, it may be necessary to identify the work area on the WORKAREA operand.

MAPMDISK DEFINE



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function establishes a mapping between a range of pages in an address space and, indirectly through pool-relative block numbers, a set of 4K blocks residing in the minidisk pool. You must first identify the minidisk pool, using the IDENTIFY function of this macro, before using the DEFINE function.

The mapping is defined in one of two ways: by using a specified list of pool-relative block numbers (the list form), or by using a specified consecutive range of pool-relative block numbers (the range form). A mapping is defined using the list form by specifying an ordered list of pool-relative block numbers that designate the 4K blocks that are to be associated with a specified consecutive range of address-space pages. A mapping is defined using the range form by specifying a consecutive range of pool-relative block numbers that designate the 4K blocks that are to be associated with a specified consecutive range of address-space pages. For both forms, the address-space pages can be pages either in an address space that you have created, or in your host-primary address space.

For each page to be mapped by the DEFINE request, the page is unlocked (if it was locked through the LOCK command) and the specified pool-relative block number is assigned to the page. Depending on the value of the PAGEVIEW parameter, the current contents of the page are either retained or discarded as part of the DEFINE operation; if the current contents are discarded, then either binary zeros or the contents of the associated 4K minidisk block are made available in the mapped page.

If PAGEVIEW=FETCH is specified and the pool-relative block number to be assigned to a page designates a 4K block that resides on a read-only minidisk, the mapped page is host-page protected by CP so any attempt to change the mapped page (or to change the storage key for the mapped page) results in a protection exception. This protection applies even if all other protection mechanisms (low-address, key-controlled and access-list-controlled protection) would allow the virtual machine to have read-write access to the page. If the pool-relative block number assigned to the page designates a 4K block that resides on a read-write minidisk, then the page is not host-page protected; a virtual machine may change the mapped page if the other protection mechanism permit the change. If PAGEVIEW=ZERO or PAGEVIEW=RETAIN is specified and the pool-relative block number to be assigned to a page designates a 4K block that resides on a read-only minidisk, return code 36 is given.

Once a mapping is established, the mapping remains in effect and the pool-relative block number specified for a page remains assigned to the page until a new mapping is established, until the mapping is removed with the REMOVE function of this macro, or until a subsystem-reset operation is performed on your virtual machine. The mapping remains in effect even if the extent-list defining the minidisk pool is changed after the mapping was established. MAPMDISK REMOVE must be issued before exiting guest applications that use MAPMDISK; otherwise, unpredictable results might occur.

Parameters

label
is an optional assembler language label on the macro.

ASIT=
specifies the real address of an 8-byte field in storage that contains the ASIT identifying the address space to be the target of the mapping. Specify this parameter as a label associated with the field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

If this ASIT is not associated with an address space your virtual machine created, or the host-primary address space for your virtual machine, then return code 8 is given.

This operand is required.

PAGECT=
specifies the real address of a 4-byte field in storage that contains the number of consecutive pages to be mapped by this request. Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The minimum number of pages to be mapped is 1, and the maximum number is 524,288. If this count is not in this range, then return code 4 is given.

If this value does not match the sum of the ENTCT fields in each of the mapping-list definition blocks, then return code 20 is given.

This operand is required.

PAGEADDR=
specifies the real address of a 4-byte field in storage that contains the 31-bit absolute address of the first page in the consecutive range of pages to be mapped. Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The range of pages to be mapped, from PAGEADDR to PAGEADDR+(PAGECT-1)*4096, must be contained within the bounds of the address space designated by the ASIT operand, otherwise an addressing exception is recognized. This range must not include any saved segments, or pages locked with the LOCK function of DIAGNOSE code X'98' otherwise return code 24 is given.

Bits 20-31 of the page address must be zeros, otherwise a specification exception is recognized.

This operand is required.

MLDBLOCK=
specifies the real storage address of the first mapping-list definition block in the chain of mapping-list definition blocks for this request. Specify this parameter as a label associated with the block in real storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the

block in real storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the block.

The address of the first mapping-list definition block must be an address on a doubleword boundary, otherwise a specification exception is recognized. See Mapping-List Format for more information.

The MLDBLOCK and PRBN operands are mutually exclusive. However, one of them is required to indicate whether the pool-relative block numbers to be associated with the mapped pages are specified in a list or as a consecutive range.

PRBN=
specifies the real address of a 4-byte field in storage that contains the pool-relative-block number to be associated with the page identified by the PAGEADDR operand. Each subsequent page in the range of pages to be mapped is associated with the next higher pool-relative block number.

Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The pool-relative block number must be less than or equal to 2^{32}-PAGECT and must have been assigned by the IDENTIFY function, otherwise return code 12 is given.

The MLDBLOCK and PRBN operands are mutually exclusive. However, one of them is required to indicate whether the pool-relative block numbers to be associated with the mapped pages are specified in a list or as a consecutive range.

PAGEVIEW=
specifies the disposition of the current contents of the pages to be mapped and the type of data to be available in the mapped pages at the completion of the DEFINE request. Specify this parameter as one of the keywords FETCH, RETAIN or ZERO or as the number of a register (in the range of 2-12 inclusive) containing a parameter code in the low order byte.

If PAGEVIEW=FETCH is specified, the current contents of the pages are discarded and the contents of the associated 4K minidisk blocks are made available in the mapped pages. For the SAVE function, the pages are considered to be unchanged pages at the completion of the DEFINE operation.

If PAGEVIEW=RETAIN is specified, the current contents of the pages are retained. For purposes of the SAVE function, the pages are considered to be changed pages.

If PAGEVIEW=ZERO is specified, the current contents of the pages are discarded and the mapped pages are considered to contain binary zeros. For purposes of the SAVE function, it is unpredictable whether the pages are considered to be changed or unchanged pages at the completion of the DEFINE operation.

If PAGEVIEW=(*reg*) is specified, then bits 24-31 of the general register indicated by (*reg*) contain a code that specifies PAGEVIEW as follows:

Code
PAGEVIEW meaning:

X'00'
PAGEVIEW=FETCH

X'01'
PAGEVIEW=RETAIN

X'02'
PAGEVIEW=ZERO

All Others
Invalid

If bits 24-31 of the register contain a code listed as invalid then a specification exception is recognized. Bits 0-23 of the register are ignored.

This operand is optional; the default is PAGEVIEW=FETCH.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this parameter as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the MAPMDISK DECLARE function. See "Coding CP Macros" on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. The macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. The normal method of changing data on a 4K minidisk block that is the target of a mapping is to change the contents of the associated mapped page and then use the SAVE function of this macro to cause the changed data to be stored on the 4K minidisk block.

   However if the data on a 4K minidisk block must be changed by some other means, such as an I/O instruction or an I/O diagnose, and you want the changed data to be visible in a mapped page, issue DIAGNOSE code X'10' against the mapped page after the data has been changed on the 4K minidisk block. The DIAGNOSE code X'10' operation will cause the mapped page to be refreshed from the 4K minidisk block on next reference.
4. Since it is unpredictable whether mapped pages established using the PAGEVIEW=ZERO operand are considered changed or unchanged pages, your program cannot use the combination of a DEFINE request with PAGEVIEW=ZERO followed immediately by a SAVE request for the same page as a way of zeroing out the contents of the associated 4K minidisk block. The SAVE request may or may not store the page of zeros on the 4K minidisk block.

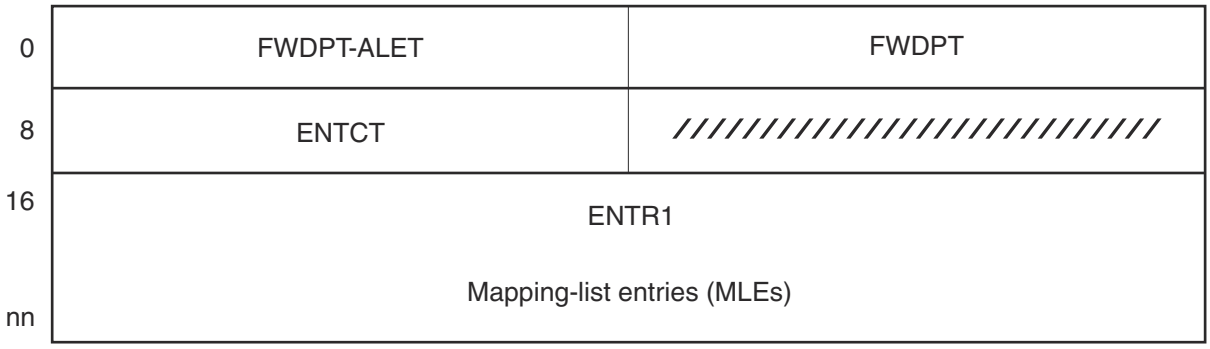
   If you want the page of zeros to be stored on the associated 4K minidisk block by the SAVE request, then the page must be changed through CPU instructions before the SAVE request is issued. Note however that simply storing a byte or word of zeros into the page is insufficient because by architecture rules, the page may not be considered to be a changed page after a store operation that does not change the value in storage. Two stores into the page must be done: the first being a store of some nonzero value, and the second a store of a zero into the same location.
5. Defining a mapping using the DEFINE function changes the data that is visible in the mapped page. However, it does not change the storage key associated with the page.

Mapping-List Format: If the DEFINE function has been requested using a specified list of pool-relative block numbers (using the MLDBLOCK parameter), then the pool-relative block numbers to be associated with the pages to be mapped are specified by a singly-linked list of mapping-list definition blocks. This mapping is a chained structure to allow for the specification of many mapping-list entries without requiring many contiguous pages of storage for the list, while retaining the option of having a single contiguous area if desired. The real address of the first block in this singly-linked list is specified by the MLDBLOCK operand.

Each mapping-list definition block contains a header, and a contiguous table of mapping-list entries. The mapping-list definition block must be aligned on a doubleword boundary. Each mapping-list entry specifies a pool-relative block number to be associated with a page in the range of pages to be mapped.

Each mapping-list definition block must be aligned on a doubleword boundary. A mapping-list definition block has the following format:

Note: Field names used in the following descriptions are for reference only. You may copy these names, or use any other names, in your own code.



FWDPT-ALET
When the virtual machine is in the access-register mode, bytes 0-3 contain an ALET specifying the address space in which the next mapping-list definition block resides. Zeros in bytes 0-3 in conjunction with zeros in bytes 4-7 indicate that this block is the last block in the chain.

Bytes 0-3 are ignored when the virtual machine is in the primary-space mode.

FWDPT
Bytes 4-7 of the mapping-list definition block are the real address of the next block in the singly-linked chain. This field must specify an address on a doubleword boundary, otherwise return code 28 is given. When the virtual machine is in the primary-space mode, zeros in bytes 4-7 indicate that this block is the last block in the chain. When the virtual machine is in the access-register mode, end-of-chain is indicated by zeros in bytes 4-7 and bytes 0-3.

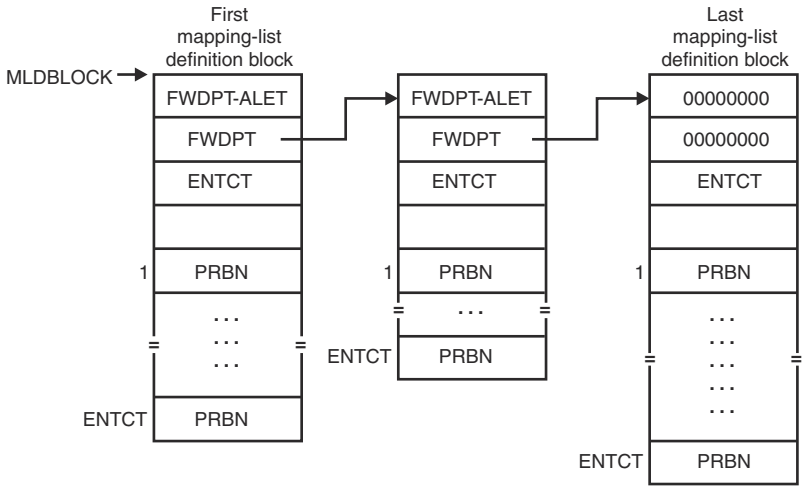
ENTCT
Bytes 8-11 of the mapping-list definition block contain a word that is the number of mapping-list entries contained in this mapping-list definition block. The minimum number of entries in a mapping-list definition block is 1, and the maximum number is 524,288. If this count is not in this range, then return code 16 is given.

If the total number of mapping-list entries in all of the extent-list definition blocks in the chain is less than the value of the page count specified by the PAGECT operand on the macro invocation, then a mapping is established only for those pages for which there is a mapping-list entry. If the total number of mapping-list entries is larger than the value of the PAGECT operand, then the extra mapping-list entries are not used. In either case, return code 20 is given.

////////
Bytes 12-15 of the mapping-list definition block are reserved for future use and should contain binary zeros.

ENTR1
Starting at byte 16 of the mapping-list definition block is a contiguous table of 4-byte mapping list entries. Each mapping-list entry is a word that contains a pool-relative block number. The ENTCT field specifies how many mapping-list entries are contained in this area.

The overall structure of the mapping list chain is shown in the following diagram:



The chain of mapping-list definition blocks is processed as though it was a single, ordered list of mapping-list entries. The first mapping-list entry in the first mapping-list definition block contains the pool-relative block number to be associated with the address-space page identified by the PAGEADDR operand on the macro invocation. Each subsequent mapping-list entry in the mapping-list definition block contains the pool-relative block number to be associated with the next page in the page range. After all of the mapping-list entries in a given mapping-list definition block are processed, the first mapping-list entry of the next mapping-list definition block in the chain is processed.

Condition Codes and Return Codes

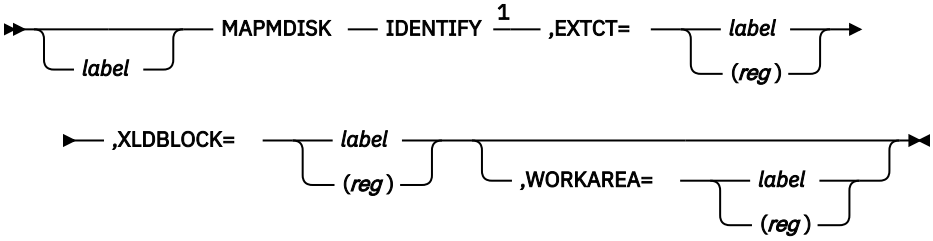
On return from the DEFINE function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	DEFINE function status
0	The mapping has been defined as requested.
4	The page count specified by the PAGECT operand is out of range. No mapping has been defined.
8	The ASIT specified by the ASIT operand does not identify either an existing address space that your virtual machine created, or your virtual machine's host-primary address space. No mapping has been defined.
12	The pool-relative block number to be associated with a page is either unassigned, or would exceed 2^{32}-1. If the error is due to an unassigned pool-relative block number, mappings have been defined for pages up to but not including the one corresponding to this error. Register 0 contains the address of the page for which the error occurred. This page resides in the address space specified by the ASIT operand. If the error is due to PRBN+PAGECT-1 exceeding 2^{32}-1, then no mappings have been defined.
16	The ENTCT field in a mapping-list definition block is out of range. Register 0 contains the address of the mapping-list definition block that contains the invalid ENTCT field. In the access-register mode, access register 0 contains the ALET specifying the address space in which the mapping-list definition block resides. Mapping-list definition blocks up to but not including the one containing the error have been processed.

Return Code in Register 15 (Decimal)	DEFINE function status
20	The page count as specified by the PAGECT operand does not match the sum of the ENTCT fields in all of the mapping-list definition blocks. If the sum of the ENTCT fields is less than the page count specified by PAGECT, then all mapping-list definition blocks have been processed; register 0 contains the address of the last mapping-list definition block. If the sum of the ENTCT fields is greater than the page count specified by PAGECT, then mapping-list definition blocks up to but not including the one that caused the error have been processed; register 0 contains the address of the mapping-list definition block that caused the error. In both cases in the access-register mode, access register 0 contains the ALET specifying the address space in which the mapping-list definition block resides.
24	The range of pages to be mapped includes a saved segment, or includes a page locked with the LOCK function of DIAGNOSE code X'98'. Mappings have been defined for all pages up to but not including the first page that is either in a saved segment, or that has been locked with the LOCK function of DIAGNOSE code X'98'. Register 0 contains the address of the megabyte containing the page for which the error occurred. This megabyte of storage resides in the address space specified by the ASIT operand.
28	The FWDPT field in a mapping-list definition block does not specify an address on a doubleword boundary. Mapping-list definition blocks up to and including the one containing the error have been processed. Register 0 contains the address of the mapping-list definition block containing the invalid FWDPT field. In the access-register mode, access register 0 contains the ALET specifying the address space in which the mapping-list definition block resides.
32	No minidisk pool was previously identified through the MAPMDISK IDENTIFY function. No mapping has been defined.
36	PAGEVIEW=ZERO or PAGEVIEW=RETAIN was specified and a pool-relative block number designates a 4K block that resides on a read-only minidisk. Mappings have been defined for pages up to but not including the one corresponding to this error. Register 0 contains the address of the page for which the error occurred. This page resides in the address space specified by the ASIT operand.
40	When in the access-register mode, an ALET used in accessing a mapping-list definition block could not be translated because of an ALET-specification-exception condition (ALET contains invalid bit settings). Mapping-list definition blocks up to and including the one containing the error have been processed. Access register 0 and general register 0 contain the ALET and address (respectively) used in the attempt to access the mapping-list definition block.
44	When in the access-register mode, an ALET used in accessing a mapping-list definition block could not be translated because of an ALEN-translation-exception condition (ALET does not designate a valid or revoked ALE). Mapping-list definition blocks up to and including the one containing the error have been processed. Access register 0 and general register 0 contain the ALET and address (respectively) used in the attempt to access the mapping-list definition block.
48	When in the access-register mode, an ALET used in accessing a mapping-list definition block could not be translated because of an addressing-capability-exception condition (ALET designates an address space for which access has been revoked). Mapping-list definition blocks up to and including the one containing the error have been processed. Access register 0 and general register 0 contain the ALET and address (respectively) used in the attempt to access the mapping-list definition block.

If an addressing or addressing-capability exception is recognized on the DEFINE function, the operation is terminated. Those pages in the page range identified by the PAGEADDR and PAGECT parameter that are processed before the point of error have mappings established as usual. Those pages in the page range at or after the point of error remain unchanged.

MAPMDISK IDENTIFY



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function identifies your virtual machine's minidisk pool and defines the assignment of pool-relative block numbers to the 4K blocks contained within the minidisk pool. You must use this function to identify the minidisk pool before you can successfully use the DEFINE function to establish mappings.

To be in a minidisk pool, a minidisk must reside on a DASD device type that is supported by CP as a paging device; this requirement is checked by the IDENTIFY function when the minidisk pool is identified. For minidisks on FBA DASD, the minidisk must start and end on 4K block boundaries (the starting FBA block number and the ending FBA block number plus one must be evenly divisible by 8); this requirement is also checked by the IDENTIFY function when the minidisk pool is identified. For minidisks on CKD DASD, the MAPMDISK functions assume that all minidisks that are part of a minidisk pool are formatted so:

- All blocks are 4K byte blocks
- Each track has consecutive record numbers starting with one
- There are no pad records.

However, these last three requirements for minidisks on CKD DASD are not checked by the IDENTIFY function. Unpredictable results may occur if these requirements are not met.

The minidisk pool is specified by an extent list, which is an ordered list of minidisk segments that comprise the minidisk pool. Each extent in the list is a contiguous range of 4K blocks on a minidisk in your virtual machine's I/O configuration. The 4K blocks contained in an extent are assigned a specified consecutive range of pool-relative block numbers that designate the blocks in mapping operations. The extent list entries must be specified with increasing, nonoverlapping ranges of pool-relative block numbers.

Your virtual machine can have at most one minidisk pool defined. If your virtual machine had previously used this function to identify a minidisk pool, and then subsequently uses the function again, the new extent list for the minidisk pool replaces the previous one.

Once a minidisk pool is identified, it remains in effect until it is redefined by a subsequent invocation of the IDENTIFY function, or until a subsystem-reset operation is performed on your virtual machine. A subsystem-reset operation deletes any minidisk pool identified for your virtual machine.

If a minidisk contained within your virtual machine's minidisk pool is subsequently detached from your virtual machine's I/O configuration, or if the minidisk is redefined to have a different device number, then the minidisk is removed from the minidisk pool, and all pool-relative block numbers corresponding to the extents that were defined for that minidisk are made unassigned pool-relative block numbers. The minidisk remains outside of the minidisk pool until it is reattached to your virtual machine's I/O configuration and another IDENTIFY operation is performed which includes one or more extents residing on the minidisk device.

Parameters

label
is an optional assembler language label on the macro.

EXTCT=
specifies the real address of a 4-byte field that contains the total number of extents contained in the extent list. (See Extent-List Format for more information on the extent list.) Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The minimum number of extents is 1, and the maximum number is 65,536. If this count is not in this range, then return code 4 is given.

If this value does not match the sum of the ENTCT fields in each of the extent-list definition blocks, then return code 20 is given.

This operand is required.

XLDBLOCK=
specifies the real storage address of the first extent-list-definition block in the chain of extent-list-definition blocks for this request. Specify this parameter as a label associated with the block in real storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the block in real storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

This address of the first extent-list-definition block must be an address on doubleword boundary, otherwise a specification exception is recognized. See Extent-List Format for more information.

This operand is required.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this parameter as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the MAPMDISK DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. It is possible to specify an extent list that has *gaps* in the assignment of pool-relative block numbers, that is, has pool-relative block numbers that are not assigned to the 4K blocks in any minidisk extent. Such an assignment can be used, for example, to skip ranges of pool-relative block numbers that correspond to off-line volumes.
4. A mapping between a page and a particular 4K minidisk block is established indirectly, by a pool-relative block number which is translated as necessary by CP to determine the associated 4K minidisk block. Because of this indirection, and the fact that a mapping association between a particular page

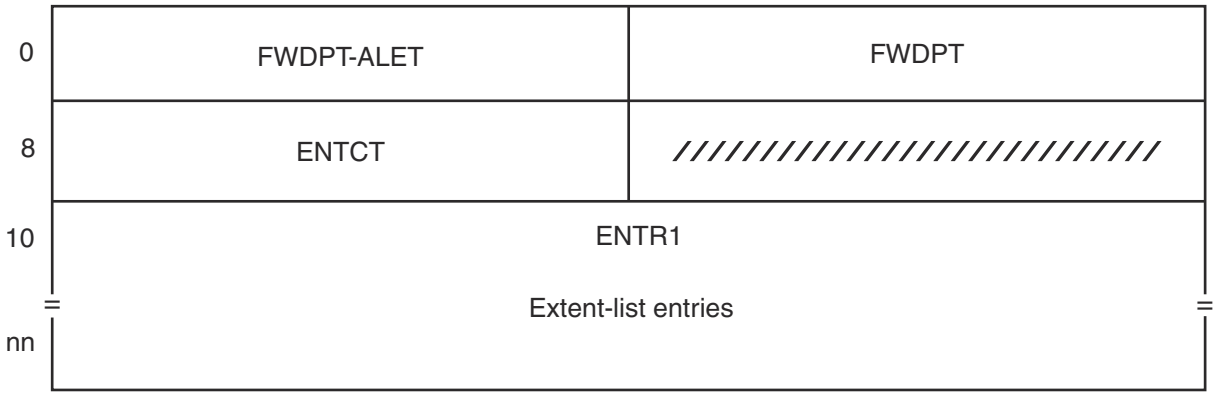
and a particular pool-relative block number remains in effect even if the current extent-list defining the minidisk pool is changed, care must be taken when changing the extent list defining the minidisk pool. Since it is unpredictable when, and how many times, a pool-relative block number is translated by CP, unpredictable operation will result if the extent list is changed so that either a particular pool-relative block number is assigned to a different 4K minidisk block in the new extent list than it was in the old extent list, or a particular pool-relative block number is unassigned in the new extent list but was assigned in the old extent list.

5. To be successfully used for mapping, a minidisk must be formatted with certain characteristics, as stated above. Minidisks formatted using the CMS FORMAT command with the BLKSIZE 4096 or BLKSIZE 4K option meets all of these formatting requirements.
6. For a minidisk on an Extended Address Volume, the minidisk end extent must reside below cylinder 65520 to be eligible for the MAPMDISK IDENTIFY function.

Extent-List Format: The extent list for the minidisk pool is specified by a singly-linked chain of extent-list definition blocks. The extent-list is a chained structure to allow for the specification of many mapping-list entries without requiring many contiguous pages of storage for the list, while retaining the option of having a single contiguous area if desired. The address of the first extent-list definition block in the chain is specified by the XLDBLOCK operand on the MAPMDISK IDENTIFY macro invocation.

Each extent-list definition block begins on a doubleword boundary, and is in the following format:

Note: Field names used in the following descriptions are for reference only. You may copy these names, or use any other names, in your own code.



FWDPT-ALET
When the virtual machine is in the access-register mode, bytes 0-3 contain an ALET specifying the address space in which the next extent-list definition block resides. Zeros in bytes 0-3 in conjunction with zeros in bytes 4-7 indicate that this block is the last block in the chain.

Bytes 0-3 are ignored when the virtual machine is in the primary-space mode.

FWDPT
Bytes 4-7 of the extent-list definition block are the real address of the next block in the singly-linked chain. This field must specify an address on a doubleword boundary, otherwise a specification exception is recognized. When the virtual machine is in the primary-space mode, zeros in bytes 4-7 indicate that this block is the last block in the chain. When the virtual machine is in the access-register mode, end-of-chain is indicated by zeros in bytes 4-7 and bytes 0-3.

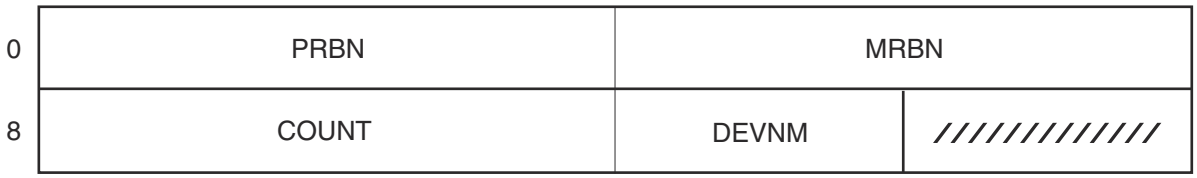
ENTCT
Bytes 8-11 of the extent-list definition block contain a word that is the number of extent-list entries contained in this extent-list-definition block. The minimum number of entries in the extent-list definition block is 1, and the maximum number is 65,536. If this count is not in this range, then return code 16 is given.

If the sum of the ENTCT fields in all of the extent-list-definition blocks does not match the value specified for the EXTCT operand on the MAPMDISK IDENTIFY macro invocation, then return code 20 is given.

////////
Bytes 12-15 of the extent-list-definition block are reserved for future use and should contain binary zeros.

ENTR1
Starting at byte 16 of the extent-list definition block is a contiguous table of 16-byte extent-list entries. The ENTCT field specifies how many extent-list entries are contained in this area. The format of each extent-list entry follows.

Each 16-byte extent-list entry defines a single extent in the minidisk extent list. An extent-list entry has the following format:



PRBN
Bytes 0-3 of the extent-list entry are an unsigned word that contains the pool-relative block number to be assigned to the 4K block designated by the MRBN field. This number must be larger than the pool-relative block numbers assigned to previous extents, and must be less than or equal to 2^{32}-COUNT. If this number is not greater than the pool-relative block numbers assigned to previous extents, or if this number is too large, return code 24 is given.

The 4K blocks residing on the extent defined by this extent-list entry are assigned consecutive pool-relative block numbers from PRBN to PRBN+COUNT-1, inclusive. This range of pool-relative block numbers is assigned to the 4K blocks within the extent in minidisk-relative block order.

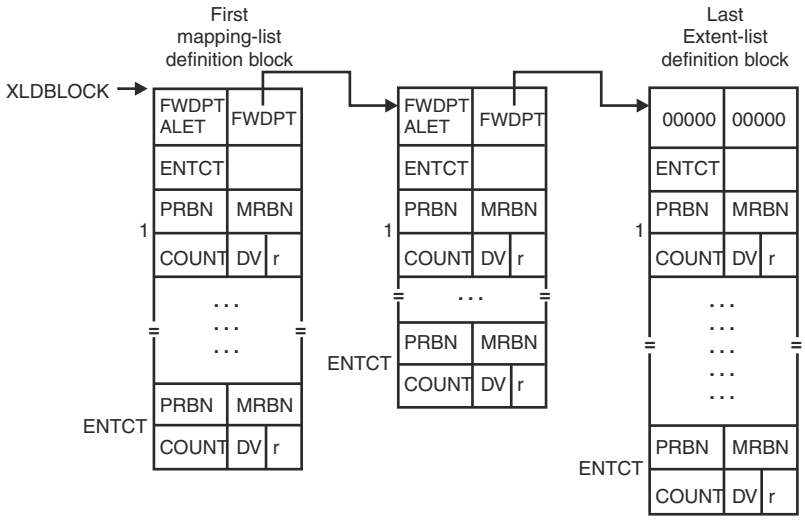
MRBN
Bytes 4-7 of the extent-list entry are a word that contains the minidisk-relative block number of the first 4K block of the minidisk that is included in the extent. This block number is relative to the start of the minidisk, and must be between 0 and the number of 4K blocks contained on the minidisk minus one. If the number is not in this range, return code 12 is given.

COUNT
Bytes 8-11 of the extent-list entry are a word that contains the number of 4K blocks of the minidisk that are included in the extent. This count must be greater than or equal to 1, and the sum of MRBN and COUNT must be less than or equal to the number of 4K blocks contained on the minidisk. If either of these conditions is not met, then return code 12 is given.

DEVNM
Bytes 12-13 of the extent-list entry are a halfword that contains the virtual device number of the minidisk on which this extent resides. This must be the device number of a minidisk in your virtual machine's I/O configuration that resides on a device type supported by CP. If the device is not a minidisk that resides on a device type supported by CP, then return code 8 is given. If the device is a minidisk residing on an FBA DASD, the minidisk must start and end on a 4K block boundary (the starting FBA block number and the ending FBA block number + 1 of the minidisk must be evenly divisible by 8). If not, return code 28 is given.

////////
Bytes 14-15 of the extent-list entry are reserved for future use and should contain binary zeros.

The overall structure of the chain of extent-list definition blocks is shown in the following diagram:



The chain of extent-list-definition blocks is processed as though it was a single, ordered list of extent-list entries. The first extent-list entry in the first extent-list-definition block is the first extent processed by the identify-pool function. Each subsequent extent-list entry in the extent-list-definition block is processed in turn. After all of the extent-list entries in a given extent-list-definition block are processed, the first entry of the next extent-list-definition block in the chain is processed.

Condition Codes and Return Codes

On return from the IDENTIFY function, register 15 contains one of the following return codes:

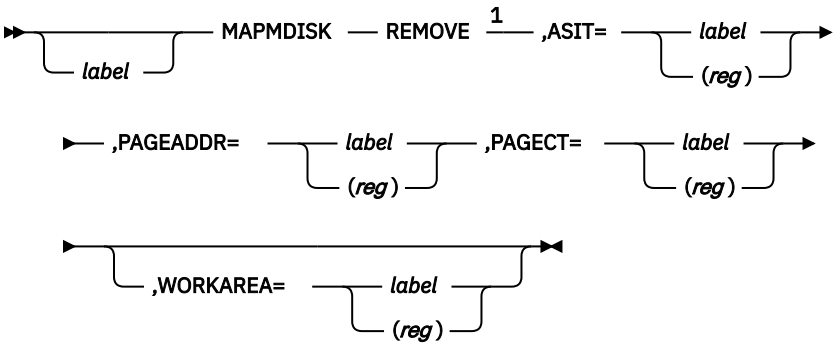
Return Code in Register 15 (Decimal)	IDENTIFY function status
0	The minidisk pool has been identified as requested.
4	The total number of extents as specified by the EXTCT operand is out of range. No new minidisk pool has been identified.
8	An extent-list entry contains either (1) a device number that does not exist in the virtual machine's I/O configuration, (2) a device number for a device that is not supported for mapping operations (not a minidisk device, or a minidisk device residing on a device type not supported by CP for paging, or a virtual disk in storage), or (3) a minidisk end extent is not below cylinder 65520 on an Extended Address Volume. Register 0 contains the address of the extent-list entry that contains the invalid device number. In the access-register mode, access register 0 contains the ALET specifying the address space in which the extent-list entry resides. No new minidisk pool has been identified.
12	An extent-list entry contains a MRBN field that is out of range, or the sum of MRBN and COUNT is out of range, or COUNT is less than or equal to zero. Register 0 contains the address of the extent-list entry. In the access-register mode, access register 0 contains the ALET specifying the address space in which the extent-list entry resides. No new minidisk pool has been identified.
16	The ENTCT field in an extent-list definition block is out of range. Register 0 contains the address of the extent-list definition block containing the invalid ENTCT field. In the access-register mode, access register 0 contains the ALET specifying the address space in which the extent-list definition block resides. No new minidisk pool has been identified.

Return Code in Register 15 (Decimal)	IDENTIFY function status
20	The total number of extents as specified by the EXTCT operand does not match the sum of the ENTCT fields in all of the extent-list-definition blocks. No new minidisk pool has been identified. If the sum of the ENTCT fields is less than the extent count specified by EXTCT, then register 0 contains the address of the last extent-list definition block. If the sum of the ENTCT fields is greater than the extent count specified by EXTCT, then register 0 contains the address of the extent-list definition block whose ENTCT field caused the sum of ENTCT fields to exceed the extent count specified by EXTCT. In both cases in the access-register mode, access register 0 contains the ALET specifying the address space in which the extent-list definition block resides.
24	An extent-list entry contains a PRBN field that is either too large, or not larger than the pool-relative block numbers assigned to previous extents. Register 0 contains the address of the extent-list entry containing the invalid PRBN. In the access-register mode, access register 0 contains the ALET specifying the address space in which the extent-list entry resides. No new minidisk pool has been identified.
28	An extent-list entry contains a device number for a minidisk residing on an FBA DASD. That minidisk either does not start or does not end on a 4K block boundary. (The starting FBA block number and the ending FBA block number + 1 of the minidisk must be evenly divisible by 8). No new minidisk pool has been identified. R0 contains the address of the non-aligned minidisk. In access register mode, access register 0 contains the ALET specifying the address space in which the extent list entry resides.

If an addressing or addressing-capability exception is recognized on the IDENTIFY function, the operation is suppressed. No new minidisk pool is identified and the minidisk pool identified by a previous IDENTIFY request, if any, remains in effect.

If the IDENTIFY function returns a nonzero return code, the minidisk pool identified through a previous IDENTIFY request, if any, remains in effect.

MAPMDISK REMOVE



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function removes a mapping between a range of pages in an address space and a set of 4K blocks residing in the minidisk pool.

The current contents of the pages in the specified range are discarded, the pages are set to binary zeros, and are considered to be unmapped pages. If host-page protection was applied to the page by the DEFINE function (because the page was mapped onto a read-only minidisk), that host-page protection is removed. MAPMDISK REMOVE must be issued before exiting guest applications that use MAPMDISK; otherwise, unpredictable results might occur.

Parameters

label
is an optional assembler language label on the macro.

ASIT=
specifies the real address of an 8-byte field in storage that contains the ASIT identifying the address space for which mappings are to be removed. Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

If this ASIT is not associated with an address space your virtual machine created, or with your virtual machine's host-primary address space, then return code 8 is given.

This operand is required.

PAGEADDR=
specifies the real address of a 4-byte field in storage that contains the 31-bit absolute address of the first page in the range of pages to be unmapped. Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range 2-12, inclusive) containing the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The range of pages to be unmapped, from PAGEADDR to PAGEADDR+(PAGECT-1)*4096, must be contained within the bounds of the address space designated by the ASIT operand, otherwise an addressing exception is recognized. In addition, this range must not include any unmapped pages, otherwise return code 12 is given.

Bits 20-31 of the page address must be zeros, otherwise a specification exception is recognized.

This operand is required.

PAGECT=
specifies the real address of a 4-byte field in storage that contains the count of pages for which a mapping is to be removed. Specify this parameter as a label associated with the field in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The minimum number of pages to be unmapped is 1, and the maximum number is 524,288. If this count is not in this range, then return code 4 is given.

This operand is required.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this parameter as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the MAPMDISK DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. The macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. Code the DEFWORKA macro at the end of your program to define any macro work areas that have not yet been defined.
3. Removing a mapping using the REMOVE function changes the data that is visible in the mapped page. However, it does not change the storage key associated with the page.

Condition Codes and Return Codes

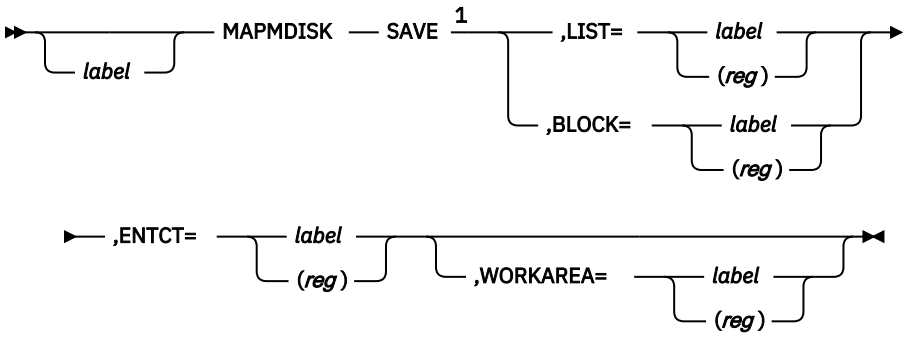
On return from the REMOVE function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	REMOVE function status
0	The mapping has been removed from the specified range of pages.
4	The page count specified by the PAGECT operand is out of range. No mappings have been removed.
8	The ASIT specified by the ASIT operand does not identify either an existing address space that your virtual machine created, or the host-primary address space for your virtual machine. No mappings have been removed.

Return Code in Register 15 (Decimal)	REMOVE function status
12	The range of pages to be unmapped includes an unmapped page. Mappings have been removed for all pages up to but not including the first unmapped page in the range. Register 0 contains the address of the page for which the error occurred. This page resides in the address space specified by the ASIT operand.

If an addressing exception is recognized on the REMOVE function, the operation is terminated. Those pages in the page range identified by the PAGEADDR and PAGECT parameter that are processed before the point of error have mappings removed as usual. Those pages in the page range at or after the point of error remain unchanged.

MAPMDISK SAVE



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

The SAVE function initiates an asynchronous operation that causes the contents of changed, mapped pages to be stored on the associated 4K minidisk blocks. This function provides a mechanism to commit changes made to those pages in a predictable fashion.

Your virtual machine must have read-write access to the address space containing the mapped pages.

The pages to be saved are specified in a save-list definition block. The size of the save-list definition block cannot exceed 4096 bytes. A block may be in the form of a list of address ranges or a list of individual pages, but not both. Normally, a given page is specified only once per SAVE request. However, it is not an error for a particular page to be specified multiple times in a single SAVE request, or multiple times in the same save-list definition block.

For each page to be saved by the SAVE request, a check is made to determine if the page has been changed since the last time a store operation (either as a result of the SAVE function, or performed automatically by CP) was performed on the page. If the page has changed, a store operation is initiated to save the contents of the page on the associated 4K minidisk block; unchanged pages are skipped. If a page is changed after the SAVE function is invoked but before completion of the SAVE function is signaled, it is unpredictable whether the changed data is stored as a result of the in-progress SAVE function. If the changed data is not stored, change integrity is maintained so a subsequent SAVE function request will cause the change data to be stored.

The order in which the pages are processed by the SAVE function is unpredictable.

Asynchronous completion of the SAVE operation is indicated by a save-completion external interruption being presented to the virtual machine. Upon the external interruption, a completion code is stored to indicate the success or failure of the SAVE operation. See Notification of SAVE Completion for details on the save-completion external interruption.

Multiple SAVE functions can be concurrently outstanding; each is distinguished by a user supplied token. The token is supplied on the save-completion external interruption. Uniqueness of the token is not enforced, but if the tokens are not unique your program may have difficulty determining which save operation completed.

Parameters

label
is an optional assembler language label on the macro.

LIST=
specifies the real address of a save-list-definition block for this request which is in ***list form***. Specify this parameter as a label associated with the block in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the block in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the block.

The address of the first save-list-definition block must be an address on a doubleword boundary, otherwise a specification exception is recognized. See Save-List Format for more information.

This parameter is mutually exclusive with BLOCK= and one of them must be specified.

BLOCK=
specifies the real address of a save-list-definition block for this request which is in ***block form***. Specify this parameter as a label associated with the block in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the block in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the block.

The address of the first save-list-definition block must be an address on a doubleword boundary, otherwise a specification exception is recognized. See Save-List Format for more information.

This parameter is mutually exclusive with LIST= and one of them must be specified.

ENTCT=
specifies the real address of a 4-byte field in storage that contains the total count of entries in the save-list definition block. Specify this parameter as the label associated with the field in storage or as the number of a register (in the range of 2-12, inclusive) contains the address of the field in storage. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The minimum number of entries is one for both forms of the macro. For the ***list form*** the maximum number of entries is 1018 and for the ***block form*** the maximum number of entries is 509. If the above counts are out of bounds then a return code of 4 is given.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this parameter as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the MAPMDISK DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. The macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. When an error completion code is stored on external interruption code X'2603', subcode X'01', the error buffer contains the address of one of the pages for which an error was found. However, because errors may have occurred on pages other than the one identified by the address in the error buffer, and since the SAVE function processes pages in an unpredictable order, when an error completion code is stored on external interruption code X'2603', subcode X'01' it is unpredictable as to what pages have or have not been saved.

3. After an error completion code is received, the following recovery procedure can be used to bypass the error on the failing page (identified by the address in the error buffer) and complete the save process on the remaining pages:
   a. Move the data in the failing page to some other page which is mapped to some other 4K minidisk block
   b. Modify the save list to remove the address of the failing page and add the address of the target page from the preceding step
   c. Reissue the SAVE request.

   Pages that were not successfully saved during the first SAVE request are saved by this second SAVE request (unless there are more errors) while pages that were saved by the first request are skipped.

   However, the above procedure is not sufficient to eliminate the possibility of future storage-degradation machine checks on the failing page. See Programming Note “3” on page 826 for a procedure to use to eliminate the possibility of future storage-degradation machine checks on the same page.
4. The error buffer must be in writeable storage, otherwise a protection exception is recognized.

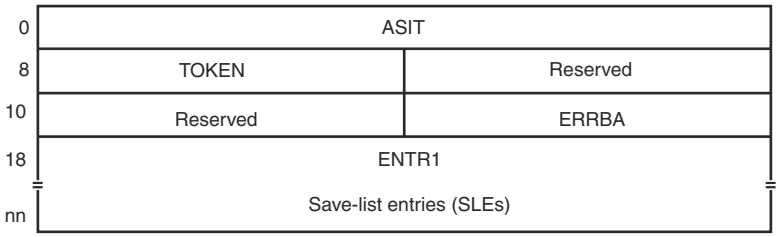
Save-List Format: The pages to be saved are specified in a save-list definition block. The block contains a header, and a contiguous table of save-list entries. The block must be aligned on a doubleword boundary and cannot exceed 4096 bytes in length. The save-list entries for a particular save-list definition block are either in the format of single pages or ranges of contiguous pages:

- For single pages, the absolute address of each page to be saved is specified by a word in the list.
- For ranges of contiguous pages, each range is specified by a pair or words in the list, the first word of a pair specifies the absolute address of the start of the range and the second word of a pair specifies the number of contiguous pages to be saved.

The two types of entries cannot be mixed within a save-list definition block and the type being used for a particular block is specified on the MAPMDISK macro.

Note: Field names used in the following descriptions are for reference only. You may copy these names, or use any other names, in your own code.

A save-list definition has the following format:



ASIT
Bytes 0-7 of the save-list definition block contains the ASIT identifying the address space in which pages will be saved. If this ASIT is not associated with an address space your virtual machine created, or the host-primary address space for your virtual machine, then return code 8 is given.

TOKEN
Bytes 8-11 of the save-list definition block contains a 32 bit token to be associated with this save request. It is the program's responsibility to maintain uniqueness to distinguish different SAVE functions. The token is returned with external interrupt code X'2603', subcode X'01'.

Reserved
Bytes 12-19 of the save-list definition block are reserved for future use and should contain binary zeros.

ERRBA
Bytes 20-23 of the save-list definition block contain the 31-bit absolute address of an 8-byte error buffer within the host-primary address space. This field must specify a nonzero address on a

doubleword boundary, otherwise a specification exception is recognized. The error buffer is always in the host-primary address space, even when the virtual machine is in the access-register mode.

Information is provided in the error buffer when the SAVE operation completes with certain errors. See Notification of SAVE Completion for details.

ENTR1

Starting at byte 24 of the save-list definition block is a contiguous table of save-list entries. Each save-list entry is either a:

- 4-byte word that contains the 31-bit absolute page address of an individual page to be saved.
- 8-bytes that consists of two words, the first specifying the 31-bit absolute page address of the start of a range of pages and the second specifying the count of contiguous pages to be saved.

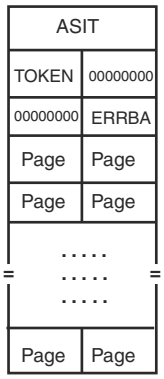
Each entry in a save-list definition block must be of the same format, either a 4-byte individual page address or an 8-byte page range specification. The two formats cannot be mixed within the same save-list definition block. The type of format being used for a save-list entry is specified on the MAPMDISK macro.

When page ranges are specified, the total number of pages contained within the supplied ranges must be at least 1 and must not exceed 524,288. If the total number of pages contained within the specified ranges is not between these values, then return code 16 is given.

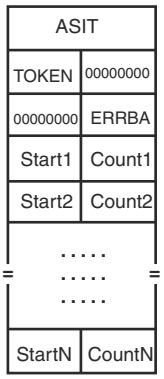
The range of pages to be mapped, from the start address to the start address+(count-1)*4096, must be contained within the bounds of the address space designated by the ASIT in the ASIT field, otherwise an addressing exception is recognized. This also applies to the individual pages that are found outside of the same bounds.

The different structures for each type of save-list definition block follows:

Single page entries save-list definition block
List form



Range of page entries save-list definition block
Block form



Condition Codes and Return Codes

On return from the SAVE function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	SAVE function status
0	The save request has been successfully initiated. When the asynchronous request completes, the completion status will be reflected through a external interrupt code X'2603', subcode X'01'.
4	The ENTCT count is either zero or above a maximum of 1018 or 509 for a list of individual or range of pages, respectively. Save processing was not performed on any page.

Return Code in Register 15 (Decimal)	SAVE function status
8	The ASIT specified by the ASIT operand does not identify either an existing address space that your virtual machine created, or your virtual machine's host-primary address space. Save processing was not performed on any page.
16	A MAPMDISK SAVE function was invoked specifying a save-list definition block in range form, and a count of 0 consecutive pages was detected. A count was greater than 524,288. The total count of all count fields is greater than 524,288. Save processing was not performed on any page. Register 0 contains the address within the save-list definition block containing the invalid count. For the case were the total count is greater register 0 contains the address of the entry that caused the overflow. In both cases in the access-register mode, access register 0 contains the ALET specifying the address space in which the save-list definition block resides.

If an addressing exception is recognized on the SAVE function, the operation is terminated. None of the pages have been processed.

Completion Conditions

Notification of SAVE Completion: When a MAPMDISK SAVE operation has completed, the virtual machine that initiated the SAVE operation is notified of the completion by a save-completion external interruption. The save-completion interruption is indicated by an external interruption code of X'2603' and a subcode of X'01'; the subcode of X'2603' is stored at location 132 on the external interruption.

The save-completion interruption is a floating interruption condition that is presented to any virtual CPU in the configuration that is enabled for the condition. The external interruption mask (bit 7 of the PSW) and subclass-mask bit 22 of control register 0 determine whether a virtual CPU is enabled for external interruption code X'2603'.

When the save-completion external interruption is presented, data is stored at real storage locations 128-133 as follows:

Location
Data stored on interruption

128-131
The 32 bit token specified for the SAVE operation.

132
External-interruption X'2603' subcode. A value of X'01' denotes a save-completion notification.

133
Bit 0 indicates whether the program-supplied error buffer contains valid data. If bit 0 is one, then the error buffer does not contain valid data. This could be due to the page being read-only or the error-buffer address being no longer addressable. If bit 0 is zero, then for completion codes X'04', through X'0C', the program-provided error buffer contains additional data related to the error.

Bits 1-7 are a completion code indicating the completion status of the MAPMDISK SAVE operation, as follows:

Code
Completion status

X'00'
All pages were saved without errors.

X'04'
An attempt was made to save at least one unmapped page. All mapped pages have been saved without error. If bit 0 of byte 133 is zero, the address of one of the unmapped pages has been stored in bytes 4-7 of the program-provided error buffer.

X'08'
The address space containing the pages to be saved no longer exists. None of the pages have been saved. If bit 0 of byte 133 is zero, the ASIT identifying the address spaces has been stored in bytes 0-7 of the program-provided error buffer.

X'0C'
Paging I/O errors were detected while saving one or more pages. It is unpredictable as to whether any pages were saved successfully. If bit 0 of byte 133 is zero, the address of one of the pages for which an error was found has been stored in bytes 4-7 of the program-provided error buffer.

If both X'04' and X'0C' conditions are detected during save processing, completion code X'0C' is presented on the save-completion notification.

PFAULT Macro -- Page-Fault Handshaking Services

CP manages the address spaces provided to virtual machines. This includes the host primary address space (base space) and host data spaces created by XC-mode virtual machines. While managing real machine storage, CP determines what portions of virtual machine storage are allowed to be resident at any given time. If a virtual machine references a portion of an address space that is not resident in real storage, CP receives a page-fault indication. CP's usual course of action is to suspend the operation of the virtual machine, perform the paging operation, and finally resume execution of the virtual machine. The page fault and its resolution by CP are transparent to the virtual machine, except for the time delay during which the virtual machine was suspended.

It is possible that for some applications, such as servers, the delay due to synchronous page-fault resolution can be a problem. If the server implements some form of multitasking to maximize its throughput, it would want to use the delay in executing one task, for instance to wait for a host-paging I/O operation to complete, to work on another. It can do this by using support for asynchronous page-fault handling.

When a page fault occurs during a storage reference and asynchronous resolution is enabled, CP initiates the paging operation and then immediately resumes execution of the virtual machine, presenting a page-fault initiation interruption to the virtual machine as a signal that a page fault has occurred. The multitasking server in the virtual machine can then suspend its current task (the one that just encountered the page fault) and select some other task to run. The new task runs in parallel with the CP paging operation to resolve the original task's page fault.

Upon completion of the paging operation, CP presents a page-fault completion interrupt to the virtual machine. This interruption informs the server that the task that generated the page fault can now be resumed since the page it requires is now available in storage.

To use asynchronous page-fault handling, the application must do some preparation and some additional processing using CP and possibly some CMS services. To activate and be able to use page-fault notification, the application must:

1. Establish an external interrupt handler for the page fault initiation and completion signals.
2. Issue the PFAULT macro to define a TOKEN address to CP.[5] This TOKEN address is the location of an area in storage that the server maintains as the task ID token of the current (running) task. The server would typically set this word to the address of the task control block for the current task. When a page fault occurs, CP obtains the token for the current task from this location and supplies it in the page-fault initiation and completion notifications for the page fault just encountered. Because this token is usually the address of the task control block, the server can process the page-fault completion interrupt without having to search for the correct task control block.
3. Call the ADD function of the ALSERV macro with the FAULTS=ASYNCH operand when establishing addressability to the VM data spaces address space. Asynchronous page-fault handling is enabled independently for each entry in a virtual machine's access list. Thus, the server can use asynchronous page-fault resolution for some types of references, but not for others.[6]

Asynchronous page-fault handling is available for references that occur in AR mode for nonprimary address spaces and in primary-space mode or AR mode to the primary space. Asynchronous page-fault handling is typically used only for page faults that require CP to perform I/O operations to resolve. If I/O operations are not required, then the time required to perform the page fault is so short that no significant benefit can be derived by using asynchronous handling considering the overhead involved.

[5] In some limited circumstances it may be necessary to invoke a DIAGNOSE X'258' directly rather than with the preferred PFAULT macro interface.

[6] The primary address space does not require any additional enablement.

Purpose

Use the PFAULT macro to enable or cancel page-fault handshaking for a virtual CPU. Page-fault handshaking allows a virtual CPU to continue execution during the time CP is resolving a host page fault. Page-fault handshaking is particularly useful for multitasking applications that can normally find some other task to run during the time that a task must be suspended while CP resolves a host page fault for it.

The following functions can be invoked with this macro:

CANCEL
Cancel page-fault handshaking for a virtual CPU

DECLARE
Define the macro work area

TOKEN
Activate page-fault handshaking for a virtual CPU and establish the location of the page-fault handshaking token. When the VERSION=2 parameter is specified with the TOKEN function, host-primary address spaces are also supported.

Usage Notes

If the VERSION=2 parameter is **not** specified with PFAULT TOKEN, then your virtual machine must be an XC virtual machine to successfully use this macro. If your virtual machine is an ESA virtual machine, then a specification exception is recognized unless VERSION=2 is specified. If the VERSION=2 parameter is specified with PFAULT TOKEN, then your virtual machine can be in any machine mode and architecture mode supported by CP.

Page-Fault Handshaking: If the VERSION=2 parameter is **not** specified with PFAULT TOKEN, then page-fault handshaking is applied to selected storage references made when in the access-register mode. If the VERSION=2 parameter is specified with PFAULT TOKEN, then handshaking is applied in all modes supported by CP, but storage references made while in the access-register mode still require FAULTS=ASYNCH on the appropriate ALSERV ADD macro. The handshaking process consists of the presentation of a pair of page-fault handshaking external interruptions to the virtual machine: the first interruption provides a page-fault initiation signal, and the second provides a page-fault completion signal.

Before page-fault handshaking actions can be performed for a virtual CPU, handshaking must be enabled for the virtual CPU through the TOKEN function of this macro. If the VERSION=2 parameter is **not** specified, the TOKEN function specifies the (guest) real address of a four byte area in the host-primary address space (the token location) that contains the page-fault handshaking token. If the VERSION=2 parameter is specified, the TOKEN function specifies the guest real address of an eight byte area if ARCHITECTURE=z is specified or a four byte area if ARCHITECTURE=ESA is specified. The current contents of this location (the current token) are provided in the page-fault initiation signal that indicates the start of a page-fault handshaking action, and the same value is provided in the corresponding page-fault completion signal.

The token is intended to identify the virtual machine task associated with the page fault in order to suspend that task upon page-fault initiation and to make that task eligible to run upon page-fault completion. Management of the token, including assigning token values, maintaining the appropriate relationships between the token and the current virtual machine task, and so forth is the responsibility of the program running in the virtual machine. It is expected that the token location is fixed for a virtual CPU. That is, the program should change the contents of the token location, rather than the address of the token location, whenever switching tasks to be executed by the virtual CPU.

Programming Note: The page-fault handshaking token is 32 bits long, or 64 bits long if the VERSION=2 with ARCHITECTURE=z parameters are specified, so that the program can use the address of the task control block as the handshaking token for the task. Assigning the token in this way allows the program to suspend and resume the task without the need for searches to find the task indicated by a page-fault initiation or page-fault completion signal.

Conditions necessary for handshaking: Page-fault handshaking is usually applied to a storage reference that will encounter a significant delay due to a page-fault condition, if all of the following conditions are met:

- The virtual CPU making the storage reference is enabled for page-fault handshaking external interruptions. (See Notification of Page-Fault Handshaking.)
- The virtual CPU has activated page-fault handshaking. A virtual CPU activates page-fault handshaking by using the TOKEN function of this macro.
- If the VERSION=2 parameter was **not** specified with PFAULT TOKEN, then the virtual CPU is in the access-register mode, and the storage reference is through an access-list entry (ALE) that is eligible for page-fault handshaking. An ALE is made eligible for page-fault handshaking by specifying FAULTS=ASYNCH on the ALSERV ADD operation that establishes the ALE.

  When the VERSION=2 parameter was specified, if the virtual machine is in XC mode and the virtual CPU is in access-register mode, then access-register-specified storage references are eligible only if the corresponding ALE has been made eligible for page-fault handshaking using FAULTS=ASYNCH on the ALSERV ADD macro. Host-primary storage references require no additional authorization.
- If VERSION=2, the bits that are on in the selection mask must be in the same state in the PSW for the virtual CPU as they are in the compare mask.

If any of these conditions are not met, then page-fault handshaking is not applied to the storage reference. The virtual CPU is suspended by CP until resolution of the page-fault condition is completed. When the resolution of the page-fault condition is completed, execution of the virtual CPU is resumed without any indication of the delay.

Under unusual circumstances, page-fault handshaking may be bypassed for a storage reference even though all of the above conditions are met.

Handshaking process: When a virtual CPU encounters a host page fault for which page-fault handshaking is to be applied, the following occurs:

1. The instruction that encountered the host page fault is nullified.
2. CP fetches four bytes, or eight bytes if VERSION=2 is specified with ARCHITECTURE=z, from storage at the token location in the host-primary address space.
3. CP presents a page-fault initiation signal to the virtual machine through a page-fault handshaking external interruption. The page-fault handshaking token fetched from the token location is provided in this initiation signal.
4. The page-fault initiation signal is presented to the virtual CPU that encountered the page-fault condition.
5. Normally, the virtual machine responds to the initiation signal by suspending the currently running task and switching to some other task.

Subsequently, when CP finishes resolution of the page-fault condition, it presents a page-fault completion signal to the virtual machine through a page-fault handshaking external interruption. The page-fault handshaking token provided in this completion signal is the same as that provided in the corresponding page-fault initiation signal for this host page fault. The page-fault completion signal may be presented to any virtual CPU in the configuration that is enabled for the interruption, even those which have not activated page-fault handshaking (that is, did not execute the TOKEN function of this macro).

Normally, the virtual machine responds to the completion signal by enabling the task identified by the page-fault handshaking token to run again.

Notification of page-fault handshaking: The page-fault initiation and page-fault completion signals described above are provided through page-fault handshaking external interruptions. A page-fault handshaking interruption is indicated by an external interruption code of X'2603' and a subcode of X'02' or X'06'; the subcode of X'2603' is stored at location 132 on the external interruption.

When the page-fault handshaking interruption is for a page-fault initiation signal, the interruption is presented to the virtual CPU encountering the host page fault condition. When the page-fault handshaking interruption is for a page-fault completion signal, it is a floating interruption condition that is

presented to any virtual CPU in the configuration that is enabled for the condition. The external interruption mask (bit 7 of the PSW) and subclass mask bit 22 of control register 0 determine whether a virtual CPU is enabled for external interruption code X'2603' (for either initiation or completion signals).

When the page-fault handshaking external interruption is presented and either the VERSION=2 parameter of PFAULT TOKEN was not in effect, or the VERSION=2 and ARCHITECTURE=ESA parameters of PFAULT TOKEN were specified, then data is stored at real storage locations 128–133 as follows:

Location
Data stored on interruption

128-131
The 32-bit page-fault handshaking token retrieved from the token location at the time that page-fault handshaking action was initiated for the page fault. The same value is provided in both the page-fault initiation and page-fault completion signals for a given instance of page-fault handshaking action.

132
External interruption X'2603' subcode. A value of X'02' denotes a page-fault handshaking external interruption.

133
Bit 0 indicates whether the interruption is for a page-fault initiation or a page-fault completion signal. If bit 0 is zero, then the interruption is for a page-fault initiation signal. If bit 0 is one, then the interruption is for a page-fault completion signal.

The contents of bits 1-7 are unpredictable.

When the page-fault handshaking external interruption is presented and the VERSION=2 and ARCHITECTURE=z parameters of PFAULT TOKEN were in effect, then data is stored at real storage locations as follows:

Location
Data stored on interruption

4536-4543
The 64-bit page-fault handshaking token retrieved from the token area at the time that page-fault handshaking action was initiated for the page fault. The same value is provided in both the page-fault initiation and page-fault completion interruptions for a given page fault.

132
External interruption X'2603' subcode. In z/Architecture mode, a value of X'06' denotes a page-fault handshaking external interruption for the VERSION=2 parameter of PFAULT TOKEN.

133
Bit 0 indicates whether the interruption is for a page-fault initiation or page-fault completion signal. If bit 0 is zero, then the interruption is for a page-fault initiation signal. If bit 0 is one, then the interruption is for a page-fault completion signal.

The contents of bits 1-7 are unpredictable.

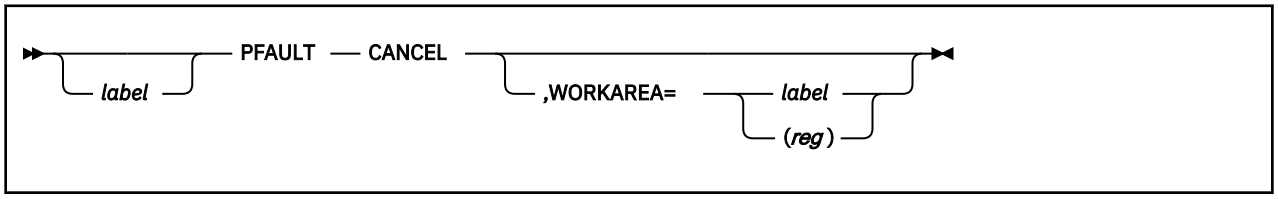
Program Exceptions

The PFAULT macro may result in one of the following program exceptions:

Problem Encountered	Cause
Access exception (See page “Access Exceptions” on page 8.)	An error occurred trying to • Fetch or store the macro parameter list (in macro work area) • Fetch a macro operand • Fetch from the token location specified by the ADDR operand (TOKEN function); key-controlled protection does not apply

Problem Encountered	Cause
Specification exception	• If the VERSION=2 parameter of PFAULT TOKEN was not in effect, your virtual machine is not an XC virtual machine. • The macro work area is not aligned on a doubleword boundary. • The parameter list generated by the macro is in error.

PFAULT CANCEL



Purpose

This function cancels page-fault handshaking for the virtual CPU that invokes the function.

The CANCEL function purges all in-progress page-fault handshaking actions that were initiated as a result of page faults encountered by the invoking virtual CPU. A page-fault handshaking action is considered to be in-progress if the page-fault initiation signal has been made pending, but the corresponding page-fault completion signal has not yet been made pending. If the page-fault completion signal has been made pending on the invoking or other virtual CPU, it remains pending.

Parameters

label
is an optional assembler language label on the macro.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range 2-12 inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the PFAULT DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding the use of this operand.

This operand is optional; you can simplify coding of your program by omitting this operand in order to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. In a multiple virtual CPU environment, the CANCEL function cancels page-fault handshaking only for the virtual CPU on which it is invoked. Other virtual CPUs that have activated page-fault handshaking continue to be eligible for page-fault handshaking.

Condition Codes and Return Codes

On return from the CANCEL function, register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	CANCEL function status
0	Page-fault handshaking is cancelled for the virtual CPU.
4	Page-fault handshaking has not been activated for the virtual CPU or has been previously canceled.

PFAULT DECLARE

Purpose

This function defines the storage required for the WORKAREA operand of the PFAULT macro. You must code this function only if you need to define the macro work area outside of the program requiring it. To simplify using the PFAULT macro, do not code the PFAULT DECLARE function and omit the WORKAREA operand on the executable functions of the PFAULT macro. This allows the PFAULT macro to control and define the necessary data expansion through the DEFWORKA macro coded at the end of your program. See “Coding CP Macros” on page 793 for additional details.

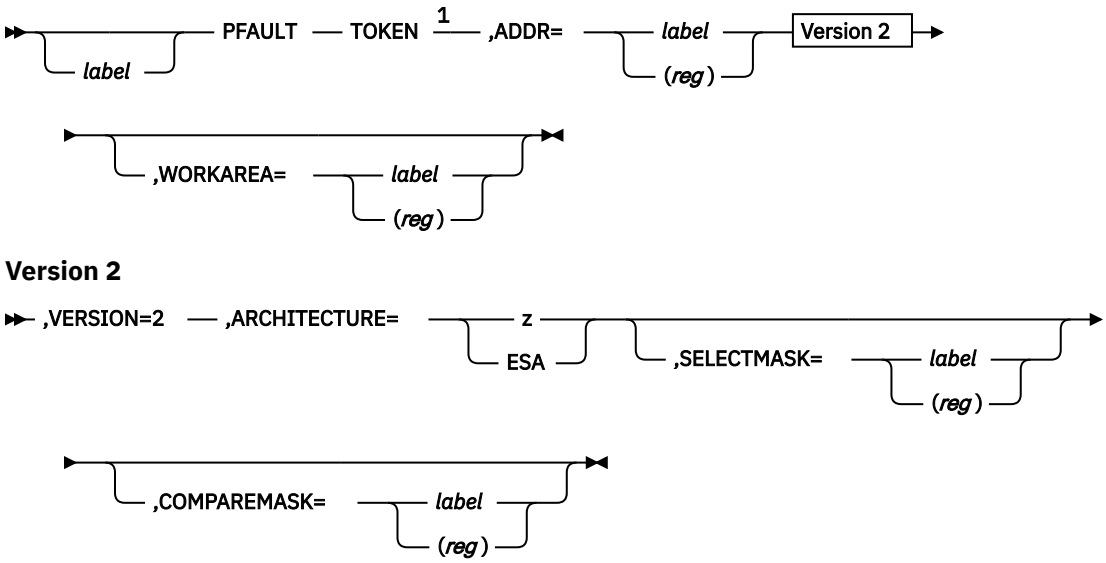
Parameters

label
is an optional assembler language label on the macro to be assigned to the defined storage.

Usage Notes

1. The DECLARE function does not generate any executable code. You may code it within a dummy section (DSECT) or a control section (CSECT).
2. Although a label is not required on the invocation of the DECLARE function, it may be necessary to identify the work area on the WORKAREA operand.

PFAULT TOKEN



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

This function activates page-fault handshaking for the virtual CPU that invokes the function and establishes the location of the page-fault handshaking token for that virtual CPU. It can also establish the selection and compare masks for that virtual CPU.

Once activated, page-fault handshaking remains in effect with the same page-fault handshaking token location until handshaking is cancelled using one of the following methods:

- The PFAULT CANCEL macro is issued on the virtual CPU.
- A CPU-reset operation is performed on the virtual CPU.
- A SIGP Set-Architecture instruction is performed by the virtual configuration.

Parameters

label
is an optional assembler language label on the macro.

ADDR=
Specifies the real address where the token is located in the host-primary address space. Specify this operand as a label associated with where the token is located in real storage, or as the number of a register (in the range 2-12, inclusive) containing the real address of the token's location in real storage.

The token is a 4-byte fullword aligned field or an 8-byte doubleword aligned field when the VERSION=2 and ARCHITECTURE=z parameters are specified.

Generation of the effective address of the token location, including address wraparound, is performed at the time of the invocation of this function. The resulting effective address is stored in the parameter list and used for all subsequent page-fault handshaking actions. That is, the addressing mode in effect at the time of a page-fault initiation or page-fault completion signal has no effect on the addressing of the token location.

This operand is required.

VERSION=2
Enables page-fault handshaking for both 31–bit and 64–bit guests and for primary address space page faults.

ARCHITECTURE=
Specifies the architecture to be used by the macro expansion. When ARCHITECTURE=z is specified, z/Architecture instructions and 64-bit registers are used in the macro expansion. When ARCHITECTURE=ESA is specified, ESA/390 instructions and 32-bit registers are used in the macro expansion. This parameter also determines the length and alignment of the token area.

Note: When specifying the 'z' operand, an upper or lower case character can be used.

SELECTMASK=
Specifies the real address of the 64-bit selection mask. Specify this operand as a label associated with the 8-byte mask in real storage, or as the number of a register (in the range 2-12, inclusive) containing the real address of the location of the selection mask in real storage.

At the time a page-fault occurs that is eligible for page-fault handshaking, if a bit in the selection mask is zero, it causes the corresponding bit in the virtual CPU PSW at the time of the page fault to be ignored. If a bit in the selection mask is one, it causes the corresponding bit in that PSW to be selected for comparison against the corresponding bit in the compare mask.

In z/Architecture mode, all 64 bits of the mask are significant. When not in z/Architecture mode, only bits 0-32 are significant.

Generation of the effective address of the location of the selection mask, including address wraparound, is performed at the time of the invocation of this macro. The resulting effective address is used to copy an 8-byte field to the parameter area.

If this operand is not specified, a mask of all zeros is the default.

COMPAREMASK=
Specifies the real address of the 64-bit compare mask. Specify this operand as a label associated with the 8-byte mask in real storage, or as the number of a register (in the range 2-12, inclusive) containing the real address of the location of the compare mask in real storage.

At the time a page-fault occurs that is eligible for page-fault handshaking, if a bit in the selection mask is zero, it causes the corresponding bit in the virtual CPU PSW at the time of the page fault to be ignored. If a bit in the selection mask is one, it causes the corresponding bit in that PSW to be selected for comparison against the corresponding bit in this compare mask.

Generation of the effective address of the location of the compare mask, including address wraparound, is performed at the time of the invocation of this macro. The resulting effective address is used to copy an 8-byte field to the parameter area.

If this operand is not specified, a mask of all zeros is the default.

WORKAREA=
specifies the address of a real storage area that is used by the macro as a work area. Specify this operand as the label associated with the storage area, or as the number of a register (in the range 2-12 inclusive) containing the address of the storage area. For a virtual machine in access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the PFAULT DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding the use of this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

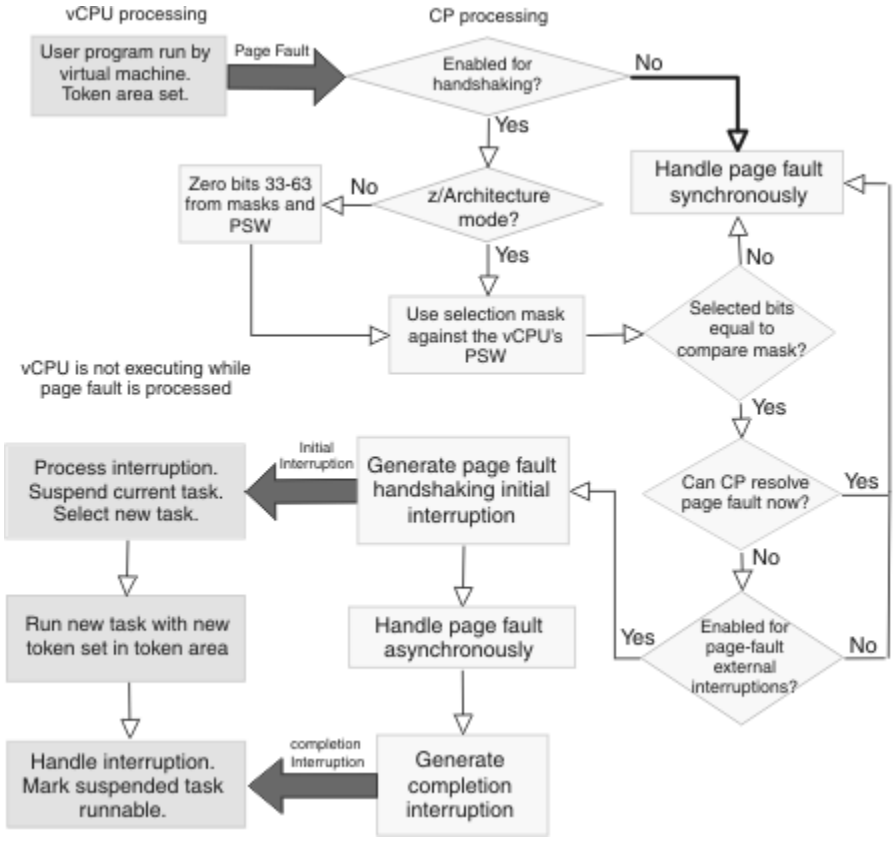


Figure 98. Page Fault Processing when VERSION=2

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. CP does not enforce any restriction on the contents of the page-fault handshaking token. Uniqueness of the page-fault handshaking token and maintenance of the established token location (that is, the token's relationship to a virtual machine task) is the responsibility of the virtual machine application.
3. The location of the page-fault handshaking token for a virtual CPU cannot be changed without first cancelling page-fault handshaking for the virtual CPU. To cancel page-fault handshaking for a virtual CPU, invoke the CANCEL function of this macro. A CPU reset operation also cancels page-fault handshaking.
4. Page-fault handshaking is most effective when the page containing the page-fault handshaking token is resident in host storage. Defining the token within (guest) real page zero achieves this effect because real page zero for a virtual CPU is always resident when the virtual CPU is running. Alternatively, the token can be defined outside of real page zero; the frequent references that are made to that page to maintain the handshaking token may be sufficient to keep the page resident in host storage. However, if the token is defined outside of real page zero, you may want to consider locking that page in host storage to guarantee residency.

   If, at the time of a handshaking-eligible host page-fault, the page containing the page-fault handshaking token is not resident in host storage then it is unpredictable whether page-fault handshaking is applied to the page fault. Even if it is applied, the virtual CPU is delayed until CP completes page-in of the page containing the token or of the faulting page, whichever occurs sooner.
5. The PFAULT macro activates page-fault handshaking and establishes the location of a page-fault handshaking token only for the virtual CPU on which the PFAULT macro is invoked. In a multiple virtual CPU environment, activation of page-fault handshaking requires invocation of the PFAULT macro by each virtual CPU for which handshaking is desired.

   Normally, each virtual CPU has a different absolute storage location for the page-fault handshaking token. Because the address of the page-fault handshaking token is a (guest) real address, this can be

achieved by defining the token at the same address within real page zero for each virtual CPU. The prefixing process transforms the real address into different (guest) absolute addresses for each virtual CPU.

6. If page fault handshaking is in effect and the virtual configuration executes a SIGP Set-Architecture instruction, page-fault handshaking is reset.
7. Registers 2-12 inclusive may be used for the ADDR, SELECTMASK, and COMPAREMASK operands, but the value specified for ADDR must be unique. The mask parameters may specify the same register.
8. If a bit in the compare mask is set to 1, the corresponding bit in the selection mask must also be set to 1, or a specification exception is returned.
9. If the selection mask and the compare mask both have a 1 in bit position 15, then the program determines that the PSW must be in the problem state for an initial interruption to be presented. If the selection mask and the compare mask are both all zeros, then all page faults pass the selection test.
10. The ARCHITECTURE parameter only indicates how the macro expands. The mode of the virtual CPU at the time the macro executes determines the size of the token area and the location in the prefix area where the interruption parameter is stored. This must match the ARCHITECTURE parameter setting or a specification exception is returned.
11. If the mode of the virtual CPU is z/Architecture mode when this macro is executed with VERSION=2, then ARCHITECTURE=z must be specified. The token area is 8 bytes and the interruption parameter is 8 bytes, which are stored in locations 4536-4543; the subcode is X'06.'

    When this macro is not executed in z/Architecture mode, either VERSION=2 cannot be specified, or VERSION=2 can be specified with ARCHITECTURE=ESA. In these cases, the token area is 4 bytes and the interruption parameter is 4 bytes, which are stored in locations 128-131; the subcode is X'02.'
12. If the token area is not accessible, a program check is presented to the virtual CPU.
13. Registers 0, 1, 14 and 15 are changed by this macro. When ARCHITECTURE=z is specified, all 64 bits of these registers may be affected.

Condition Codes and Return Codes

On return from the TOKEN function, register 15 contains one of the following 32 bit return codes:

Return Code in Register 15 (Decimal)	TOKEN function status
0	Page-fault handshaking is activated for the virtual CPU.
8	Page-fault handshaking has already been activated for the virtual CPU. The established page-fault handshaking token location is unchanged.

REFPAGE — Page Reference Services

Purpose

Use the REFPAGE macro to temporarily identify an ordered set of pages that your program will reference over a relatively short amount of time. Such an ordered set of pages is known as a page-reference pattern. Identifying your program's short-term page-reference pattern allows CP to perform some pre-fetching of pages, that is, retrieving pages into processor storage in advance of your program's need for them. This reduces the number of page faults your program encounters, and thus improves the performance of the system.

CP performs pre-fetching by using the page-reference pattern information to form logical blocks of pages. These logical blocks are sets of pages that your program will require at about the same time. When your program encounters a page fault for the first of the pages in a logical block, CP may initiate page-in of the other pages in the logical block as well. However, the logical blocks formed from the page-reference pattern are temporary. Once pages identified in a page-reference pattern have been brought into processor storage for referencing, the logical blocking created from the page-reference pattern is lost.

The following page-reference services can be invoked using the REFPAGE macro:

DECLARE
Define the macro work area.

INFORMB
Identify a regular page-reference pattern using the ***block form***.

INFORML
Identify a complex page-reference pattern using the ***list form***.

Usage Notes

To successfully use any of the functions of this macro, you must be an XC virtual machine. If not, an specification exception is recognized for an ESA virtual machine.

Specifying Page-Reference Patterns: The REFPAGE macro provides two ways to indicate your program's page-reference pattern. They are the ***block form*** and the ***list form***.

Block form is used when pages will be referenced in either ascending or descending address order and there is a regular pattern in the addresses of the pages to be referenced. The REFPAGE macro uses the concept of groups, subgroups, spans and skips to define such a regular page-reference pattern. They are defined as:

Group
is a page-reference pattern that consists of one or more sets of pages called *subgroups*.

Subgroup
is the portion of the block-form reference pattern that is treated as a single logical block when retrieving pages into processor storage. Depending on the storage demand on the z/VM system, page-in of all of the pages in the subgroup into processor storage may be initiated at the time of the first reference to a page in the subgroup.

A subgroup consists of one or more sets of contiguous pages to be referenced, called *spans*, possibly separated by sets of contiguous pages that will not be referenced, called *skips*.

Span
is a set of contiguous pages that will be referenced in sequential order.

Skip
is a set of contiguous pages, between spans, that will not be referenced.

See Examples for various examples of block-form page-reference patterns.

List form is the other way to indicate your program's page-reference pattern. It is used when there is no regular pattern in the addresses of the pages your program will reference but nonetheless the set of pages about to be referenced is known in advance. For example, your program may soon reference a set of pages determined from a lookup in an index. This set of references will not occur in strictly ascending or descending page address order, nor is there some regular pattern that relates one page in the pattern to the next in the pattern. An example of a non-regular reference pattern that might result is:

```
Reference     Page address            Reference     Page address
   1st          00020000                6th          02BC3000
   2nd          0000E000                7th          00021000
   3rd          00DE2000                8th          00022000
   4th          01E00000                9th          00E00000
   5th          0000A000               10th          00023000
```

The list form of the REFPAGE macro allows the identification of this complex reference pattern by specifying the page addresses for each of the pages to be referenced, in the order in which the references will be made.

Both forms of the REFPAGE macro construct logical blocks of pages. These blocks are constructed of pages that are currently paged-out (reside on paging DASD, not in processor storage) and can span 1MB segment boundaries, but not address space boundaries. Logical blocks allow CP to bring a whole or partial block into processor storage when the virtual machine references a page in the block. However, it does not guarantee any pre-paging activity on behalf of the virtual machine. Pre-paging is dependent on system resource availability.

Program Exceptions

The REFPAGE macro may result in one of the following program exceptions:

Problem Encountered	Cause
Access exception (See page “Access Exceptions” on page 8.)	An error occurred trying to • Fetch or store the macro parameter list (in macro work area) • Fetch a macro operand • Fetch the page-reference list (INFORML function) • Access (as if a fetch) a page in a block-form reference pattern (INFORMB function); key-controlled protection does not apply; access-exception conditions other than addressing are reported as return codes • Access (as if a fetch) a page specified in the page-reference list (INFORML function); key-controlled protection does not apply; access-exception conditions other than addressing are reported as return codes
Specification exception	• Your virtual machine is an ESA virtual machine. • The macro work area is not aligned on a doubleword boundary. • The parameter list generated by the macro is in error. • The page-reference list is not aligned on a word boundary.

REFPAGE DECLARE

Purpose

This function defines the storage required for the WORKAREA operand of the REFPAGE macro. You must code this function only if you need to define the macro work area outside of the program requiring it. To simplify using the REFPAGE macro, do not code the REFPAGE DECLARE function and omit the WORKAREA operand on the executable functions of the REFPAGE macro. This allows the REFPAGE macro to control and define the necessary data expansion through the DEFWORKA macro coded at the end of your program. See “Coding CP Macros” on page 793 for additional details.

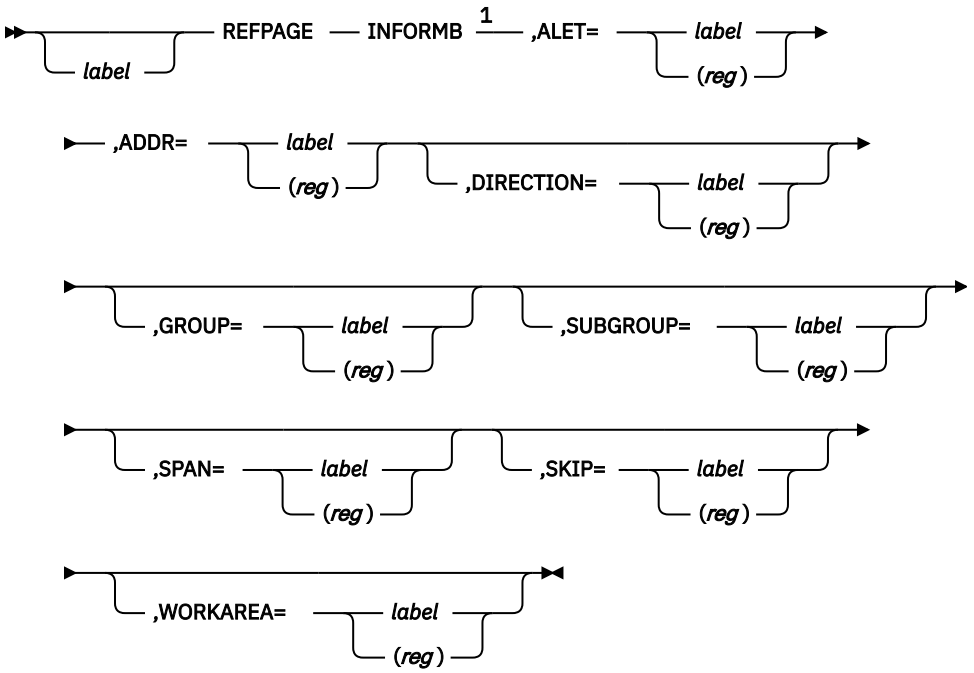
Parameters

label
is an optional assembler language label on the macro to be assigned to the defined storage.

Usage Notes

1. The DECLARE function does not generate any executable code. You may code it within a dummy section (DSECT) or a control section (CSECT).
2. Although a label is not required on the invocation of the DECLARE function, it may be necessary to identify the work area on the WORKAREA operand.

REFPAGE INFORMB



Notes:

[1] The keyword parameters that follow can be specified in any order.

Purpose

The INFORMB function identifies an ordered reference pattern in which pages will be referenced in either strictly ascending or descending address sequence and the pages to be referenced form a repeating pattern or are contiguous to each other.

Parameters

label
is an optional assembler language label on the macro.

ALET=
specifies the real address of a 4-byte field in storage that contains the access-list-entry-token (ALET) identifying the address space for which the page-reference pattern is being defined. Specify this parameter as a label associated with the field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The ALET contained in the field must designate a private address space that your virtual machine owns, otherwise return code 8 is given. This ALET is always used to identify the target address space; that is, this ALET is used even when the virtual machine is in the primary-space mode.

This operand is required.

ADDR=
specifies the real address of a 4-byte field in storage that contains the 31-bit starting address in the page-reference pattern. Specify this parameter as a label associated with the field, or as the number

of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The starting address is used as the lowest address in the page-reference pattern when the DIRECTION operand indicates *ascending* (the default), or is the highest address in the page-reference pattern when the DIRECTION operand indicates *descending*.

This operand is required.

DIRECTION=
specifies the real address of a 1-byte field in storage that contains an indication of the direction of the page-reference pattern. Specify this parameter as a label associated with the field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

A value of X'01' (positive one, binary) in the field indicates an *ascending* page-reference pattern while a value of X'FF' (negative one, binary) indicates a *descending* page-reference pattern. When an ascending page-reference pattern is indicated, the starting address specified by the ADDR operand is the lowest address of the page-reference pattern. When a descending page-reference pattern is indicated, the starting address specified by the ADDR operand is the highest address of the page-reference pattern.

There is no validity checking for values other than X'01' and X'FF' in the field. Results are unpredictable if an invalid value is specified.

This operand is optional; the default is an ascending page-reference pattern.

GROUP=
specifies the real address of a 4-byte field in storage that contains the number of sub-groups in the page-reference pattern. Each subgroup constitutes a logical block of pages that is brought into processor storage as a unit. Specify this parameter as a label associated with the 4-byte field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The default and minimum value for this parameter is 1 and the maximum is 262,144. If the value is not within this range, then return code 20 is given.

Note: If GROUP operand is 1 or omitted then there will be only one logical block of pages formed. When referencing a large number of pages, there may be places in the page-reference pattern that may have long periods of computation before continuing to referencing the remainder. In these cases, a value greater than 1 should be used to allow breaking the page-reference pattern into subgroups. This allows only those pages in the subgroup that are referenced to be brought into processor storage.

This operand is optional; the default value is 1.

SUBGROUP=
specifies the real address of a 4-byte field in storage that contains the number of spans contained in each subgroup of the page-reference pattern. Specify this parameter as a label associated with the field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

A subgroup is a logical block of pages. Subject to the storage demands on the z/VM system, CP attempts to bring all of the pages in a subgroup into processor storage when the first page in the subgroup is referenced.

The default and minimum value allowed for SUBGROUP is 1 and the maximum is 262,144. If the value is not within this range, then return code 20 is given.

If SUBGROUP operand is omitted or 1 then the SPAN operand specifies the subgroup size if the GROUP operand is greater than 1.

This operand is optional; the default value is 1.

SPAN=
specifies the real address of a 4-byte field in storage that contains the number of contiguous pages that will be referenced sequentially. Specify this parameter as a label associated with the field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The default and minimum value allowed for SPAN is 1 and the maximum is 262,144. If the value is not within this range, then return code 20 is given.

Note: If the operands SPAN and SUBGROUP are either omitted or 1, then there will not be any blocking of pages.

This operand is optional; the default value is 1.

SKIP=
specifies the real address of a 4-byte field in storage that contains the number of pages that will be skipped from the last page of a given span to the first page of the next span. Sequences of skipped pages are not brought into processor storage when referencing the subgroup. Specify this parameter as a label associated with the 4-byte field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The default and minimum value for SKIP is 0 and the maximum is 131,072. If the value is not in this range, then return code 20 is given.

Note: If SKIP is zero then all pages specified for keywords SPAN and SUBGROUP are contiguous.

This operand is optional; the default value is 0.

WORKAREA=
specifies the real address of a storage area that is used by the macro as a work area. Specify this parameter as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the REFPAGE macro DECLARE function. See “Coding CP Macros” on page 793 for additional detail regarding the use of this operand.

This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. To indicate a contiguous span of pages in one logical page block set GROUP and SUBGROUP to 1 and SKIP to 0 and set SPAN to the number of contiguous pages.
3. If all the operands use their default value, there will not be any page blocking.

4. Multiple REFPAGE macro INFORMB functions can be concurrently active. If the same page(s) are specified in more then one INFORMB or INFORML request then the logical blocks of pages that result will be interleaved. The last INFORM function will be the only one with its logical block of pages intact. The other logically blocked pages are partially intact. There could be reduced expected performance depending on how the logical blocks are referenced.
5. Logical blocks are formed from pages that reside on DASD. If the pages are located in host storage, they will not be blocked.
6. CP's normal page stealing forms blocks of pages similar to those formed by REFPAGE. It is possible for a block of pages that was formed by normal page stealing to point into a REFPAGE-formed block. If a page from the steal-formed block is referenced then it is possible that part of the REFPAGE-formed block will be brought into storage.
7. The value of the keywords when multiplied together minus the value of SKIP should not exceed the maximum value of 524,288:

   (GROUP * SUBGROUP * (SPAN + SKIP)) - (SKIP) =< 524,288
8. Large numbers can cause a virtual machine to have performance problems when issuing this macro.

Condition Codes and Return Codes

On return from REFPAGE INFORMB, register 15 will contain one of the following return codes:

Return Code in Register 15 (Decimal)	REFPAGE macro INFORMB function status:
0	The notification of the pending order of page references has been accepted and processed.
8	The ALET operand does not identify either an existing private address space that your virtual machine created, or your virtual machine's base address space. An address space (including the base) must be private when it is used with this function. The page-reference pattern has not been established.
12	A shared page was detected in the page-reference pattern.
20	A value was specified for either GROUP, SUBGROUP, SPAN, or SKIP that was out of range. The page-reference pattern has not been established.

Examples

Example 1: This example shows how to define a logical block of pages to be brought into storage. This will allow the program to have pages *P1,P2,P3,P6,P7,P8,P11,P12,P13,P16,P17,P18* brought in with one page fault when *P1* is referenced.

```
         LA    2,STARTLOW
         ...
         REFPAGE INFORMB,ADDR=(2),GROUP=L1,SUBGROUP=L4,SPAN=L3,SKIP=L2
         ...
L1       DC    F'1'
L2       DC    F'2'
L3       DC    F'3'
L4       DC    F'4'
STARTLOW DC    X'00012000'  (Addr of P1)
```

This defines a page-reference pattern as follows:

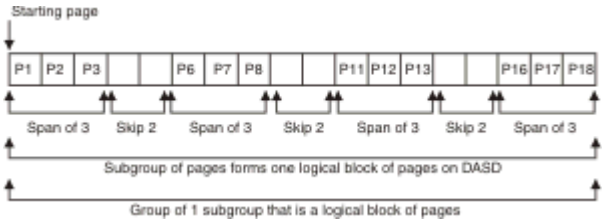


Figure 99. Group of 1 Subgroup That Has 4 Spans of Pages

Example 2: This example shows how to define two logical blocks of pages to be brought into storage separately. This will allow the program to have pages *P1,P2,P3,P6,P7,P8* brought into storage for one page fault. The second subgroup will not be brought in until *P11* is referenced.

```
         LA    2,STARTLOW
         ...
         REFPAGE INFORMB,ADDR=(2),GROUP=L2,SUBGROUP=L2,SPAN=L3,SKIP=L2
         ...
L2       DC    F'2'
L3       DC    F'3'
STARTLOW DC    X'00012000'  (Addr of P1)
```

defines a page-reference pattern as follows:

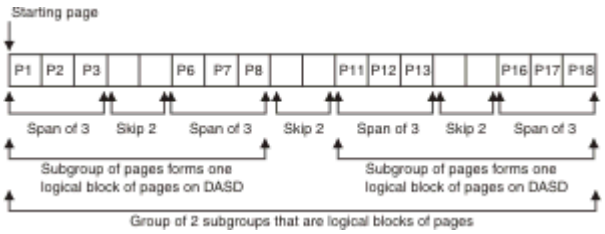


Figure 100. Group of 2 Subgroups That Have 2 Spans of Pages in Each Subgroup

Example 3: This example shows how to define two logical blocks of pages to be brought into storage separately in reverse order. This will allow the program to have pages *P18,P17,P16,P13,P12,P11* brought into storage for one page fault. The second subgroup will not be brought in until *P8* is referenced.

```
         LA    2,STARTHIG
         ...
         REFPAGE INFORMB,ADDR=(2),GROUP=L2,SUBGROUP=L2,SPAN=L3,
                SKIP=L2,DIRECTION=DESCEND
         ...
L2       DC    F'2'
L3       DC    F'3'
STARTHIG DC    X'01F12000'  (Addr of P18)
DESCEND  DC    X'FF'
```

defines a page-reference pattern as follows:

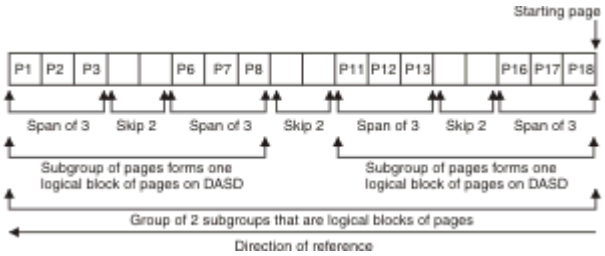


Figure 101. Group of 2 Subgroups That Have 2 Spans of Pages in Each Subgroup

Example 4:

```
         LA    2,STARTLOW
         ...
         REFPAGE INFORMB,ADDR=(2),SPAN=L18
         ...
```

```
L18      DC    F'18'
STARTLOW DC    X'00012000'  (Addr of P1)
```

defines a page-reference pattern that has 18 contiguous pages in one logical block of pages.

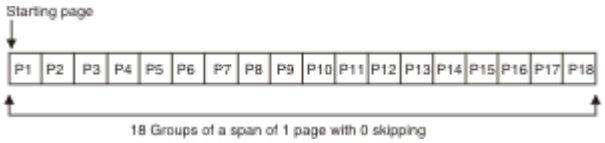


Figure 102. Group of a Span of 18 Pages in One Logical Block

Example 5:

```
         LA    2,STARTHIG
         ...
         REFPAGE INFORMB,ADDR=(2),SPAN=L18,DIRECTION=DESCEND
         ...
L18      DC    F'18'
STARTHIG DC    X'01F12000'  (Addr of P18)
DESCEND  DC    X'FF'
```

defines a page-reference pattern that has 18 contiguous pages in one logical block of pages. The keyword DIRECTION indicates that keyword ADDR is the highest address.

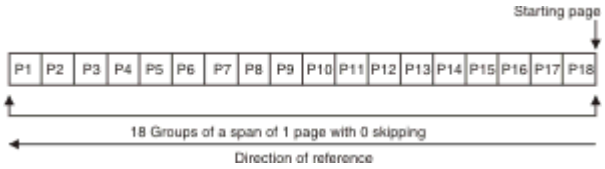
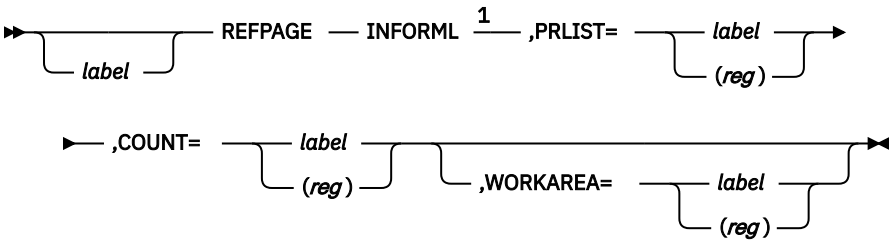


Figure 103. Group of a Span of 18 Pages in One Logical Block

where each page is to be considered an individual group with a span of one without any skips.

REFPAGE INFORML



Notes:

[1] The keyword operands that follow can be specified in any order.

Purpose

The INFORML function identifies an ordered reference pattern in which there is no regular pattern or relationship among the addresses of the pages to be referenced. Instead, the page reference pattern is identified by a list of pages in the reference pattern.

Parameters

label
is an optional assembler language label on the macro.

PRLIST=
specifies the real address of the page-reference list. Specify this parameter as a label associated with the page-reference list in storage, or as the number of a register (in the range of 2-12, inclusive) containing the address of the page-reference list. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the page-reference list.

The page-reference list specifies the absolute addresses of pages to be included in the page-reference pattern. (See Page-Reference List Format for more information on the page-reference list.) The page-reference list is processed in the order it is specified to form one or more logical blocks of pages. Pages specified by consecutive entries in the list are placed the same logical block as long as the ALETs for the entries are the same. If consecutive entries have different ALETs, the current logical block is ended and a new logical block is started. Thus, if the list contains entries with differing ALETs then multiple logical blocks are formed. Logical blocks are formed within an address space not across address spaces.

This operand is required.

COUNT=
specifies the real address of a 4-byte field in storage that contains the count of entries in the page-reference list specified by the PRLIST operand. Specify this parameter as a label associated with the field, or as the number of a register (in the range of 2-12, inclusive) containing the address of the field. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the field.

The minimum number of entries is two and the maximum number of entries is 16,384. If the count is not in this range, then return code 4 is given.

This operand is required.

WORKAREA=

specifies the real address of a storage area that is used by the macro as a work area. Specify this parameter as the label associated with the storage area, or as the number of a register (in the range of 2-12, inclusive) containing the address of the storage area. For a virtual machine in the access-register mode, the access register corresponding to the base register selected by the assembler for the label (label form) or the access register associated with the specified general register (register form) is used to determine the address space containing the storage area.

The storage area is defined by the DEFWORKA macro coded at the end of your program or through the REFPAGE DECLARE function. See “Coding CP Macros” on page 793 for additional details regarding this operand.

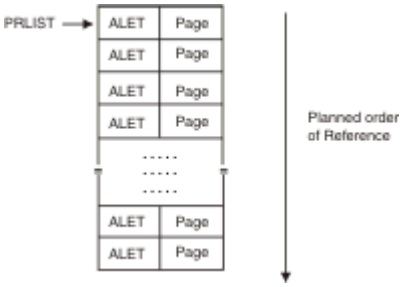
This operand is optional; you can simplify coding of your program by omitting this operand to defer the definition of the macro work area to the DEFWORKA macro.

Usage Notes

1. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
2. The order of the entries in this list defines the order each page is planned to be referenced by the caller. If the reference order does not match their ordering in the list, performance improvements may not be realized.
3. Multiple REFPAGE macro INFORML functions can be concurrently active. If the same page(s) are specified in more than one INFORMB or INFORML request then the logical blocks of pages that result will be interleaved. The last INFORM function will be the only one with its logical block of pages intact. The other logically blocked pages are partially intact. There could be reduced expected performance depending on how the logical blocks are referenced.
4. The number of entries of 16,384 should be used with caution. Large numbers can cause a virtual machine performance problems when issuing this macro.

Page-Reference List Format: The list pointed to by PRLIST is a set of word pairs where each pair is an ALET and 31-bit absolute page address. Each ALET must designate a private address space that your virtual machine owns, otherwise return code 8 is given. Logical blocks are formed from each consecutive identical ALET within the list. If there are different ALETs then there will be multiple logical blocks formed. If an ALET appears twice in the list with an intervening different ALET a second logical block will be formed for the same address space.

Thus, the list of entries appears as follows:



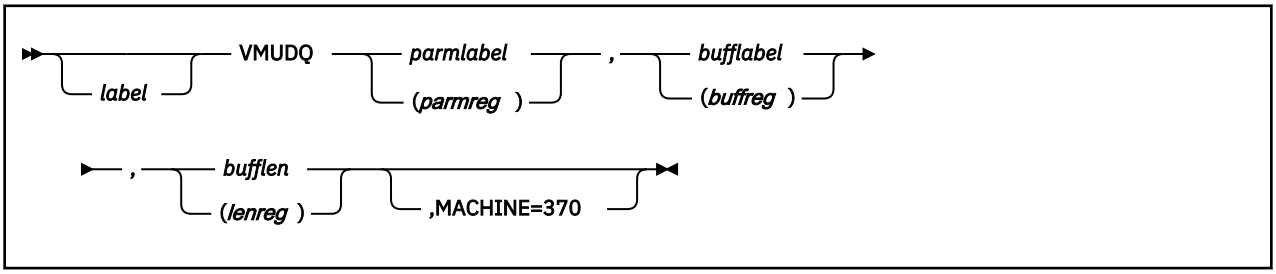
The ALETs contained in the page-reference list are always used to identify the target address spaces; that is, they are used even when the virtual machine is in the primary-space mode.

Condition Codes and Return Codes

On return from REFPAGE INFORML register 15 contains one of the following return codes:

Return Code in Register 15 (Decimal)	REFPAGE macro INFORML function status:
0	The notification of the pending order of page references has been accepted and processed.
4	The value for keyword COUNT is either less than 2 or greater than the maximum of 16,384. This request was not processed.
8	An ALET specified within the page-reference list does not identify either an existing private address space that your virtual machine created, or your virtual machine's host-primary address space. An address space (including the base) must be private when it is used with this function. The list of pending page references may have been formed up to the ALET in error.
12	A shared page was detected in page-reference list.

VMUDQ – VM User Directory Query



Purpose

The VMUDQ (VM User Directory Query) macro provides an application interface for making specified queries against the CP User Directory. Documented here is the LSTMDISK (List MDISK Definitions) function, which writes selected MDISK definitions from the user directory to a designated buffer.

Note: IBM privilege class B is needed.

Parameters

label
is an optional assembler language label.

parmlabel
(*parmreg*)
specifies the address of a function parameter list held within an address space. Specify this operand as a label associated with the storage area, or as the number of a general register that is part of an address/ALET pair of the storage area.

If it is specified as the number of a general register, this register is assumed to be already loaded with the address, and the corresponding access register is assumed to be already loaded with the ALET of the target address space. If the ALET value is 0, the target address space is the user's primary address. If the virtual machine is not an XC virtual machine, the ALET value is ignored, and the target address space is the user's host-primary address space.

The parameter list is maximum of six doublewords long, and is described under MDISK Parameter List Contents. It must be on a doubleword boundary and must not span a page boundary. Otherwise, a specification exception is recognized.

This operand is required.

bufflabel
(*buffreg*)
specifies the address of the data buffer held within an address space. Specify this operand as the label associated with the storage area, or as the number of a register that is part of an address/ALET pair of the storage area. If it is specified as the number of a general register, this is assumed to be already loaded with the address, and the corresponding access register is assumed to be already loaded with ALET of the target address space. If the ALET value is 0, the target address space is the user's host-primary address space. If the virtual machine is not an XC virtual machine, the ALET value is ignored, and the target address space is the user's host-primary address space.

This operand is required.

bufflen
(*lenreg*)
is the length in bytes of the allocated data buffer.

The minimum value of the length required for the data buffer is 60 bytes, otherwise, a specification exception is recognized.

This operand is required.

MACHINE=370
has no supported function, because 370 virtual machines are not supported. Results are undefined if this operand is specified.

LSTMDISK Function: This function creates a list of MDISK definitions from the CP user directory based on, or qualified by, the:

- User IDs to which the MDISK definitions belong
- Virtual devices to which the MDISK definitions are defined
- Volumes containing volsers on which MDISK definitions are found
- SSI member system or SUBCONFIG ID for which the MDISK definitions are found.

An asterisk (*) can be used as a trailing wild card to select a wider group of definitions.

The criteria for this list are in a parameter list pointed to by the PARMADR of the macro. The list is built in the buffer defined in the macro.

MDISK Parameter List Contents: Figure 104 on page 876 shows the structure of the MDISK parameter list. It consists of six doublewords.

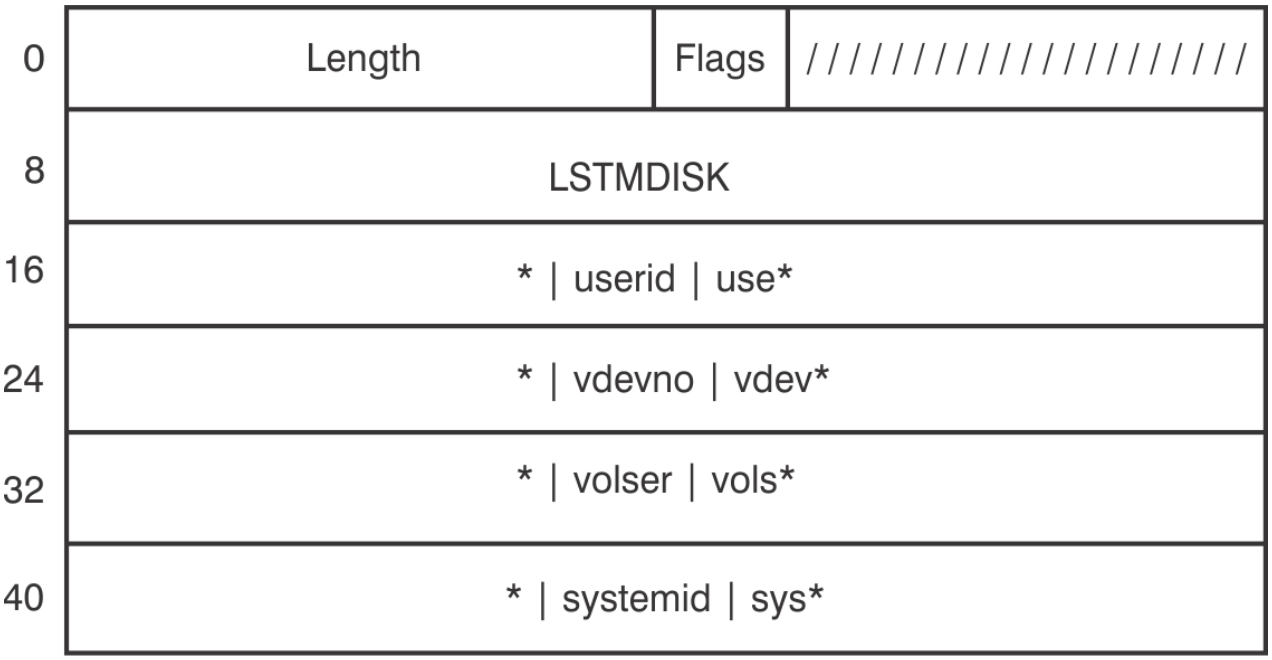


Figure 104. Contents of the LSTMDISK Function Parameter List

Length
The length is a fullword containing the length in bytes of the length field itself, the reserved field, and the parameter string that follows. The length field must be one of the following values: 16, 24, 32, 40, or 48; otherwise a specification exception is recognized.

Flags
contains flags as follows:

X'80'
Indicates that a SUBCONFIG ID should be included in the output record instead of a system name.

////////
is reserved for later use.

LSTMDISK
is the name of the function to be performed by the macro. It must be LSTMDISK; otherwise, a specification exception is recognized.

**** | userid | use****
selects MDISK definitions that belong to the specified user IDs. The *userid* from a USER or IDENTITY directory statement can be used. SUBCONFIG IDs cannot be specified. If you specify an asterisk (*) alone, all MDISK definitions are retrieved. If you specify an asterisk following a set of characters, all the MDISK definitions owned by all the user IDs that begin with those characters are retrieved. The value is left-justified in the field and padded to eight characters with blanks. If you leave the field blank, an asterisk is the default.

**** | vdevno | vdev****
selects MDISK definitions that are defined to the specified virtual device number(s). Multiple virtual device numbers are specified by using an asterisk (*). If you specify an asterisk (*) alone, all MDISK definitions are selected regardless of *vdevno*. If you specify an asterisk following a set of numbers, all MDISK definitions for all the virtual devices whose addresses begin with those numbers are retrieved. A four-digit address is assumed; therefore, 000* would mean 0000 through 000F and 04* would mean 0400 through 04FF, and so on. The value is left-justified in the field and padded to eight characters with blanks. If you leave the field blank, an asterisk is the default.

**** | volser | vols****
selects MDISK definitions that are found on the volume containing the specified *volser*. If you specify an asterisk (*) alone, all MDISK definitions are retrieved. If you specify an asterisk following a set of characters, all the MDISK definitions on the volumes with *volser*s that start with those characters are retrieved. If you leave the field is blank, an asterisk is the default. The value is left-justified in the field and padded to eight characters with blanks.

**** | systemid | sys****
selects MDISK definitions that are found on the specified SSI member system. An asterisk (*) can be used as a trailing wild card to select a range of systems that begin with a common set of characters. An asterisk (*) by itself indicates that the MDISK definitions on all member systems are to be analyzed. The value is left-justified in the field and padded to eight characters with blanks. If the field is blank, the default assumed is blank, indicating that only the MDISK definitions on the current system are to be analyzed. Specifying a system other than by an asterisk or a blank is allowed only when an SSI-enabled directory is in use.

Output Buffer Format: The output buffer consists of seven blank-delimited fields containing the information indicated in the following example:

```
OWNERID- VDEV VOLSER DEVTYPE- START----- SIZE------ SYSTEM
12345678 1234 123456 12345678 1234567890 1234567890 12345678
---------+---------+---------+---------+---------+---------+-------
User123  0191 ABC123 3390     0000000010 0000000200 SYSTEM1
User456  0191 CCC123 FB-512   0000000010 0000200000 SYSTEM1
User789  0222 CCC222 3380     0000000010 END        SYSTEM2
UserABC  0345 000345 3390     0000000000 0000000100
```

Notes:

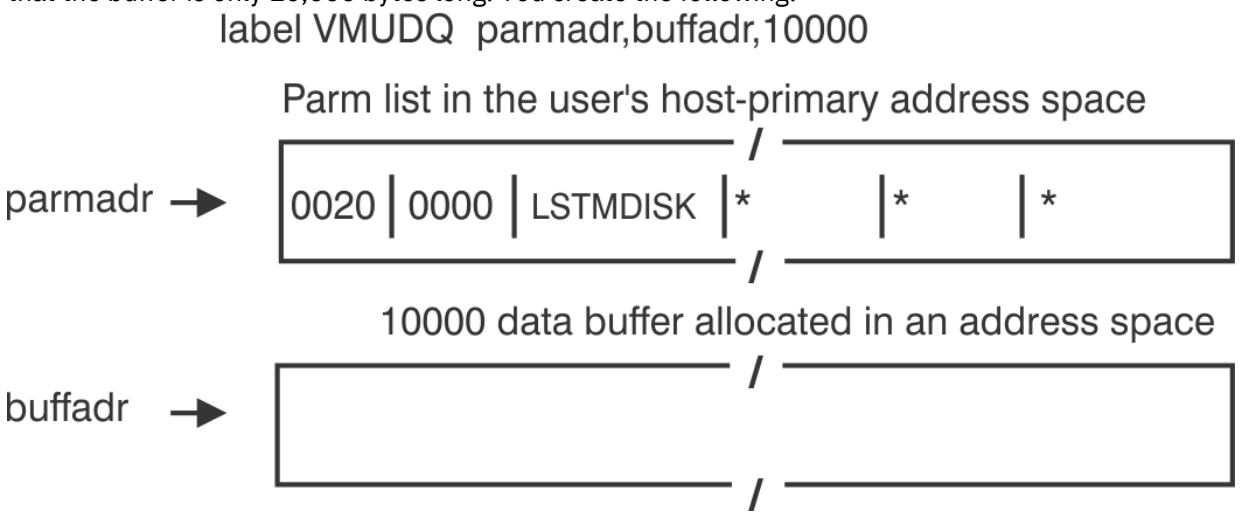
1. In the above example the "START" field may contain either the starting cylinder or block number. The "SIZE" field contains the remaining number of cylinders or blocks. The non-device specific "END" in the size field denotes the end of the volume. Also, the header lines are not part of the returned data.
2. In the above example, the first three lines of output include a system ID. A system ID indicates that the device was defined within a SUBCONFIG stanza and the device is local to a specified system. The last line of output does not include a system ID, indicating that the device was defined within an identity or user stanza and the device is global to all systems.

   If an SSI-enabled or SSI-ready directory is in use, a flag can be set in the Flags field of the parameter list to cause the SUBCONFIG ID that contains the MDISK definition to be returned instead of the system name.

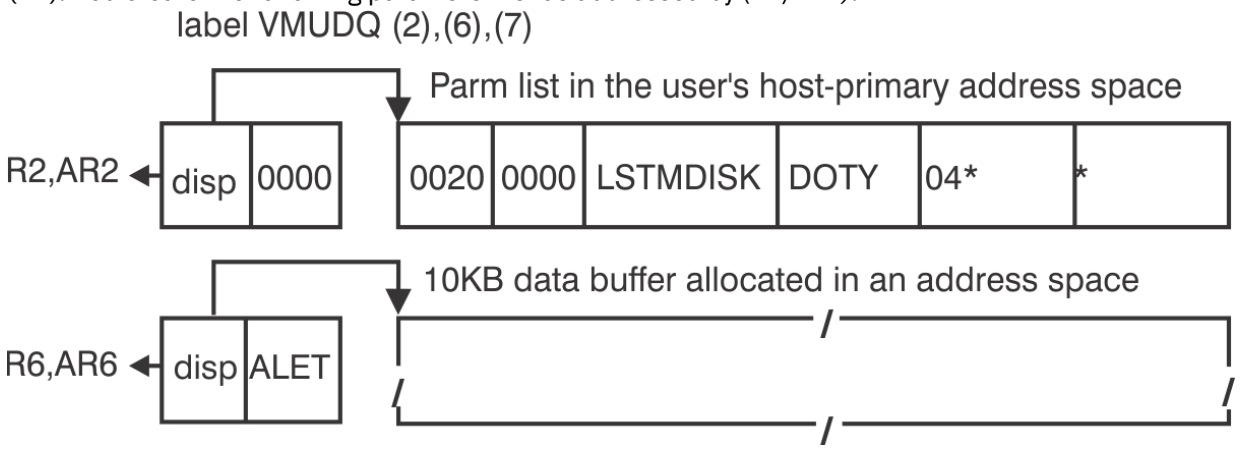
3. If the MDISK was defined with a size of "END", then size that will be returned in the output buffer will simply be "END".
4. Temporary disks (T-disks) and virtual disks in storage are ignored by the LSTMDISK function and are not included in the output buffer.

Examples

1. Assume you want to create a list of all MDISK definitions in the CP user directory for the system on which you are currently running and place them in a buffer addressed by *buffaddr*. Further assume that the buffer is only 10,000 bytes long. You create the following:



2. Assume you want to create a list of MDISK definitions belonging to user DOTY with addresses in the range of 0400 through 04FF, regardless of the volume on which they reside, and place them in a buffer addressed by *(R6/AR6)*, and further assume that the buffer is only 10000 bytes long as specified in (R7). You create the following parameter list as addressed by *(R2/AR2)*.



Usage Notes

1. An address space is acquired in blocks of 256 pages; therefore, the number of bytes in the buffer to be passed on the call would equal:

```
(number pages requested+255/256) * 1048576.
```

2. This macro modifies general and access registers 0, 1, 14, and 15. All others remain unchanged.
3. If *userid*, *volser*, or *systemid* contains an asterisk (*) not used as a trailing wild card, conflicting results can occur, because an asterisk is treated as a trailing wild, card left-justified, and padded to eight characters with blanks.

4. If the MDISK was defined with a volume serial number of &SYSRES, then the value in the VOLSER output column is the value specified for the &SYSRES option when DIRECTXA created the user directory. If this option was not specified, the value +VMRES is used.
5. In an SSI-enabled directory, MDISK definitions can be global or local. A global MDISK definition is one that is included in a user or identity stanza and the minidisk being defined can be linked by virtual machines on any member system of the SSI cluster. A local MDISK definition is one that is included in a SUBCONFIG stanza and the minidisk being defined can be linked only by virtual machines on the SSI member system to which the SUBCONFIG stanza applies. When specifying the system ID parameter (* | *systemid* | *sys**), you are requesting all MDISK definitions that are global and local to a particular system. If you specify a system ID that is not specified on any BUILD statements in the directory, then all global MDISK definitions are reported. If you specify a system ID that is specified on a BUILD statement in the directory, then all global MDISK definitions and local MDISK definitions for the specified system are reported.
6. If an SSI-enabled or SSI-ready directory is in use, a flag can be set in the Flags field of the parameter list to cause the SUBCONFIG ID that contains the MDISK definition to be returned instead of the system name.

Condition Codes and Return Codes

One of the following condition codes is returned:

Condition Code	Description
0	Normal exit from VMUDQ.
1	VMUDQ has detected an error.
3	VMUDQ has encountered a paging error while trying to access the target area in the data space or an object directory page.

On return from the LSTMDISK function, register 15 contains one of the following return codes if the condition code is 0.

Function	Return code in Register 15	Description
General	X'00'	Successful. Register 0 contains the number of bytes moved as the result of query.
	X'04'	No records were found to match the criteria specified.
	X'08'	The buffer length supplied is insufficient. Register 0 contains the number of bytes needed for the query.
LSTMDISK	X'100'	An invalid user ID was specified.
	X'104'	An invalid virtual device number was specified.
	X'108'	An invalid volume serial number was specified.
	X'10C'	An invalid system ID was specified.
	X'110'	The user directory does not include SUBCONFIG information but SUBCONFIG IDs were requested.

Program Exceptions

You may receive one of the following program checks if the VMUDQ input data is invalid.

Problem Encountered	Cause
Specification exception	Any of the following: • The input parameter list does not start at double-word boundary. • The length of buffer is less than 60 bytes, which is the minimum output buffer size. • The function specified is not LSTMDISK. • The input parameter list does cross a page boundary. • The length of the input parameter list is not one of the values 16, 24, 32, 40, or 48.
Access exception (See page "Access Exceptions" on page 8.)	An error occurred trying to: • Fetch the parameter list • Store into the data buffer.

Part 6. Architectural Extensions and Accommodations for Virtual Machines

This part contains the following chapters:

- Chapter 26, “Collaborative Memory Management Assist,” on page 883
- Chapter 27, “370 Accommodation Facility Overview,” on page 893
- Chapter 28, “370 Accommodation Facility Definition,” on page 901
- Chapter 29, “Store Hypervisor Information (STHYI) Instruction,” on page 913

© Copyright IBM Corp. 1991, 2020

Chapter 26. Collaborative Memory Management Assist

The collaborative memory management assist is a machine feature that allows z/Architecture guests with the appropriate support to exchange memory usage and status information with z/VM. This sharing of information provides several benefits:

- z/VM can make more effective decisions, leading to a more efficient use of available memory.
- z/VM page-write overhead is eliminated for pages that the guest has designated as unused or volatile.
- The guest can make better decisions when assigning pages.
- Guest page-clearing overhead can be eliminated for pages that z/VM has indicated contain zeros.
- z/VM can accommodate a proportionately larger number of guests for a given system paging load.

This chapter describes the EXTRACT AND SET STORAGE ATTRIBUTES instruction, the interface used by a guest to collaborate with z/VM, as well as associated changes to the virtual-machine interface. The material is an extension to the material in z/Architecture Principles of Operation (publibfi.boulder.ibm.com/epubs/pdf/dz9zr011.pdf), SA22-7832.

The collaborative memory management assist provides a means for communicating state information about a 4 KB block of storage between a program running in a z/Architecture virtual machine and the z/VM control program. This sharing of state information allows the program and z/VM to make more efficient memory management decisions. The collaborative memory management assist includes the following features:

- A unique block-usage state and block-content state that are associated with each 4 KB block of main storage (that is, memory) in the virtual configuration.
- The privileged EXTRACT AND SET STORAGE ATTRIBUTES instruction which can be used to extract and optionally set the block-usage and block-content states of a 4 KB block.
- A new reason for recognizing an addressing-exception program interruption.
- The new block-volatility-exception program-interruption condition.

Storage

The following is an extension to "Chapter 3. Storage", in *z/Architecture Principles of Operation*.

Collaborative Memory Management Block State

When the collaborative memory management assist is installed, a unique block-usage state and block-content state are associated with each 4 KB block of storage that is available in the virtual-machine configuration. Block-usage states and block-content states are not part of addressable storage. The current states of a 4 KB block are extracted and optionally set by EXTRACT AND SET STORAGE ATTRIBUTES.

Block-usage States

Following is a description of the block-usage states that can be assigned to a 4 KB block and their meanings:

- **Stable state.** A 4 KB block in the stable block-usage state can be referenced by the program. If z/VM reclaims the real-storage page frame in which the block resides, z/VM preserves (pages out) the block contents.
- **Unused state.** A 4 KB block in the unused state should not be referenced by the program; otherwise, an addressing exception might be recognized or unpredictable results might occur. At any time the

© Copyright IBM Corp. 1991, 2020

contents of a 4 KB block in the unused block-usage state can be discarded. When a block in the unused block-usage state is discarded, the block-content state of the block is set to the logically-zero state, the reference and change bits for the block are set to zeros, and the remainder of the storage key is set to unpredictable values. (For a description of the logically-zero state, see "Block-content States" on page 884.)

- **Volatile state.** A 4 KB block in the volatile block-usage state can be referenced by the program. However, at any time the contents of a 4 KB block in the volatile block-usage state can be discarded. When a block in the volatile block-usage state is discarded, the block-content state of the block is set to the logically-zero state, the reference and change bits for the block are set to zeros, and the remainder of the storage key is set to unpredictable values. (For a description of the logically-zero state, see "Block-content States" on page 884.)

  If the program references a 4 KB block that is in the volatile block-usage state and is discarded (i.e., the block is in the volatile block-usage state and the logically-zero block-content state), a block-volatility exception is recognized.
- **Potentially-volatile state.** A 4 KB block in the potentially-volatile block-usage state can be referenced by the program. Based on change indication, the block-usage state might change to the volatile or stable state as follows:

  If the change bit for the 4 KB block is one, the block-usage state might be changed to the stable state.

  If the change bit for the 4 KB block is zero, the contents of the block might be discarded. When a block in the potentially-volatile block-usage state is discarded, the block-usage state of the block is set to the volatile state, the block-content state of the block is set to the logically-zero state, the reference and change bits for the block are set to zeros, and the remainder of the storage key is set to unpredictable values. (For a description of the logically-zero state, see "Block-content States" on page 884).

Block-content States

Following is a description of the block-content states that can be assigned to a 4 KB block and their meanings:

- **Resident state.** The contents of a 4 KB block in the resident block-content state are present in z/VM real storage and immediately accessible by the program.
- **Preserved state.** The contents of a 4 KB block in the preserved block-content state are not present in z/VM real storage and are preserved elsewhere. When the program references a 4 KB block in the preserved block-content state, the contents of the block are restored and the block-content state of the block is changed to the resident state.
- **Logically-zero state.** The contents of a 4 KB block in the logically-zero block-content state are not present in z/VM real storage and are known to be zero. The logically-zero block-content state in conjunction with the block-usage state of the 4 KB block indicates whether the page is accessible by the program, as follows:

  If the block-usage state of the 4 KB block is the stable state and the block-content state of the block is the logically-zero state when the block is referenced by the program, the contents of the block are set to zeros and the block-content state is changed to the resident state.

  If the block-usage state of the 4 KB block is the unused state and the block-content state of the block is the logically-zero state when the block is referenced by the program, an addressing exception is recognized.

  If the block-usage state of the 4 KB block is the volatile state and the block-content state of the block is the logically-zero state when the block is referenced by the program, a block-volatility exception is recognized.

The program can also change the states associated with a page by using the EXTRACT AND SET STORAGE ATTRIBUTES instruction.

For 4 KB blocks that are in the stable block-usage state, the block-content state might change at any time from the resident to the preserved state or from the preserved to the resident state.

For 4 KB blocks that are in a block-usage state other than the stable state, the states associated with the block might change at any time as described under the block-usage states above.

For 4 KB blocks that are in the stable block-usage state and the logically-zero block-content state, the logically-zero state is removed when non-zero contents are stored in the block. This might occur at other times as well.

The collective term **discarded state** refers to either of two state combinations: the unused block-usage state and the logically-zero block-content state, or the volatile block-usage state and the logically-zero block-content state. Note that a discarded state need not arise from a discard action by z/VM. For example, an EXTRACT AND SET STORAGE ATTRIBUTES instruction performing a Set Unused operation on a stable logically-zero block or on a stable preserved block places that block into a discarded state.

References made by an entity other than the CPU (such as the channel subsystem) to storage in a 4 KB block that is in a discarded state are treated as if the block is not in the configuration. See "Implications for the DIAGNOSE Instruction and Non-CPU Accesses" on page 891.

As a result of the machine recognizing block-usage states, some combinations of block-usage and block-content states are not permissible. Thus, if the program executes EXTRACT AND SET STORAGE ATTRIBUTES specifying an impermissible state combination, the machine replaces the impermissible combination with a permissible combination. The table in Figure 105 on page 885 summarizes which combinations are permissible and which are not. The table also shows the state combinations (in parentheses) which replace the impermissible combinations.

Block-Content States

Block-Usage States	r	p	z
S	Y	Y	Y
U	Y	N(Uz)[1]	Y
V	Y	N(Vz)[2]	Y
P	Y	N(Sp)[3] (Vz)[3]	N(Vz)[4]

Legend:
r - Resident block-content state
p - Preserved block-content state
z - Logically-zero content state
S - Stable block-usage state
U - Unused block-usage state
V - Volatile block-usage state
P - Potentially-volatile block-usage state

Y - Yes - permissible
N - No - not permissible

Figure 105. Summary of permissible collaborative memory management state combinations

Footnotes on block/block-content state combinations shown in parentheses:

Notes:

1. Use of the EXTRACT AND SET STORAGE ATTRIBUTES instruction to set the block-usage state of a 4 KB block in the preserved block-content state to the unused state results in discarding the block by changing the block-usage state to the unused state and the block-content state to the logically-zero state.
2. Use of the EXTRACT AND SET STORAGE ATTRIBUTES instruction to set the block-usage state of a 4 KB block in the preserved block-content state to the volatile state results in discarding the block by

changing the block-usage state to the volatile state and the block-content state to the logically-zero state.

3. Use of the EXTRACT AND SET STORAGE ATTRIBUTES instruction to set the block-usage state of a 4 KB block in the preserved block-content state to the potentially-volatile state results in the block-usage state remaining the stable state and the block-content state remaining the preserved state, if the block has been changed, or results in discarding the block by changing the block-usage state to the volatile state and the block-content state to the logically-zero state, if the block has not been changed.
4. Use of the EXTRACT AND SET STORAGE ATTRIBUTES instruction to set the block-usage state of a block in the logically-zero block-content state to the potentially-volatile state results in changing the block-usage state to the volatile state, and the block-content state remains the logically-zero state.

Modification of Translation Tables

When the collaborative memory management assist is installed, translation tables should reside in 4 KB blocks that are in the stable block-usage state. The results of a translation that requires a translation-table entry that resides in a 4 KB block that is not in the stable block-usage state are unpredictable.

Assigned Storage Locations

The definitions of the assigned storage locations below are extended as indicated when the collaborative memory management assist is installed in a z/Architecture virtual machine:

160
(Real Address)

Exception Access Identification: During a program interruption due to a block-volatility exception, zeros might be stored into location 160.

162
(Real Address)

Operand Access Identification: During a program interruption due to a block-volatility exception recognized by the MOVE PAGE instruction, the contents of the R_1 field of the instruction are stored in bit positions 0-3 of location 162, and the contents of the R_2 field are stored in bit positions 4-7. If the block-volatility exception was recognized during the execution of an instruction other than MOVE PAGE, the contents of location 162 are unpredictable.

168-175
(Real Address)

Translation-Exception Identification: During a program interruption due to a block-volatility exception, bits 0-51 of the absolute address causing the exception are stored in bits 0-51 of locations 168-175. Bits 52-60 of locations 168-175 are unpredictable. If the exception was recognized during the execution of MOVE PAGE, bit 61 of locations 168-175 is set to one; otherwise, bit 61 is set to zero. Bits 62-63 of locations 168-175 are stored as zeros.

Control

The following is an extension to "Chapter 4. Control" in *z/Architecture Principles of Operation*.

Resets

The definitions of subsystem reset and clear reset are extended as indicated when the collaborative memory management assist is installed in a z/Architecture virtual machine:

Subsystem Reset

In addition to the standard operations performed by subsystem reset, if the collaborative memory management assist is installed, then the block-usage state of all 4 KB blocks is set to the stable state.

Clear Reset

In addition to the standard operations performed by clear reset, if the collaborative memory management assist is installed, the block-content state of all 4 KB blocks is set to the logically-zero state. Since a clear reset includes a subsystem reset, the block-usage state of all 4 KB blocks is also set to the stable state.

Interruptions

The following is an extension to "Chapter 6. Interruptions" in *z/Architecture Principles of Operation*.

Addressing Exception

z/Architecture defines a main storage location as not available in the configuration when the location is not installed, when the storage unit is not in the configuration, or when power is off in the storage unit. An address designating a storage location that is not available in the configuration is referred to as invalid. An addressing exception is generally recognized when the CPU attempts to reference a main storage location that is not available in the configuration.

When the collaborative memory management assist is installed, a main storage location is also considered not available in the configuration when the location is within a 4 KB block that is in the unused block-usage state and the logically-zero block-content state. However, addressing exceptions due to collaborative memory management block-usage and block-content states are not recognized for 4 KB blocks designated by the following instructions:

- EXTRACT AND SET STORAGE ATTRIBUTES
- INSERT STORAGE KEY EXTENDED
- SET STORAGE KEY EXTENDED
- RESET REFERENCE BIT EXTENDED

Block-volatility Exception

A block-volatility exception is recognized when the collaborative memory management assist is installed and the CPU attempts to reference a main storage location that is within a 4 KB block that is in the volatile block-usage state and for which the contents of the block have been discarded (as indicated by the referenced block also being in the logically-zero block-content state).

The unit of operation is nullified.

When an interruption occurs, information about the 4 KB block address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See “Assigned Storage Locations” on page 886 for a detailed description of this information.

Block-volatility exceptions are not recognized for 4 KB blocks that are designated by the following instructions:

- EXTRACT AND SET STORAGE ATTRIBUTES
- INSERT STORAGE KEY EXTENDED
- SET STORAGE KEY EXTENDED
- RESET REFERENCE BIT EXTENDED

When a block-volatility exception occurs during the fetching of an instruction or during the fetching of a DAT table entry associated with an instruction fetch, it is unpredictable whether the ILC is 1, 2, or 3. When the exception is associated with fetching the target of EXECUTE, the ILC is 2.

In all cases of block-volatility exceptions not associated with instruction fetching, the ILC is 1, 2, or 3, indicating the length of the instruction that caused the reference.

A block-volatility exception is indicated by a program-interruption code of 001A hex (or 009A hex if a concurrent PER event is indicated).

Access Exceptions

The access exceptions consist of those exceptions which can be encountered while using an absolute, instruction, logical, real, or virtual address to access storage. When the collaborative memory management assist is installed, the block-volatility exception is added to the list of these exceptions, in all translation and address-space-control modes.

A block-volatility exception for a particular access (instruction, operand, ART table, or DAT table) occurs at the same priority as an addressing exception for that access.

Control Instructions

The following is an extension to "Chapter 10. Control Instructions" in *z/Architecture Principles of Operation*.

Program Exceptions

The descriptions of general and control instructions typically indicate the cases in which access exceptions might be recognized. In these cases, when the collaborative memory management assist is installed, block-volatility exceptions are implicitly included among the access exceptions that might be recognized, except where indicated below.

Certain control instructions might list individual exceptions in their definition, rather than the overall term "access exceptions." In those cases, except as noted in the sections below, wherever addressing exception is listed as a program exception that an instruction might recognize, a block-volatility exception is also recognized when appropriate for that access when the collaborative memory management assist is installed. For example, BRANCH AND SET AUTHORITY lists an addressing exception on the dispatchable-unit control table; it also recognizes a block-volatility exception on that table when applicable.

Storage-key Manipulation Instructions

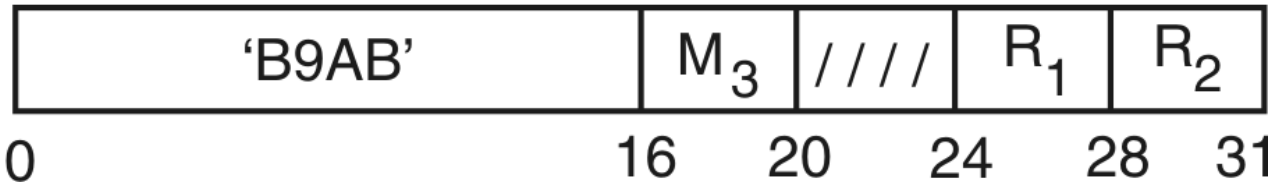
For the instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED, a block-volatility condition is not recognized for the address specified by general register R_2. Similarly, for these instructions, an addressing exception is not recognized due to the 4 KB block at the address specified by general register R_2 being in the unused block-usage state and logically-zero block-content state. Instead, the instruction accesses the storage key without regard to the block-usage and block-content states of this block.

TEST PROTECTION

If the collaborative memory management assist is installed and the location designated by the first operand resides within a 4 KB block that is in both the volatile block-usage state and the logically-zero block-content state, a block-volatility exception is not recognized. Instead, the instruction is completed by setting the condition code to 3.

EXTRACT AND SET STORAGE ATTRIBUTES

ESSA R_1,R_2,M_3 [RRF]



The block-usage state and the block-content state of the 4 KB block designated by the second operand are extracted into the first operand location. The block-usage state and block-content state might be optionally set based on the value of the M_3 field.

Asynchronous to the execution of the instruction, either or both the block-usage state and the block-content state might be changed, as described in the "Associated Functions" on page 891.

In the 24-bit addressing mode, bits 40-51 of general register R_2 designate a 4 KB block in absolute storage, and bits 0-39 and 52-63 of the register are ignored. In the 31-bit addressing mode, bits 33-51 of general register R_2 designate a 4 KB block in absolute storage, and bits 0-32 and 52-63 of the register are ignored. In the 64-bit addressing mode, bits 0-51 of general register R_2 designate a 4 KB block in absolute storage, and bits 52-63 of the register are ignored. In all addressing modes, bits 52-63 should be zero; otherwise the program might not operate compatibly in the future.

When the instruction completes, the general register designated by the R_1 field contains the block-usage state and block-content state of the designated 4 KB block before any specified state change is made. The format of this register is as follows:

Bits 0-59 are not used and unpredictable. Programs that depend on the value of these bits might not operate compatibly in the future.

Block-usage State (US)

Bits 60-61 contain a 2-bit code value indicating the block-usage state of the designated 4 KB block. Following are the meanings of each block-state code value:

Code value
Meaning

0
Stable state

1
Unused state

2
Potentially-volatile state

3
Volatile State

Block-content State (CS)

Bits 62-63 contain a 2-bit code value indicating the block-content state of the designated 4 KB block. Following are the meanings of each block-content state code value:

Code value
Meaning

0
Resident state

1
(reserved)

2
Preserved state

3
Logically-zero state

The M_3 field designates a 4-bit operation-request code (ORC). Following are the meanings of values of this field:

Code value
Meaning

0
Extract Block Attributes. The current block-usage and block-content states of the designated 4 KB block are extracted. No change is made to either state.

1
Set Stable State. The current block-usage and block-content states of the designated 4 KB block are extracted. Following extraction of the states, the block-usage state is set to the stable state.

2
Set Unused State. The current block-usage and block-content states of the designated 4 KB block are extracted. Following extraction of the states, the following occurs:

- The block-usage state is set to the unused state.
- If the block-content state is the preserved state, the contents of the block are immediately discarded and the block-content state is set to the logically-zero state.

3
Set Volatile State. The current block-usage and block-content states of the designated 4 KB block are extracted. Following extraction of the states, the following occurs:

- The block-usage state is set to the volatile state.
- If the block-content state is the preserved state, the contents of the block are immediately discarded and the block-content state is set to the logically-zero state.

4
Set Potentially-Volatile State. The current block-usage and block-content states of the designated 4 KB block are extracted. Following extraction of the states, the following occurs:

- If the block-content state is the resident state, the block-usage state is set to the potentially-volatile state.
- If the block-content state is the preserved state and the change bit for the 4 KB block is one, the block-usage state remains the stable state and the block-content state remains the preserved state. (See Figure 105 on page 885.)
- If the block-content state is the preserved state and the change bit for the 4 KB block is zero, the contents of the block are immediately discarded and the block-usage state is set to the volatile state and the block-content state is set to the logically-zero state.
- If the block-content state is the logically-zero state, the block-usage state is set to the volatile state.

5
Set Stable State and Make Resident. The current block-usage and block-content states of the designated 4 KB block are extracted. Following extraction of the states, the block-content state is set to the resident state and the block-usage state is set to the stable state.

6
Set Stable State If Resident. The current block-usage and block-content states of the designated 4 KB block are extracted. Following extraction of the states, the block-usage state is set to the stable state only if the block-content state is the resident state.

7-15
Reserved. The instruction should not be issued with any of these values in M_3; otherwise the program might not operate compatibly in the future.

When the contents of a 4 KB block are discarded, either synchronously or asynchronously, the reference and change bits in the storage key for the block are set to zeros and the values of the access-control and fetch-protection bits are changed to unpredictable values. With the exception of this case, the instruction has no effect on the reference and change bits for the block.

A serialization and checkpoint-synchronization function is performed before the operation begins and a checkpoint-synchronization function is performed again after the operation completes.

Associated Functions

Subsequent to the execution of EXTRACT AND SET STORAGE ATTRIBUTES, either the block-content state, the block-usage state, or both, might be changed, as follows:

- If the change bit in the storage key is one for a 4 KB block in the potentially-volatile block-usage state, the block-usage state of the block might be changed to the stable state. If the change bit is zero for such a block, the contents of the block might be discarded, thereby causing the block-usage state to be changed to the volatile state, and the block-content state to be changed to the logically-zero state.
- While a 4 KB block is in the stable block-usage state, its contents are not discarded. However, the block-content state of such a block might be changed from the preserved to the resident state or from the resident state to the preserved state.
- While a 4 KB block is in the unused or volatile block-usage state, its contents might be discarded, thereby also causing the block-content state to be changed to the logically-zero state.

Special Conditions

The M_3 field must specify an ORC value in the range of 0-6; otherwise, a specification exception is recognized and no other action is taken.

The 4 KB block designated by the second operand is not subject to low-address protection or key-controlled protection. The block is subject to host page protection when M_3 specifies an ORC value in the range of 1-6. That is, if the block is contained in a read-only page range of a named saved system (NSS) or discontiguous saved segment (DCSS), it is forbidden to change its block-content state.

Unlike ordinary instructions, execution of EXTRACT AND SET STORAGE ATTRIBUTES does not recognize a block-volatility condition for the address specified by general register R_2. Similarly, execution of EXTRACT AND SET STORAGE ATTRIBUTES does not recognize an addressing exception due to the 4 KB block at the address specified by general register R_2 being in the unused block-usage state and logically-zero block-content state.

Resulting Condition Codes

The code remains unchanged.

Program Exceptions

- Addressing (operand 2)
- Operation (if the collaborative memory management assist is not installed)
- Privileged operation
- Protection (store, operand 2, due to host page protection when M_3 specifies an ORC value in the range of 1-6)
- Specification

Programming Notes:

1. After changing the state of a 4 KB block to the unused block-usage state, the program should first change the state of the 4 KB block back to the stable state before referencing the 4 KB block; otherwise, an addressing exception or unpredictable results might occur.
2. If the program changes the state of a 4 KB block to the potentially-volatile or volatile block-usage state, the program should ensure that it will not be harmed or that it can reconstruct the contents of the block if the contents of the block are discarded.

Implications for the DIAGNOSE Instruction and Non-CPU Accesses

An access by an entity other than the CPU (such as the I/O subsystem) to a 4 KB block that is in a discarded state (that is, a block in the logically-zero block-content state and either the unused or the volatile block-usage state) is treated like an access to a block not available in the configuration. For example, an attempt to access a CCW, IDAW, or I/O buffer contained in block in a discarded state results

in a channel program check. Likewise, an access by a CPU asynchronously to an instruction, such as during the storing of an interruption parameter, treats a block in a discarded state like a block not available in the configuration.

It is unpredictable whether a synchronous access by the DIAGNOSE instruction to a block in a discarded state is treated like an access by the CPU (resulting in an addressing or block-volatility exception) or by an entity other than the CPU, as described above. In the latter case, DIAGNOSE might yield the same result (such as a particular condition code or return code) for a block in a discarded state that it would yield for a block not available in the configuration.

To avoid these consequences, the program should ensure that blocks referenced by DIAGNOSE or by non-CPU accesses are in the stable state.

Implications for ESA/390 and ESA/XC Guests

The collaborative memory management assist is available only to guests in z/Architecture mode. EXTRACT AND SET STORAGE ATTRIBUTES results in an operation exception in an ESA/390 or ESA/XC virtual machine. Therefore, a program in such a machine can not place storage blocks into states other than the stable state.

A z/Architecture guest might share storage with an ESA/390 or ESA/XC guest through a named saved system (NSS) or discontiguous saved segment (DCSS) or through the ADRSPACE PERMIT service. It is therefore possible for a guest not operating in z/Architecture mode to encounter storage blocks in a discarded state. A reference by such a guest to a block in a discarded state is treated as a reference to a block not available in the configuration. For example, such a reference by most instructions generates an addressing exception. A block-volatility exception is not presented to an ESA/390 or ESA/XC guest.

Implications for Saved Systems and Segments

Storage in the page ranges of an NSS or DCSS should be in the stable block-usage state at the time of the SAVESYS or SAVESEG command. Otherwise, the SAVESEG or SAVESYS operation might fail, or the contents and the block-usage and block-content states of that storage will be unpredictable when the storage is subsequently loaded into a virtual machine by IPL or DIAGNOSE X'64', respectively.

When an NSS or DCSS containing shared ranges is imbedded into virtual-machine storage, there is a single copy of the block-usage and block-content states for each block in the range. If the range is shared read-only (indicated by page descriptor code SR on DEFSEG or SAVESEG), then EXTRACT AND SET STORAGE ATTRIBUTES instructions are not permitted to change the block-usage or block-content states of blocks in this range (though the block-content state might change in normal system operation). If the range is shared read-write (page descriptor code SW or SN), then all sharing virtual machines observe and operate on a common state for the block. If one sharer issues EXTRACT AND SET STORAGE ATTRIBUTES to change the block-content state, other sharers will observe the new state through EXTRACT AND SET STORAGE ATTRIBUTES Extract, and all sharers are subject to the consequences, such as discard operations and resulting block-volatility and addressing exceptions. For proper operation, virtual machines sharing read-write storage must cooperate with regard to collaborative memory management states, just as they must coooperate with regard to storage contents and storage keys.

Implications for the VMDUMP Command

When the collaborative memory management assist is used, the VMDUMP command omits, without indication, any non-stable pages in the specified range.

Chapter 27. 370 Accommodation Facility Overview

This chapter provides an overview of the 370 Accommodation Facility. It includes background information describing why this facility is useful and introductory information describing how to use it.

Background

One of the reasons that CMS users find it difficult to migrate from virtual machines using the System/370 architecture to virtual machines using a later architecture (370-XA, ESA/370, ESA/390, or ESA/XC, [7] herein called generically *ESA-family* architectures) is that they are using, and must continue to use, programs that cannot tolerate these architectures. Unlike most operating systems, CMS runs its applications in virtual machine supervisor state. This allows applications to include privileged instructions. The lack of low-level system services in older versions of CMS often forced applications to use privileged instructions to accomplish their goals. While the 370-XA architecture was defined to be highly upward compatible with the System/370 architecture *for problem-state instructions*, this does not cover all instructions used by CMS applications. Privileged instructions, the format of the Program Status Word (PSW), and interruption-parameter formats, are not upward compatible from the System/370 architecture to the ESA-family architectures.

Many CMS programs therefore exist that are restricted to running in 370 virtual machines. Many of these programs have not been converted to tolerate ESA-family architectures, so users have been prevented from using the enhancements provided by the newer architectures.

System/370 Constraints

For CMS Level 11 and prior levels, in a 370 virtual machine, CMS could use at most 16 megabytes of storage for programs, due to the 24-bit addresses provided by the System/370 architecture.

Beginning with CMS Level 12, CMS no longer supports 370 virtual machines. Therefore, users are encouraged to migrate to ESA-family virtual machines. The 370 Accommodation Facility can help you in this migration.

High-level Description

The 370 Accommodation Facility allows CMS applications written for 370 virtual machines to run in ESA-family virtual machines. The 370 Accommodation Facility is successful in running the majority of 370 CMS applications, but not *every* 370 application is guaranteed to run in ESA-family virtual machines with 370 Accommodation active.

Two levels of 370 Accommodation support are offered. The first level is provided by CP processing alone. The second level is provided by supplementing the CP processing with CMS assistance. Although the first level of support has less overhead, the second level of support is more powerful, and allows more 370 programs to run in ESA-family virtual machines. Applications that cannot run with either of these 370 Accommodation levels active must be converted to run in an ESA-family virtual machine.

A program written for System/370 may do one or more of the following that prevent it from running in an ESA-family virtual machine:

- Use instructions that exist only in the System/370 architecture. Certain privileged instructions, including I/O instructions and storage-key operations, are unique to System/370, and are replaced by

[7] z/VM supports only ESA/390 (XA mode) and ESA/XC (XC mode) virtual machines for CMS. z/VM also supports Z mode (z/Architecture) and zCMS can be run in it, but a virtual machine in z/Architecture mode may not have 370ACCOM on.

© Copyright IBM Corp. 1991, 2020

new instructions in the ESA family. The instructions of most importance to CMS programs written to run in a 370 virtual machine are: SIO, TIO, SSK, ISK, DIAGNOSE code X'18', and DIAGNOSE code X'20'. Other instructions less commonly used are: SIOF, HIO, HDV, CLRIO, RRB, and the SIOF-Real subcode of DIAGNOSE code X'98'. The 370 Accommodation Facility extends the ESA-family architectures to provide these instructions.

- Load a BC-mode PSW. This is usually done either by a LOAD PSW (LPSW) instruction or by the loading of the new PSW as part of interruption processing. 370 Accommodation extends the ESA-family architectures by causing the BC-mode PSW to be translated into an equivalent ESA PSW, without interrupting the application.
- Manipulate the system mask in the PSW assuming that the PSW is a BC-mode PSW. Because CMS has not always offered ENABLE and DISABLE macros, programs often resorted to direct manipulation of the system mask using instructions like SET SYSTEM MASK (SSM) or STORE THEN OR SYSTEM MASK (STOSM). Because CMS always uses a BC-mode PSW when running in a 370 virtual machine, most programs were written to assume that mode. BC-mode PSWs are not provided in the ESA family, so such programs usually fail in these modes. 370 Accommodation extends the ESA-family architectures by causing a PSW that is manipulated in this manner to be *repaired*, without interrupting the application.
- Use the interval timer. In a System/370 virtual machine, the interval timer is a facility which allows a program to measure either elapsed time or CPU time, depending on the setting established by the CP SET TIMER command. The 370 Accommodation facility extends the ESA-family architectures to provide the interval timer.
- Use interruption parameters, having fetched them from the locations associated with BC-mode PSWs. When an interruption occurs in System/370 architecture, the machine stores the interruption parameters in different places depending on whether the PSW at the time of the interruption was a BC-mode PSW or an EC-mode PSW. 370 CMS programs that replace CMS interruption new PSWs with their own are usually written to use the interruption parameters in the BC-mode locations. Even if the BC-mode PSW that would be loaded as part of interruption processing is repaired, such *PSW-stealing* programs would fail in ESA-family virtual machines because the interruption parameters are not stored in the locations being examined by the program. 370 Accommodation extends the ESA-family architectures by recognizing that a 370 program has stolen the interruption new PSW (specifically, that the interruption new PSW designates BC mode, or in the case of I/O interruptions, that the I/O new PSW designates BC mode or is a 24-bit EC-mode PSW) and presenting interruption parameters in both System/370 BC-mode and ESA-family format and locations. The BC-mode format allows the stealing program to process the interruption. The ESA-family format allows CMS to process it in the event that the stealing program passes control to CMS, which often happens when the interruption trapped is not one that the application sought to handle. Because it is not possible to know beforehand which interruptions will be handled by 370 or by ESA-family programs, both interruption formats are stored in these cases. The additional 370 Accommodation services provided by CMS also allow CP to detect when a program has modified just the low-order three bytes or the low-order four bytes of either the I/O or the external new PSWs. Many programs do this with the intent of merely redirecting control to their own interruption handler, but should nevertheless be considered to be stealing the PSW.

When Should 370 Accommodation be Used?

In general, 370 Accommodation should be activated to try any program that worked in a 370 virtual machine but fails in an ESA-family virtual machine. There are specific symptoms, however, that are indicative of programs that may be aided by 370 Accommodation:

- If a program issues an instruction that is not valid in a given architecture, an operation-exception program interruption occurs. This is the result that would occur if a 370 application tries to issue an instruction (for example, TIO) that does not exist in the ESA-family architectures. CMS reports this event with messages similar to the following:

```
DMSABE148T System abend 0C1 called from hhhhhhhh reason code 00000000
DMSABE141T Operation exception occurred at hhhhhh in routine cccccccc
```

  The first message may not always appear.

- If a program manipulates the system mask of an ESA PSW as if it were BC mode, or loads a BC-mode PSW in an ESA-family virtual machine, a specification-exception program interruption occurs. CMS reports this event with messages similar to the following:

  ```
  DMSABE148T System abend 0C6 called from hhhhhhhh reason code 00000000
  DMSABE141T Specification exception occurred at hhhhhh in routine cccccccc
  ```

  The first message may not always appear.
- If a program uses the interval timer, and pauses until it receives an external interruption from the interval timer, it will appear to hang.
- If a program steals the CMS I/O new PSW and then pauses until it receives a particular interruption, it may appear to hang.
- If a program steals the CMS external new PSW and then pauses until it receives a particular interruption, it may appear to hang, or it may pass an interruption on to CMS that will confuse CMS. CMS reports such an event with messages similar to the following:

  ```
  DMSHDE744R Unexpected external interrupt detected, interrupt status consists
             of: CODE=hhhh, CPUID=hhhh, PARAMETER=hhhhhhhh. Enter a
             1 for ABEND or 2 for RESUME:
  ```

- If a program steals the CMS program-new PSW in an ESA-family virtual machine, replacing it with a BC-mode PSW, the following CP message would result if a program interruption ever occurs:

  ```
  HCPGIR453W CP entered; program interrupt loop
  ```

If you see any of these symptoms when attempting to run a 370 application in an ESA-family virtual machine, 370 Accommodation should be activated to see if it helps.

Choosing a Level of 370 Accommodation

Two levels of 370 Accommodation support are offered. In the first level, CP alone performs dynamic repair actions that allow many 370 programs to run. In the second level of support, CMS manipulates its own I/O and external new PSWs to make it easier for CP to detect PSW-stealing programs. This allows a great number of additional 370 programs to run successfully in ESA-family virtual machines.

The most important difference between the two levels of support is that when it is not needed by an application, there is no performance penalty for activating just the CP level of 370 Accommodation, but there is a performance penalty if you activate the CMS level of 370 Accommodation. To choose which level is best, you should experiment with your 370 applications. First try activating just the CP level of 370 Accommodation when you run your 370 program. If your program works, then there is no need to use the CMS level of 370 Accommodation. If your program does not work, or works except in certain circumstances, try activating the CMS level of 370 Accommodation and running the program again. If it does not work with the CP level of support, but does work when you activate the CMS level of support, then you should use the CMS level of 370 Accommodation when running that program.

Activating 370 Accommodation

The CP level of the 370 Accommodation Facility is activated by issuing the following command:

```
CP SET 370ACCOM ON
```

You can determine the current setting by issuing the CP QUERY SET command.

The CMS level of the 370 Accommodation Facility is activated by issuing the following CMS command:

```
SET CMS370AC ON
```

When you issue the SET CMS370AC ON command, CMS will issue a CP SET 370ACCOM ON command as part of its processing. If you later issue the SET CMS370AC OFF command, CMS will restore the setting of

CP SET 370ACCOM to what it was when the SET CMS370AC ON command was issued. Issue the QUERY CMS370AC command to interrogate the current setting of the CMS 370 Accommodation facility.

If you find that activating 370 Accommodation helps a 370 application to run in an ESA-family virtual machine, you must decide when to activate the facility for everyday execution of the application. You have the following choices:

1. Issue the CP SET 370ACCOM ON command or the SET CMS370AC ON command before starting the 370 application. This is the simplest approach, especially when testing the usefulness of the facility.
2. Write a *cover exec* to preserve the current 370 Accommodation setting, activate it, run the 370 application, and then restore 370 Accommodation to its original setting. A cover exec that activates just the CP level of 370 Accommodation might look like the following:

```
/* Cover exec for a 370 application */
Address Command

/* Get the current state of 370ACCOM */
Parse Value Diag(8,'QUERY SET') With '370ACCOM' accsetting . '15'x

/* Strip any trailing commas in case other parameters are */
/* someday added to the QUERY SET response. */
accsetting=Strip(accsetting,',','Trailing')

/* Activate the 370 Accommodation Facility */
If accsetting\='ON' Then 'CP SET 370ACCOM ON'

/* Run application with the arguments passed to this cover exec. */
/* (Replace 370APPL below with the name of your 370              */
/* application). */
Address CMS '370APPL' Arg(1)
retcode=rc   /* Preserve return code for exit */

/* Restore 370ACCOM, if necessary */
If accsetting\='ON' Then 'CP SET 370ACCOM' accsetting

Exit retcode   /* Return with return code from application */
```

   There are several features of this exec worth mentioning:

   a. It makes no assumptions about the setting of 370 Accommodation when the exec is invoked. Upon exit, it is restored to whatever value it had on entry.
   b. When parsing the response to QUERY SET, it tolerates the presence of parameters after 370ACCOM.
   c. It allows arguments to be passed to the application program, and returns the application's return code upon exit.

   A cover exec that activates the CMS level of 370 Accommodation might look like the following:

```
/* Cover exec for a 370 application */
Address Command

/* Get the current state of CMS370AC */
'PIPE COMMAND QUERY CMS370AC|SPEC W3|VAR accsetting'
If rc\=0 Then Exit rc

/* Activate the CMS 370 Accommodation facility */
If accsetting\='ON' Then 'SET CMS370AC ON'

/* Run application with the arguments passed to this cover exec. */
/* (Replace 370APPL below with the name of your 370              */
/* application). */
Address CMS '370APPL' Arg(1)
retcode=rc   /* Preserve return code for exit */

/* Restore CMS370AC, if necessary */
If accsetting\='ON' Then 'SET CMS370AC' accsetting

Exit retcode   /* Return with return code from application */
```

   This cover exec assumes that the level of CMS is at least CMS Level 12.

3. Issue the CP SET 370ACCOM ON command from your PROFILE EXEC. Your PROFILE EXEC runs each time you IPL CMS. If you find that you always want to activate 370 Accommodation, then putting it in your profile can save you from worrying about each individual application. It is probably best to avoid putting a SET CMS370AC ON command in your PROFILE EXEC, due to the performance degradation it induces. If that userid generally runs only one program, however, or if you find that you nearly always need the CMS level of 370 Accommodation, then you should consider it.
4. The systems programmer may issue the CP SET 370ACCOM ON command from the SYSPROF EXEC associated with the CMS you IPL, or perhaps an exec called from SYSPROF EXEC. The SYSPROF EXEC runs each time you IPL CMS, before your PROFILE EXEC runs. If the systems programmer believes that most users find the facility helpful, you may find that it is already active when your PROFILE EXEC runs. To determine the setting of the CP SET 370ACCOM command, issue the CP QUERY SET command after your PROFILE EXEC runs. To determine the setting of the SET CMS370AC command, issue the QUERY CMS370AC command.

There are several things to be aware of when using 370 Accommodation:

1. The 370 Accommodation Facility is automatically turned off whenever your virtual machine experiences a subsystem reset. Such a reset usually occurs during the execution of the following CP commands:

   IPL
   DEFINE STORAGE
   SET MACHINE
   SYSTEM RESET
   SYSTEM CLEAR
   DETACH CPU

   370 Accommodation is turned off automatically in case the next operating system you IPL is not CMS. The architectural extensions that exist when 370 Accommodation is active might confuse a guest operating system, so CP turns it off.
2. It is possible, though unlikely, for a program to successfully run in an ESA-family virtual machine *before* activating 370 Accommodation, but to fail once 370 Accommodation is activated. See “Possible Adverse Effects on a Working Program” on page 898 for more information on how that can occur. If you find a program like this, you may want to consider using an alternative for activating 370 Accommodation other than including the CP SET 370ACCOM ON or SET CMS370AC ON command in your PROFILE EXEC or the SYSPROF EXEC.
3. When it is not needed by an application, there is no performance penalty for activating just the CP level of 370 Accommodation. When it is needed, there is a slight performance cost because CP is repairing conditions during execution that would otherwise result in program exceptions. The degree of the performance degradation is related to how much repair action is required. Normally, you will not notice it.

   On the other hand, there may be more of a performance cost for activating the CMS level of 370 Accommodation. In order to help CP recognize programs that steal the I/O and external new PSWs, CMS changes its own I/O and external new PSWs in a way that causes CP to perform a small amount of extra processing for every I/O and external interruption reflected to the virtual machine. If your virtual machine fields a lot of these interruptions, you might notice the performance penalty. If you are running a 370 program that *does* need the CMS level of 370 Accommodation, though, the cost of having it active doesn't necessarily rise. It depends only on the frequency with which I/O and external interruptions are received.

Running a Restricted CMS MODULE

The CMS GENMOD command used to have a 370 option that could be used to generate a MODULE file that CMS restricted to executing in a 370 virtual machine. This option prevented the execution of applications in ESA-family virtual machines with 370 Accommodation active.

If you have used this option, you need to do one of the following to enable your programs to run in that environment:

- Use the GENMOD command to generate a new module without using the 370 option. Your program can then run in any virtual machine architecture, so you can run it with 370 Accommodation active.
- Use the CMS SET GEN370 OFF command to cause CMS to bypass checking for that condition when loading the MODULE file. If you no longer have access to the original source program, or at least the TEXT files, this may be your only choice. If you use this method, you may consider adding this command to a *cover exec* for the application, or possibly including it in your PROFILE EXEC.

What is Not Provided by the 370 Accommodation Facility

Not all programs written for a 370 CMS virtual machine run successfully in an ESA-family virtual machine with 370 Accommodation active. The following are some reasons a program may not work:

- If an application uses internal CMS fields, the application is not specifically helped by 370 Accommodation. The format and location of some internal CMS fields have changed from VM release to VM release, and if an application program depends on the layout of such a field, it may fail. Such a program may already fail in 370 virtual machines of the later CMS releases, so it is not unexpected.
- If an application makes use of instructions or architectural facilities that are not typically used by CMS or 370 CMS applications, it may not run even with 370 Accommodation active. An example of such an architectural facility is Dynamic Address Translation (DAT). If a program uses System/370 DAT, it may not work in an ESA-family virtual machine.
- If an application selectively enables for I/O interruptions by setting only some of the channel mask bits in the BC-mode PSW, or by setting specific bits in control register 2, it may not run even with 370 Accommodation active. When this kind of application runs in an ESA-family virtual machine, control register 2 is not examined; and because 370 Accommodation performs only an approximate translation of the PSW enablement, the application may not work as expected.
- If an application loads a BC-mode PSW with the problem-state bit set to one, it may not run with 370 Accommodation active. 370 Accommodation assigns a special meaning to the problem-state bit in BC-mode PSWs, and is not always able to determine whether the special meaning should apply to the PSW in which it's set.

Possible Adverse Effects on a Working Program

If an application program does any of the following, it may fail in an ESA-family virtual machine with 370 Accommodation active, even though it runs when 370 Accommodation is not active. If you encounter such a program, 370 Accommodation should be set off before running the program.

1. If a program depends upon getting an operation-exception program interruption in an ESA-family virtual machine when issuing an instruction valid only on the System/370 architecture, it may fail. When 370 Accommodation is active, CP assumes the application was written to run in a 370 virtual machine, and simulates the instruction. The application may become confused when the expected program interruption does not occur. The likely result is that it will fail to exploit 31-bit addressing, even though it is running in an ESA-family virtual machine.
2. If a program depends upon getting a specification-exception program interruption when loading a BC-mode PSW in an ESA-family virtual machine, it may fail. When 370 Accommodation is active, CP assumes the application was written to run in a 370 virtual machine and translates the PSW to EC mode. The application may become confused when the expected program interruption does not occur.
3. If a program depends upon getting a specification-exception program interruption in an ESA-family virtual machine when setting unassigned bits in the PSW system mask, it may fail. When 370 Accommodation is active, CP assumes the application was written to run in a 370 virtual machine and is trying to enable for I/O interruptions in what it expects is a BC-mode PSW, and CP translates the PSW to the EC-mode equivalent. The application may become confused when the expected program interruption does not occur.

It may sound unlikely that an application would depend on any of these events, but there is a case in which it is possible. Some application programs need to know what architecture mode they are running in because sometimes different actions must be taken by the program for different architecture modes. Usually such a program determines what mode it is in by obtaining the information from CMS. That information is available from CMS using the Extract/Replace CSL service, from information in the simulated OS CVT, or from fields in NUCON. Some applications, however, were written to determine it themselves. The usual procedure is to try something that should work in one architecture mode and fail in others. For example, the program might try loading an EC-mode, 31-bit-mode, PSW. If a specification exception program interruption occurs, then that PSW format is not valid in the current architecture, so the architecture must be System/370 architecture. If no program interruption occurs, then it must be one of the ESA-family architectures. The example just given continues to work even with 370 Accommodation active because the program interruption occurs in a 370 virtual machine, and a 370 virtual machine is not affected by 370 Accommodation.

But, suppose the program tries loading a BC-mode PSW instead. If a program interruption occurs, it knows it is using one of the ESA family architectures because a BC-mode PSW is not permitted in those architectures. If no program interruption occurs, it knows it is using the System/370 architecture. But with 370 Accommodation active, CP allows a BC-mode PSW to be loaded in an ESA-family virtual machine and silently converts it to the equivalent ESA PSW. No program interruption is presented to the application. If an application performs a test like this, it might incorrectly conclude it is in a 370 virtual machine instead of an ESA-family virtual machine. When you have such an application, 370 Accommodation should be set off before running the program. One possible sign of such an incorrect conclusion by the program would be that the program will not exploit 31-bit addressing, even when it is in an ESA-family virtual machine.

Chapter 28. 370 Accommodation Facility Definition

This chapter provides a detailed description of the architecture changes observable in ESA-family virtual machines when the 370 Accommodation Facility is active.

System/370 Instructions

When 370 Accommodation is active, the ESA-family virtual machines are extended to support some instructions that are normally limited to System/370 architecture. The following sections describe these additions. Except where specifically mentioned, the operation of the instructions is as defined in the *System/370 Principles of Operation*.

System/370 I/O Instructions

The following System/370 I/O instructions are provided when 370 Accommodation is active:

- START I/O (SIO)
- START I/O FAST RELEASE (SIOF)
- TEST I/O (TIO)
- HALT I/O (HIO)
- HALT DEVICE (HDV)
- CLEAR I/O (CLRIO)
- DIAGNOSE code X'18'
- DIAGNOSE code X'20'
- DIAGNOSE code X'98', SIOF-Real subcode

These instructions operate the same under 370 Accommodation as they do for System/370, with the following changes:

- The operand System/370 I/O address is treated as an ESA-family device number.
- The addressed channel and device is available to all CPUs in the configuration. The instructions are not restricted to a single CPU as they would be in a 370 virtual machine on VM/ESA.
- The I/O address specified is not limited to X'1FFF', which is normally the case for a 370 virtual machine on VM/ESA. This has the effect of making it appear as though there are 256 channels available to the program.
- If *vestigial status* is pending at the subchannel, it is discarded before proceeding with instruction execution. For more information on this new type of status in the subchannel, see “Vestigial Status” on page 908.
- If the subchannel is not enabled at the beginning of instruction execution, it becomes enabled before proceeding with instruction execution. That is, bit 8 of word 1 of the subchannel-information block associated with the subchannel is set to one.
- I/O interruptions generated from operations initiated by these instructions are masked by interruption subclass (ISC) not channel number. Whether a CPU is enabled for interruptions is determined, therefore, by bit 6 of the PSW and the interruption-subclass mask in control register 6, not channel-enablement bits in the PSW and control register 2.

For the START I/O FAST RELEASE (SIOF) and CLEAR I/O (CLRIO) instructions, it should additionally be noted that bit 0 of control register 0, the block-multiplexing-control bit, *is* examined during instruction execution, even though that bit position is unassigned in the ESA-family architectures.

© Copyright IBM Corp. 1991, 2020

SET STORAGE KEY (SSK)

This instruction operates the same under 370 Accommodation as it does for System/370, with the following changes:

- This instruction operates as if bit 7 of control register 0, the System/370 storage-key-exception-control bit, were set to one. That is, bit 7 of control register 0 is not examined, and is considered to be a one during instruction execution.

INSERT STORAGE KEY (ISK)

This instruction operates the same under 370 Accommodation as it does for System/370, with the following changes:

- This instruction operates as if bit 7 of control register 0, the System/370 storage-key-exception-control bit, were set to one. That is, bit 7 of control register 0 is not examined, and is considered to be a one during instruction execution.
- This instruction operates as if the PSW were in BC mode, with respect to determining what should be placed into bit positions 29-30 of general register R_1.

RESET REFERENCE BIT (RRB)

This instruction operates the same under 370 Accommodation as it does for System/370, with the following changes:

- This instruction operates as if bit 7 of control register 0, the System/370 storage-key-exception-control bit, were set to one. That is, bit 7 of control register 0 is not examined, and is considered to be a one during instruction execution.

ESA-Family Instructions

When 370 Accommodation is active, the following ESA-family instructions are changed. Except where specifically mentioned, the operation of the instructions is as defined in the *Principles of Operation* document appropriate to the specific ESA-family architecture. References are made below to the term *vestigial status*. For more information on this new type of status in the subchannel, see "Vestigial Status" on page 908.

TEST SUBCHANNEL (TSCH)

This instruction operates normally, with the following change:

- If executed against a subchannel that has vestigial status pending, the instruction is executed as if the vestigial status were normal status. That is, the status is stored in the interruption-response block (IRB), with bit 31 of word 0, the status-pending bit, set to one. The status is then cleared, and the instruction completed with condition code 0.

STORE SUBCHANNEL (STSCH)

This instruction operates normally, with the following change:

- If executed against a subchannel that has vestigial status pending, the instruction is executed as if the vestigial status were normal status. That is, the status is stored in the subchannel-information block (SCHIB), with bit 31 of word 0, the status-pending bit, set to one. The status remains pending as vestigial status.

TEST PENDING INTERRUPTION (TPI)

This instruction operates normally, with the following change:

- If TPI recognizes and clears an interruption condition, it discards any vestigial status pending at the subchannel. The status in abeyance behind the vestigial status — there must be such status for the

subchannel to be interruption-pending — becomes ordinary pending status, and the subchannel becomes no longer interruption-pending.

Discarding Vestigial Status

The following ESA-family instructions are changed to discard any vestigial status at the beginning of instruction execution. That is, if vestigial status is pending at the subchannel when any of the following instructions is executed, the vestigial status is discarded before proceeding with instruction execution. If any additional status was held in abeyance behind the vestigial status, it will become normal pending status.

- CLEAR SUBCHANNEL (CSCH)
- HALT SUBCHANNEL (HSCH)
- MODIFY SUBCHANNEL (MSCH)
- RESUME SUBCHANNEL (RSCH)
- START SUBCHANNEL (SSCH)
- DIAGNOSE code X'58'
- SSCH-Real subcode of DIAGNOSE code X'98'
- DIAGNOSE code X'A4'
- DIAGNOSE code X'A8'

Other Instructions

When 370 Accommodation is active, the following instructions are changed:

DIAGNOSE code X'28'

This instruction is provided in both System/370 virtual machines and ESA-family virtual machines. When used by System/370 programs, the Ry register contains a device address, but when used by ESA-family programs, the Ry register contains a subchannel number. Since CP does not know, when the instruction is issued, what kind of program is running, the following heuristic is used to interpret the Ry value specified by the program when 370 Accommodation is active: If the Ry value, when viewed as a System/370 device address, selects a device which exists, and at least one of the following is true, then the Ry value is interpreted as a device address. Otherwise, the Ry value is interpreted as a subchannel number.

- There is an active channel program on the device with that device address, and the channel program was started by a System/370 I/O instruction (e.g., SIO or SIOF).
- The Ry value, when viewed as a 370-XA subchannel number, does not select a subchannel with a device assigned to it.
- The Ry value, when viewed as a 370-XA subchannel number, selects a subchannel with a device assigned to it, but there is no active channel program on that device.

The Interval Timer

The System/370 architecture offers a timing facility called the interval timer, which is a location in storage which is updated by the machine as time passes. When running in a System/370 virtual machine on VM/ESA, the CP SET TIMER command can be used to select whether the interval timer should be updated both when the virtual machine is running and when the virtual machine is in wait state (TIMER REAL), or just when the virtual machine is running (TIMER ON). If no use of the interval timer is intended, the CP SET TIMER OFF command can be used to disable interval timer emulation.

When 370 Accommodation is active, CP will respect the setting of the CP SET TIMER command, and will emulate the presence of an interval timer in ESA-family virtual machines. Note, however, that since this is done without the machine assistance that is normally available for System/370 virtual machines, the

interval timer may not be decremented as often as prescribed by the *System/370 Principles of Operation*. On average, though, CP will update the interval timer at the proper overall rate. The observable difference is that it may not be as smooth as it would be if machine assistance were available. For example, instead of updating the interval timer once every 1/300 of a second, it may be updated twice in 1/150 of a second, or three times in 1/100 of a second.

When 370 Accommodation is activated, the setting of CP SET TIMER is switched from OFF to either ON or REAL, depending on what it was set to before the virtual machine switched to one of the ESA family of architectures. If the System/370 architecture was not previously used, the default is ON. This is the same transition that occurs when a SET MACHINE 370 command is processed.

When an interval timer interruption becomes pending, it is masked by bit 24 of control register 0, the interval-timer subclass-mask bit in the System/370 architecture, even though that bit position is unassigned in the ESA-family architectures. When an interval-timer external interruption is reflected, the interruption condition is never combined with other external interruption conditions, even though this can occur in the System/370 architecture.

When 370 Accommodation is active, the CP SET TIMER command is available to change the type of interval-timer emulation. Normally this command is not available to ESA-family virtual machines. Similarly, the EXTERNAL INTERVAL command can be used to force an interval-timer interruption to become pending.

PSW Conversions

When 370 Accommodation is active, certain events in an ESA-family virtual machine cause changes to be made to the Program Status Word (PSW), either when it is being introduced or when it is being stored. The following is a summary of the PSW conversions that can occur. They are described in more detail following the summary.

- When a BC-mode PSW is introduced, it is converted to an equivalent EC-mode PSW. [8]
- When an EC-mode PSW with unassigned bits set in the system mask is introduced, it is repaired by converting the system mask to the equivalent EC-mode system mask.
- When a *mapped PSW* is introduced, it is converted into an EC-mode PSW substantially equivalent to the original EC-mode PSW. For more information on this new PSW format, see “Mapped PSWs” on page 905.
- When a supervisor call, external, I/O, or program interruption is being presented, and the interruption new PSW is a BC-mode PSW, or in the case of I/O interruptions, if the I/O new PSW is a BC-mode or 24-bit EC-mode PSW, the EC-mode PSW that would be stored as the interruption-old PSW is first converted to BC mode so that the interruption code can be stored as part of the old PSW. If such a conversion cannot be done without loss of information, it is converted to a mapped PSW instead.

BC-mode PSW Conversion

When a program written for System/370 introduces a BC-mode PSW with 370 Accommodation active, it is converted to an EC-mode PSW. A BC-mode PSW is typically introduced with a LOAD PSW (LPSW) instruction or when a new PSW is loaded during the interruption processing. Commands such as CP STORE PSW and CP SYSTEM RESTART can also be used to introduce a new PSW.

However the PSW is introduced, it is converted into an EC-mode PSW using the following process:

1. The condition code is moved from bit positions 34-35 to 18-19.
2. The program mask is moved from bit positions 36-39 to 20-23.
3. Bit 6, the EC-mode I/O-enablement bit, is set to one if any of bits 0-5 are set to one.
4. Bits 0-5, 16-17, and 24-39 are set to zero.

[8] PSWs in the ESA family are not called EC mode, because since only one PSW format exists, differentiation is not necessary. The term as used here refers to any PSW with bit 12 set to one.

5. Bit 12, the EC-mode bit is set to one.

When this conversion is performed, no interruption or other indication is presented to the program.

BC-mode System Mask Conversion

When a program written for System/370 introduces a system mask in the (EC-mode) PSW setting unassigned bits to one, the PSW is repaired. Specifically, whenever the PSW has a format error caused by an unassigned bit in the system mask being set to one, if the rest of the PSW is valid, the format error is repaired by clearing bits 0-5 of the PSW and setting bit 6, the I/O enablement bit, to one. No interruption or other indication of this repair action is presented to the program.

Mapped PSW Conversion

When a *mapped PSW* is introduced, it is converted into an EC-mode PSW that is approximately the same as the EC-mode PSW from which the mapped PSW was originally formed. Mapped PSWs are described in detail in "Mapped PSWs" on page 905. Mapped PSWs may be stored, for example, as interruption old PSWs If an interruption handler reloads the interruption old PSW, introducing the mapped PSW as the current PSW, it is converted into an EC-mode PSW.

If the mapped PSW was unchanged from the time it was stored, then the new EC-mode PSW can differ from the original EC-mode PSW only in the following ways:

- If the original PSW had a key other than X'0' or X'E', the key has been restored to X'0' instead. Unless the program was depending upon getting an access exception for a key mismatch, this should not cause any detrimental effects. Since the original PSW had to be in supervisor state to be mapped in the first place, changing the key this way does not violate any virtual-machine protection mechanisms.
- If the original PSW was enabled for machine-check interruptions, it is now disabled.
- If the original DAT-off PSW had bit 16 set to one, it is now set to zero. Bit 16 is not significant for CMS programs that do not use DAT, so this should not be noticeable.

PSW Conversions During Interruption Processing

When an external, supervisor call, program, or I/O interruption is being presented and the interruption new PSW is a BC-mode PSW, or in the case of I/O interruptions, if the I/O new PSW is a BC-mode or 24-bit EC-mode PSW, the EC-mode PSW that would be stored as the interruption old PSW is first converted to BC mode. This is necessary in order to store the interruption code in the old PSW, which is where 370 CMS applications expect it to be. See "Interruption Parameters" on page 906 for more information on the storing of interruption parameters when 370 Accommodation is active. If the PSW cannot be converted without loss of information, it is converted to a mapped PSW instead.

Specifically, if bits 0-5, 16-17, and 24-39 in the original EC-mode PSW are all zeros, then it can be directly converted to a BC-mode PSW with no loss of information. The following describes that process:

1. The condition code is moved from bit positions 18-19 to 34-35.
2. The program mask is moved from bit positions 20-23 to 36-39.
3. Bits 18-23 are set to zero.
4. Bits 0-5, the BC-mode I/O-enablement bits for channels 0 through 5, are set to one if bit 6 was one in the original PSW.
5. Bit 12, the EC-mode bit, is set to zero.

Mapped PSWs

If the PSW cannot be converted directly without losing information, it is transformed into a mapped PSW. This is the only condition that causes a PSW to be mapped. See "PSW Mapping Algorithm" on page 910 for details of the transformation. A mapped PSW may be relied on to have the following attributes:

- Bit 6 contains the I/O enablement mask bit.
- Bit 7 contains the external enablement mask bit.

- Bit 14 contains the wait-state bit.
- Bits 16-31, when a mapped PSW is stored, contain the interruption code. When a mapped PSW is reloaded, these bits are ignored.
- Bits 32-33 contain the instruction-length code (ILC) of the instruction which caused the interruption, if appropriate for the type of interruption,
- Bits 34-35 contain the condition code.
- Bits 36-39 contain the program mask.

The PSW fields previously listed have the same bit positions in a mapped PSW as they do in a BC-mode PSW. This is appropriate since the interruption handler expects to see a BC-mode old PSW. So long as an interruption handler confines itself to changing just those bits in the interruption old PSW, the mapped PSW may be reloaded safely and still be transformed by CP back into a proper EC-mode PSW.

Interruption Parameters

When a 370 CMS application replaces one of CMS's interruption new PSWs, it probably replaces it with a BC-mode PSW. A program that replaces an interruption new PSW, or even part of an interruption new PSW, is called a *PSW stealer*. When a PSW stealer's interruption handler is driven, it typically examines the interruption to see if it is one that requires special processing. If so, the processing is done, and the interrupted program is resumed. If not, the interruption is typically *passed on* to CMS by loading the PSW that was the new PSW before it was stolen. For instance, if the program is trying to use the clock comparator, it may replace CMS's external new PSW to trap interruptions due to the clock comparator before CMS sees them. If the external interruption is from the clock comparator, it is handled. If the external interruption is not from the clock comparator, it is passed on to CMS. For example, the program may be driving a particular I/O device and want to trap I/O interruptions arriving from that device before CMS sees them.

Interruption codes, such as the external interruption code and the I/O interruption code, are stored as part of the interruption old PSW in System/370 architecture when the PSW at the time of the interruption is in BC mode. In the ESA-family architectures, however, they are stored in other locations in the prefix page. A 370 program that steals a PSW probably looks for the interruption code and other parameters in the locations that are reserved for them according to the BC-mode definition instead of the EC-mode definition. When 370 Accommodation is active, CP handles the different requirements by storing interruption parameters in *both* sets of locations if the interruption new PSW is in BC mode, or in the case of I/O interruptions, if the I/O new PSW is a BC-mode or 24-bit EC-mode PSW. The parameters are put in both sets of locations rather than just in the BC-mode locations to allow CMS to find them where it expects if the PSW stealer ends up passing the interruption on to CMS for processing. Because the restart new PSW and the machine-check new PSW are not typically stolen by CMS programs, this checking is only performed for external, SVC, program, and I/O interruptions.

Because the interruption code is stored as part of the BC-mode old PSW, and these bits are already in use in an EC-mode PSW, the current (always EC-mode) PSW is converted to BC mode before storing it during interruption presentation. Because an EC-mode PSW in ESA-family architectures contains more information than a System/370 BC-mode PSW, something must be done when bits are set in the EC-mode PSW that cannot be transferred directly to a BC-mode PSW. When it is not possible to convert the BC-mode PSW into EC mode without losing information, an attempt is made to convert it into a mapped PSW. The setting of some bits causes the mapping operation to be skipped, some bits are ignored, and some bits are relocated to other positions in the new, mapped PSW. If the resulting BC-mode or mapped PSW is subsequently reloaded, an approximation of the original EC-mode PSW is restored by relocating the bits back to their original positions. See “PSW Conversions During Interruption Processing” on page 905 for general information and “PSW Mapping Algorithm” on page 910 for details of the transformation.

Along with converting the current PSW to BC mode during presentation, the new PSW is converted to EC mode, if it is not in EC mode already, as it becomes the current PSW. This procedure should allow the interruption handler to find the interruption parameters either where they belong for BC-mode PSWs or

where they belong for EC-mode PSWs. An interruption presented in this way is called a *hybrid interruption*.

Special Conditions

The following special conditions may arise during the presentation of the interruption and the storing of the interruption parameters:

- If a program loads a BC-mode PSW that exactly matches the external, SVC, or program new PSW, either all or none of the following may occur to the value in the location assigned for the corresponding old PSW:
  - The value is converted to a BC-mode or mapped PSW
  - The value is combined with the ILC (if appropriate for that type of interruption) and with the interruption code
  - The value is replaced in the old PSW location.
- The storing of the interruption old PSW may be observed to be a multiple-access reference. That is, intermediate values may be observed by other CPUs in the location assigned to hold the interruption old PSW before the final value is stored. The storing of the interruption old PSW is normally single-access and doubleword-concurrent.

Presentation of Interruptions

Under 370 Accommodation, when an external, I/O, program, or SVC interruption is to be presented, the corresponding interruption new PSW is first fetched and examined. When this interruption new PSW is an EC-mode PSW, the interruption is presented as an ESA-family interruption; when the interruption new PSW is a BC-mode PSW, the interruption is presented as a hybrid interruption. In the case of I/O interruptions, if the I/O new PSW is a 31-bit EC-mode new PSW, the interruption is presented as an ESA-family interruption; when the I/O new PSW is a BC-mode or 24-bit EC-mode PSW, the interruption is presented as a hybrid interruption. A hybrid interruption differs from an ESA-family interruption in the following ways:

- Before being stored as the interruption old PSW, the PSW at the time of the interruption is transformed into an equivalent BC-mode PSW, if possible. Otherwise, the PSW is transformed into a mapped PSW. See “PSW Conversions During Interruption Processing” on page 905 for information about this PSW conversion.
- In addition to storing interruption parameters according to the ESA-family of architectures, parameters are stored according to System/370 BC mode. For external, SVC, and program interruptions, the interruption code appears both in its ESA-family architecture location and in bytes 2-3 of the interruption old PSW. For I/O interruptions, the device number is stored in bytes 2-3 of the I/O old PSW, and an ESA-family I/O interruption code (including the subchannel ID and the subchannel's interruption parameter) is stored in its usual location.
- In System/370 architecture, a side effect of an I/O interruption is to clear the status pending at the subchannel. In ESA-family architectures, the subchannel remains status-pending until TSCH is executed (or the status is cleared another way). Under 370 Accommodation, a hybrid I/O interruption differs from both of these: a copy of the status is stored as the System/370 channel-status word (CSW), but the subchannel remains status-pending with *vestigial status* until some other event clears the vestigial status (see “Vestigial Status” on page 908).

When 370 Accommodation is active, all I/O interruptions store a System/370 CSW. If the interruption is not a hybrid interruption, this is the only way in which the interruption differs from a normal ESA-family I/O interruption. Specifically, if the I/O interruption is not a hybrid interruption, no device number is stored in the I/O old PSW, and the subchannel remains status-pending with *normal* (not vestigial) status.

Vestigial Status

When a hybrid I/O interruption is presented, the pending interruption condition is cleared at the subchannel, and subchannel status is stored in the form of a System/370 channel-status word (CSW). This status also remains pending in the subchannel as vestigial status. Vestigial status is handled as follows:

1. If TSCH or STSCH is executed against a subchannel that has vestigial status pending, the instruction is executed as if the vestigial status were normal pending status. That is, TSCH stores the status (with the status-pending bit on) in the IRB, clears it, and completes with condition code 0. STSCH stores the status (with status-pending on) in the SCHIB but leaves it pending as vestigial status.
2. If CSCH, HDV, HIO, HSCH, MSCH, RSCH, SIO, SIOF, SSCH, TIO, DIAGNOSE code X'18', DIAGNOSE code X'20', DIAGNOSE code X'58', DIAGNOSE code X'98' SIOF-Real and SSCH-Real subcode, DIAGNOSE code X'A4', or DIAGNOSE code X'A8' is executed against a subchannel that has vestigial status pending, the vestigial status is discarded and the instruction thereafter executes normally.
3. If additional status is presented (either intermediate or primary status following intermediate status, or secondary status following primary status, or unsolicited status) while vestigial status is pending at the subchannel, then the subchannel becomes interruption-pending. The additional status is held in abeyance in the channel subsystem until the vestigial status is cleared. Such status is never merged with the vestigial status, even for combinations for which the ESA-family architectures would permit merging.

   In such a case, when the interruption subsequently occurs for this subchannel, any vestigial status that is still pending at that time is discarded, and the status in abeyance becomes ordinary pending status. The interruption is then presented as having arisen from this (formerly abeyant) status. Presentation of this interruption follows standard 370-Accommodation rules: it is presented as a ESA-family or hybrid interruption according to the I/O new PSW. Note that if a hybrid interruption is presented, the new (formerly abeyant) status itself becomes vestigial status.
4. TPI does not present an I/O-interruption code designating a subchannel that has only vestigial status pending because, by definition, such a subchannel is not interruption-pending. However, as noted above, the arrival of subsequent status for a subchannel that has vestigial status pending makes that subchannel interruption-pending again. In that case, TPI can recognize and clear that interruption condition. If TPI presents an I/O-interruption code designating a subchannel with status in abeyance behind vestigial status, then the vestigial status is discarded during the execution of TPI. The abeyant status becomes ordinary pending status; and the subchannel becomes no longer interruption-pending.

Programming Notes:

1. Discarding the vestigial status on SIO, SIOF, or SSCH usually makes the subchannel idle and allows the new I/O request to proceed. However, if subsequent status was held in abeyance behind the vestigial status, then the status in abeyance becomes normal (not vestigial) pending status when the original vestigial status is discarded. This newly pending status may prevent execution of a new start function.
2. A consequence of the handling of status that the device presents while vestigial status is pending in the subchannel is that 370 Accommodation will not work reliably in conjunction with a PSW-stealing program in a virtual multiprocessor environment when more than one CPU is simultaneously enabled for the same interruption subclass. As long as only one processor remains enabled for interruptions from a subchannel, the program that receives a hybrid I/O interruption is guaranteed that the status remains available to TSCH provided that the TSCH is executed before the CPU next enables for interruptions from that subchannel. When multiple CPUs are enabled for the same subclass, the status made vestigial by presentation of an interruption on one CPU may be prematurely cleared by presentation on another CPU of an interruption arising from subsequent status.

Notes on the Definition of Vestigial Status:

1. The handling of vestigial status when additional status arrives (described in case “3” on page 908) is done for the following reasons:

   a. A program that processes the interruption according to the ESA-family architectures must be able to retrieve the first status report through TSCH. Thus, the arrival of additional status after the hybrid

interruption is presented but before the program reaches its TSCH must not cause the vestigial status to be discarded.

b. Conversely, a program that processes the interruption according to the System/370 architecture must be allowed to receive the second status report. That is, the vestigial status must not postpone presentation of the second status indefinitely because this could result in a *hang* condition: A program written for System/370 will use the status logged in the CSW and will never issue TSCH. When this program is finished with the CSW, it will simply reenable I/O interruptions. At that time, the vestigial status can safely be discarded and the second status report presented.

The critical dependency here is that an ESA-family program will issue TSCH before enabling for I/O interruptions. ESA-family programs that delay the TSCH beyond the point of reenablement may fail under 370 Accommodation because a second incoming status report will clear the vestigial status before it can be presented to TSCH.

2. Merging of vestigial status with subsequent status is prohibited because such merging would not preserve the behavior desired for 370 Accommodation. If status were merged, then the single resulting status report would have to be classified as either vestigial or ordinary status, and neither is correct. Making the merged status vestigial prevents the subsequent status from being presented in the form of a System/370 CSW. Programs written for System/370 will never process the second status condition. Conversely, making the merged status ordinary status may cause the first status condition to be presented redundantly in CSW form, so that programs written for System/370 will process the same status indication twice.

   Forbidding merging has the additional benefit of simplifying the 370 Accommodation environment by ensuring that when status is presented in a CSW, a subsequent TSCH will present identical status. Thus, System/370 and ESA-family programs will see the same sequence of status reports.

The CMS 370 Accommodation Facility

When the SET CMS370AC ON command is used to activate the CMS level of 370 Accommodation, the following actions are taken by CMS:

- The current setting of the CP 370 Accommodation facility is interrogated and preserved for restoration when the CMS 370 Accommodation facility is turned off again.
- The CP SET 370ACCOM ON command is issued to turn on the CP 370 Accommodation facility.
- The CMS I/O and external new PSWs are replaced with BC-mode PSWs which direct interruption processing to *glue* code.
- When the glue code gets control on an I/O or external interruption, it converts the old PSW associated with the interruption from a BC mode or mapped PSW to the original EC-mode PSW that was in effect at the time of the interruption. Control is then passed to the normal CMS interruption handler by loading the interruption new PSW that was in effect before the SET CMS370AC ON command was issued.

Notes on the Definition:

1. The intent of these actions is to allow CP and CMS to better handle 370 programs which steal only part of the interruption new PSW. Consider a program which overlays just the low-order three bytes of the I/O new PSW. The original I/O new PSW used by CMS is a 31-bit EC-mode PSW. If an application overlays just the low-order three bytes, the result will still be a 31-bit EC-mode PSW. Loading such a PSW during the reflection of an I/O interruption does not trigger CP's hybrid-interruption processing, so the application's interruption handler will not find the interruption parameters in the locations it expects. Moreover, the application's interruption handler will receive control in 31-bit mode, which is likely to cause it to fail, as 370 programs normally expect to run in 24-bit mode. Finally, if the instruction address in the original CMS I/O new PSW is above the 16-megabyte line, the resulting PSW has an instruction address which points neither to the CMS interruption handler nor to the application's interruption handler. Clearly a failure is very likely if control is passed to an unintentional location in storage.
2. Now consider the same scenario, having issued the SET CMS370AC ON command before running the 370 program. Now when the 370 program executes, the I/O new PSW is in BC mode, which in fact is

what the application was coded to expect. When an I/O interruption occurs, hybrid interruption processing is performed by CP because the new PSW is in BC mode. This allows the application's interruption handler to find the interruption parameters in the locations it expects. The application's interruption handler will also receive control in 24-bit mode, which is what it expects. Since the CMS glue code is guaranteed to reside below the 16-megabyte line, there's no danger of an incorrect instruction address being formed when the I/O new PSW is modified.

3. If the application's interruption handler loads the original CMS I/O new PSW that it found, it would pass control to the glue code. The glue code will convert the I/O old PSW back into its original EC-mode form, because that's what the CMS interruption handler expects to find, and then pass control to the I/O interruption handler by loading the original 31-bit EC-mode PSW. This allows the CMS I/O interruption handler to be entered in the addressing mode it expects, with an interruption old PSW in the form it expects. The pending status which caused the I/O interruption is still pending as vestigial status, and is thus available to be drained by the TSCH instruction in the CMS I/O interruption handler.

If the SET CMS370AC OFF command is issued, CMS restores its I/O and external new PSWs to their original values, and sets the CP 370 Accommodation facility to what it was when SET CMS370AC ON was issued. **It is very important that the CP SET 370ACCOM OFF command *not* be issued when SET CMS370AC ON is in effect.** This would leave CMS with BC-mode I/O and external new PSWs without the CP facility active to convert them into EC mode when an interruption occurs. CMS will probably fail soon after receiving the next I/O or external interruption.

Although the CMS 370 Accommodation facility is very useful for running certain types of PSW-stealing applications, it should be noted that since it causes CMS to run with BC-mode I/O and external new PSWs, it generates some additional overhead due to the extra CP processing involved with hybrid interruptions. This extra processing will occur even when you are not running a 370 application program. If possible, it is best to limit your use of the CMS 370 Accommodation facility to when you need it for a particular 370 application program instead of running with it active all of the time.

PSW Mapping Algorithm

"PSW Conversions During Interruption Processing" on page 905 describes a case when it is necessary to convert, when using 370 Accommodation, an EC-mode PSW to a BC-mode PSW. If that is not possible without losing information, the EC-mode PSW is converted instead into a mapped PSW. This section describes the details of that conversion. This information is presented for diagnostic purposes only.

If an EC-mode PSW cannot be converted directly to a BC-mode PSW without losing information, it is transformed according to the following table:

Table 205. Converting an EC-mode PSW to a mapped PSW. This table shows what happens to each bit of an ESA-family EC-mode PSW as it gets converted to a mapped PSW during interruption presentation. This conversion only occurs for PSWs that cannot be converted without loss of information.

Bit Positions in Original EC-mode PSW	Meaning in EC-mode PSW	Bit Positions in New Mapped PSW	Explanation
0	Unassigned	N/A	A 1 in this position will prevent conversion.
1	PER	13	This bit position is vacated in order to free up contiguous bits for the addressing mode and high-order bits of the instruction address.
2-4	Unassigned	N/A	A 1 in any of these positions will prevent conversion.
5	DAT	N/A	A 1 in this position will prevent conversion.
6	I/O enablement	6	This bit is left in this position in case it is manipulated by the interruption handler.
7	External enablement	7	This bit is left in this position in case it is manipulated by the interruption handler.
8-11	Key	11	CMS typically uses only keys X'0' and X'E' in the PSW. These values will be converted to B'0' and B'1', respectively. Any other PSW key will be converted to B'0'.

Table 205. Converting an EC-mode PSW to a mapped PSW. This table shows what happens to each bit of an ESA-family EC-mode PSW as it gets converted to a mapped PSW during interruption presentation. This conversion only occurs for PSWs that cannot be converted without loss of information. *(continued)*

Bit Positions in Original EC-mode PSW	Meaning in EC-mode PSW	Bit Positions in New Mapped PSW	Explanation
12	EC mode	N/A	A 0 in this position will prevent conversion. Bit 12 of the new mapped PSW will be set to 0 to indicate BC mode. This guarantees that the mapped PSW has an early format error associated with it, which in turn ensures that CP will be notified if this mapped PSW is ever re-loaded.
13	Machine-check enablement	N/A	This bit will be ignored in the EC-mode PSW. When the mapped PSW is later re-loaded, the machine-check enablement bit will be set to 0.
14	Wait state	14	This bit is left in this position in case it is manipulated by the interruption handler.
15	Problem state	N/A	A 1 in this position will prevent conversion. Bit 15 of the new PSW will be set to 1 to indicate the PSW is a *mapped* PSW.
16	Address-space control for ESA or Z; unassigned for XC	N/A	A 1 in this position will be ignored. If the virtual machine is in ESA mode or Z mode, the bit is not effective anyway (PSW bit 5 is off). If the virtual machine is in XC mode, the bit must be 0.
17	Address-space control for ESA or Z; AR-mode for XC	10	This bit is preserved because if the virtual machine is in XC mode, this bit position indicates access-register mode.
18-19	Condition code	34-35	Relocate to positions assigned for this field in a BC-mode PSW.
20-23	Program mask	36-39	Relocate to positions assigned for this field in a BC-mode PSW.
24-31	Unassigned	N/A	A 1 in any of these positions will prevent conversion.
32	Addressing mode	0	
33-37	Bits 1-5 of instruction address	1-5	
38-39	Bits 6-7 of instruction address	8-9	
40-63	Bits 8-31 of instruction address	40-63	

The following table shows the converse, which is where each bit of the resulting mapped PSW originates:

Table 206. Constructing a mapped PSW. This table shows where each bit of the resulting mapped PSW originates, when an ESA-family EC-mode PSW is converted to a mapped PSW during interruption presentation. This conversion only occurs for PSWs that cannot be converted without loss of information.

Bit Positions in Resulting Mapped PSW	Meaning in BC-mode PSW	Bit Positions in Original EC-mode PSW	Explanation
0	Channel 0	32	Addressing mode bit of original EC-mode PSW
1-5	Channels 1-5	33-37	Bits 1-5 of instruction address in EC-mode PSW
6	I/O enablement	6	
7	External enablement	7	
8-9	Key bits 0-1	38-39	Bits 6-7 of the instruction address in EC-mode PSW
10	key bit 2	17	The access-register-mode bit for XC-mode virtual machines
11	Key bit 3	8-11	The 4-bit key in the EC-mode PSW will be encoded into one bit for the mapped PSW.
12	BC/EC mode	N/A	Bit 12 of the new mapped PSW will be set to 0 to indicate BC mode
13	Machine-check enablement	1	PER bit of original EC-mode PSW

Table 206. Constructing a mapped PSW. This table shows where each bit of the resulting mapped PSW originates, when an ESA-family EC-mode PSW is converted to a mapped PSW during interruption presentation. This conversion only occurs for PSWs that cannot be converted without loss of information. *(continued)*

Bit Positions in Resulting Mapped PSW	Meaning in BC-mode PSW	Bit Positions in Original EC-mode PSW	Explanation
14	Wait state	14	
15	Problem State	N/A	This bit will be set to one in the new mapped PSW. It and bit 12 are the flags that this BC-mode PSW is actually a *mapped* PSW.
16-31	Interruption code	N/A	This field is the reason behind the conversion. When the interruption is presented, bits 16-31 of the interruption old PSW will contain the interruption code appropriate for the interruption.
32-33	ILC	N/A	When the interruption is presented, bits 32-33 of the interruption old PSW will contain the instruction-length code appropriate for the interruption.
34-35	Condition code	18-19	
36-39	Program mask	20-23	
40-63	Bits 8-31 of instruction address	40-63	

When a mapped PSW is loaded, it is converted back into an approximation of the original EC-mode PSW from which it was formed. This conversion takes place whenever the loaded PSW has bit 15 set to one, and bits 1-5, 8-9, and 12 set to zero. Any other loaded PSW with bit 12 set to zero is treated like a normal BC-mode PSW.

Chapter 29. Store Hypervisor Information (STHYI) Instruction

Use the STHYI instruction to access certain system information available from the following function codes:

FC=0
Processor Capacity Information

FC=1
Hypervisor Environment Information

FC=2
Guest List

FC=3
Designated Guest Information (requires a search key value)

FC=4
Resource Pool List

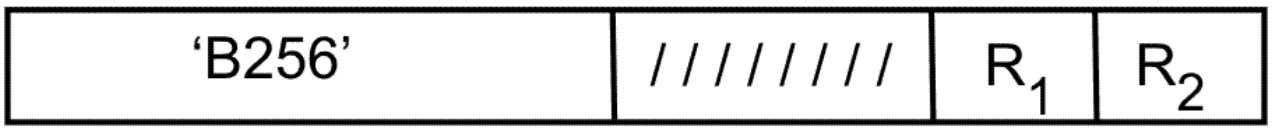
FC=5
Designated Resource Pool Information (requires a search key value)

FC=6
Resource Pool Member List (requires a search key value)

The STHYI instruction is valid in both supervisor and problem states, whenever the store-hypervisor-information-facility bit, bit 74 of the response to the STORE FACILITY LIST EXTENDED (STFLE) instruction, is 1. Otherwise, STHYI results in an operation exception.

When the instruction is provided, some function codes might not be provided as indicated by return code. Function code 0 can be used to determine which function codes are provided and whether they are authorized for the issuer of STHYI.

STHYI R_1,R_2 [RRE]



The R_1 and R_2 fields each designate an even-odd pair of general registers and must specify different even-numbered registers; otherwise a specification exception is recognized.

Note: Bits 16-23 of the instruction are currently unassigned. These bits should contain zeros; otherwise, the program may not operate compatibly in the future.

A function code in general register R_1 indicates which function to perform. All other input is determined by the function code. The function code is in the rightmost 16 bits of general register R_1 (bits 16-31 in ESA/390 or ESA/XC mode, or bits 48-63 in z/Architecture mode). For function codes 2, 4 and 6, the buffer length as an unsigned number of 4 KB pages must be specified in general register R_1 (bits 0-15 in ESA/390 or ESA/XC mode, or bits 32-47 in z/Architecture mode). If the value is zero, then a specification exception is recognized. Those bits should be zeros for other function codes to ensure the program operates compatibly in the future. No checking is done for address wrapping. No checking is done for

© Copyright IBM Corp. 1991, 2020

access to pages of the buffer beyond the last page updated by the response. The remaining bits of general register R_1 should be zeros in order for the program to operate compatibly in the future.

General register R_2 contains the guest logical address of a response buffer, which must be on a 4 KB boundary or a specification exception is recognized. For function codes 0, 1, 3 and 5 the response buffer is 4 KB in length. For function codes 2, 4 and 6 the length is as specified in R_1.

Function codes 3, 5 and 6 require an additional parameter that identifies the object for which that information is requested. Guest general register R_{1+1} must contain the guest logical address of a buffer on a doubleword boundary containing the search key. The key value is an 8-byte value containing a string of up to 8 EBCDIC characters, left-aligned and padded on the right with blanks. If the buffer is not aligned on a doubleword boundary a specification exception is recognized.

Input addresses are guest logical addresses with the same meaning as described for DIAGNOSE codes. See Chapter 1, “The DIAGNOSE Instruction in a Virtual Machine,” on page 3.

When the instruction completes with condition code 3, R_{2+1} contains the following return code. The return code is stored in bits 0-31 in ESA/390 or ESA/XC mode, or bits 0-63 in z/Architecture mode. The response buffer is unmodified except as noted below.

Return Code in R_{2+1}	**Meaning**
X'04'	Unsupported function code
X'08'	Not authorized for the function code
X'0C'	Missing or invalid name
X'10'	Specified name is unknown
X'14'	Response buffer is too small. The minimum required size is returned in response buffer field INFCRQSZ. Values are also provided in INFCVRSN, INFCHDLN and INFCTOTL. Other response buffer contents are unpredictable.

When the instruction completes with condition code 0, general register R_{2+1} contains a return code of 0 indicating that the instruction completed successfully. The return code is stored in bits 0-31 in ESA/390 or ESA/XC mode, or bits 0-63 in z/Architecture mode. The contents of the buffer beyond the reported length of the response are unpredictable.

Unless otherwise stated:

- All character strings returned in the response buffer are left-justified EBCDIC padded on the right with blanks (X'40') and without a terminating null character. This is referred to as "EBCDIC format" in the response buffer.
- All numeric values are unsigned binary values.

Programs written to use HCPINFBK COPY on one release of z/VM can be executed on a new release of z/VM without change. To exploit the function provided by a follow-on release, examine your program for necessary changes. A program will be able to use the contents of the response buffer without change as long as the following are all true:

- Offset fields are used to locate the contents of sections of the response buffer.
- Length fields are used to determine whether fields are included in the response buffer.
- Count fields are used to determine how many of a repeated section are included in the response buffer.
- Validity bits are used to determine whether values of certain fields are meaningful. If the validity bit is off, then the values of the specified fields could not be determined.

Function Code X'0000' - Processor Capacity Information

Function code X'0000' returns information on capacity of general-purpose processors (CPs), z Integrated Information Processor (zIIP), and Integrated Facility for Linux processors (IFLs), including current CPU

resources available at the machine, logical partition, hypervisor, and guest levels, as well as any caps that restrict the guest's use of these resources. This information enables an application to determine the maximum capacity of CPs, zIIPs, and IFLs available to software running in the issuing virtual machine.

When function code X'0000' is specified, general register R_2 contains the guest logical address of a 4 KB response buffer, which must be on a 4 KB boundary or a specification exception is recognized.

When the instruction completes with condition code 3, the response buffer located by the guest logical address in R_2 is unchanged. When the instruction completes with condition code 0, CPU capacity information will be stored into the buffer at the guest logical address specified by register R_2.

The response buffer returns one section of data for each of the following:

- The Header section. See “Function Code X'0000' Response Header (INF0HDR DSECT)” on page 915.
- The Machine section. See “Function Code X'0000' Response Machine Section (INF0MAC DSECT)” on page 917.
- The Partition section. See “Function Code X'0000' Response Partition Section (INF0PAR DSECT)” on page 918.

The response buffer returns up to three sections of data for each of the following:

- The Hypervisor section. See “Function Code X'0000' Response Hypervisor Section (INF0HYP DSECT)” on page 921.
- The Guest section. See “Function Code X'0000' Response Guest Section (INF0GST DSECT)” on page 923.

Multiple levels can be reported for the Hypervisor and Guest sections when, for example, z/VM is run as a guest of z/VM. A maximum of three levels of Hypervisor and Guest sections are reported. When more than three levels exist, the three levels closest to the hardware that support the STHYI instruction are returned in the buffer. These sections are numbered 1-3, starting with the level that is closest to the hardware. The remaining unused area in the 4 KB response buffer is cleared (set to 0).

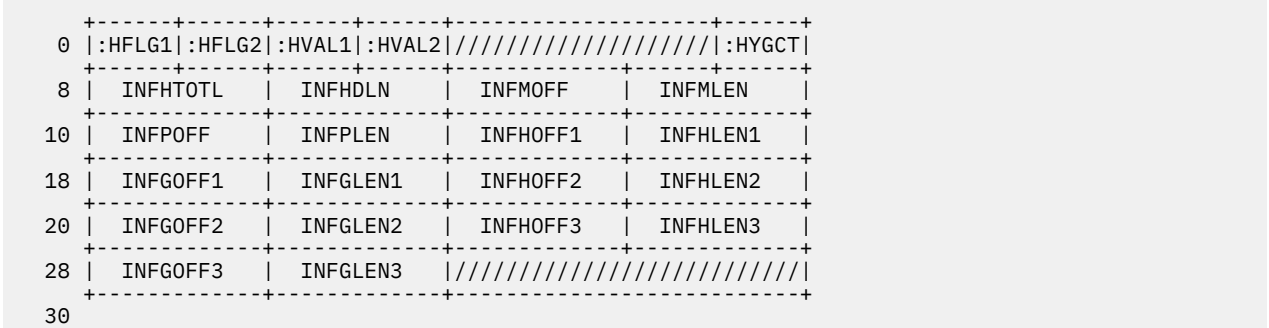
Each Hypervisor section in the function code 0 response contains a mask of the supported function codes. When running a hypervisor second level or higher, a complete response for function code 0 is available only if all hypervisor levels support function code 0.

Function Code X'0000' Response Buffer Format (INF0HDR)

INF0HDR DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

The header section is placed at the beginning of the response buffer and identifies the location and length of all other sections. Valid sections have nonzero offset and length values in the header. Each section provides information about the validity of fields within that section.

Function Code X'0000' Response Header (INF0HDR DSECT)




```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF0HDR        Mappings for STHYI
0000    0 Bitstring    1 INFHFLG1       Header Flag Byte 1 These flag
```

```
                                        settings indicate the environment
                                        that the instruction was executed
                                        in and may influence the value of
                                        validity bits. The validity bits,
                                        not these flags, should be used
                                        to determine if a field is valid.
          1... ....      INFGPDU        X'80' INFGPDU Global Performance
                                        Data unavailable
          .1.. ....      INFSTHYI       X'40' INFSTHYI One or more
                                        hypervisor levels below this
                                        level does not support the STHYI
                                        instruction. When this flag is
                                        set the value of INFGPDU is not
                                        meaningful because the state of
                                        the Global Performance Data
                                        setting cannot be determined.
          ..1. ....      INFVSI         X'20' INFVSI Virtualization stack
                                        is incomplete. This bit indicates
                                        one of 2 cases: 1. One or more
                                        hypervisor levels do not support
                                        STHYI. For this case INFSTHYI
                                        will also be set. 2. There were
                                        more than 3 levels of
                                        guest/hypervisor information to
                                        report.
          ...1 ....      INFBASIC       X'10' INFBASIC Execution
                                        environment is not within a
                                        logical partition.
0001    1 Bitstring    1 INFHFLG2       Header Flag Byte 2 reserved for
                                        IBM use
0002    2 Bitstring    1 INFHVAL1       Header Validity Byte 1 Reserved
                                        for IBM use
0003    3 Bitstring    1 INFHVAL2       Header Validity Byte 2 Reserved
                                        for IBM use
0004    4 Bitstring    3 *              Reserved for IBM use
0007    7 Unsigned     1 INFHYGCT       Count of reported hypervisors/
                                        guests. This indicates how many
                                        hypervisor/guest sections are in
                                        the response buffer up to a
                                        maximum of 3.
          00000003       INF0YGMX       3 Maximum Hypervisor/Guest
                                        sections
0008    8 Unsigned     2 INFHTOTL       Total length of the response
                                        buffer in bytes, including the
                                        Header, Machine, Partition, and
                                        Hypervisor and Guest sections.
000A   10 Unsigned     2 INFHDLN        Length of Header Section in bytes
000C   12 Unsigned     2 INFMOFF        Offset to Machine Section mapped
                                        by INF0MAC
000E   14 Unsigned     2 INFMLEN        Length of Machine Section in
                                        bytes
0010   16 Unsigned     2 INFPOFF        Offset to Partition Section
                                        mapped by INF0PAR
0012   18 Unsigned     2 INFPLEN        Length of Partition Section in
                                        bytes
0014   20 Unsigned     8 INFHYGS1       Hypervisor/Guest Header 1
0014   20 Unsigned     2 INFHOFF1       Offset to Hypervisor Section 1,
                                        mapped by INF0HYP
0016   22 Unsigned     2 INFHLEN1       Length of Hypervisor Section 1 in
                                        bytes
0018   24 Unsigned     2 INFGOFF1       Offset to Guest Section 1 mapped
                                        by INF0GST
001A   26 Unsigned     2 INFGLEN1       Length of Guest Section 1 in
                                        bytes
001C   28 Unsigned     8 INFHYGS2       Hypervisor/Guest Header 2
001C   28 Unsigned     2 INFHOFF2       Offset to Hypervisor Section 2
                                        mapped by INF0HYP
001E   30 Unsigned     2 INFHLEN2       Length of Hypervisor Section 2 in
                                        bytes
0020   32 Unsigned     2 INFGOFF2       Offset to Guest Section 2 mapped
                                        by INF0GST
0022   34 Unsigned     2 INFGLEN2       Length of Guest Section 2 in
                                        bytes
0024   36 Unsigned     8 INFHYGS3       Hypervisor/Guest Header 3
0024   36 Unsigned     2 INFHOFF3       Offset to Hypervisor Section 3
                                        mapped by INF0HYP
0026   38 Unsigned     2 INFHLEN3       Length of Hypervisor Section 3 in
                                        bytes
0028   40 Unsigned     2 INFGOFF3       Offset to Guest Section 3 mapped
                                        by INF0GST
002A   42 Unsigned     2 INFGLEN3       Length of Guest Section 3 in
                                        bytes
```

```
002C   44 Bitstring    4 *              Reserved for IBM use
          00000030       INF0HDSZ       *-INF0HDR Size of header in bytes
          00000006       INF0HDSD       (*-INF0HDR+7)/8 Size of Header in
                                        doublewords
```

Function Code X'0000' Response Hypervisor/Guest Entry (INF0HDYG DSECT)

```
     +-------------+-------------+-------------+-------------+
   0 |  INFYOFF    |  INFYLEN    |  INFGOFF    |  INFGLEN    |
     +-------------+-------------+-------------+-------------+
   8


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF0HDYG       Mappings for STHYI
0000    0 Unsigned     2 INFYOFF        Offset to Hypervisor Section
                                        mapped by INF0HYP
0002    2 Unsigned     2 INFYLEN        Length of Hypervisor Section in
                                        bytes
0004    4 Unsigned     2 INFGOFF        Offset to Guest Section mapped by
                                        INF0GST
0006    6 Unsigned     2 INFGLEN        Length of Guest Section in bytes
          00000008       INF0HYSZ       *-INF0HDYG Size of section
                                        description in bytes
          00000001       INF0HYSD       (*-INF0HDYG+7)/8 Size of section
                                        description in doublewords.
```

Function Code X'0000' Response Machine Section (INF0MAC DSECT)

```
     +------+------+------+------+-------------+-------------+
   0 |:MFLG1|:MFLG2|:MVAL1|:MVAL2|  INFMSCPS   |  INFMDCPS   |
     +------+------+------+------+-------------+-------------+
   8 |  INFMSIFL   |  INFMDIFL   |        INFMNAME-          |
     +-------------+-------------+---------------------------+
  10 |        -INFMNAME          |         INFMTYPE          |
     +---------------------------+---------------------------+
  18 |                       INFMMANU                        |
     |                                                       |
     +-------------------------------------------------------+
  28 |                       INFMSEQ                         |
     |                                                       |
     +---------------------------+---------------------------+
  38 |         INFMPMAN          |///////////////////////////|
     +---------------------------+---------------------------+
  40 |                       INFMPLNM                        |
     +-------------+-----------+---------------------------+
  48 |  INFMSZIIP  |  INFMDZIIP |///////////////////////////|
     +-------------+-----------+---------------------------+
  50


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF0MAC        Mappings for STHYI
0000    0 Bitstring    1 INFMFLG1       Machine Flag Byte 1
          1... ....      INFMPOOL       X'80' INFMPOOL Reserved for IBM
                                        use.
0001    1 Bitstring    1 INFMFLG2       Machine Flag Byte 2 reserved for
                                        IBM use
0002    2 Bitstring    1 INFMVAL1       Machine Validity Byte 1
          1... ....      INFMPROC       X'80' INFMPROC Processor Count
                                        Validity When this bit is on, it
                                        indicates that INFMSCPS,
                                        INFMDCPS, INFMSIFL, and INFMDIFL
                                        contain valid counts. The
                                        validity bit may be off when: -
                                        STHYI support is not available on
                                        a lower level hypervisor, or -
                                        Global Performance Data is not
                                        enabled.
          .1.. ....      INFMMID        X'40' INFMMID Machine ID Validity
                                        This bit being on indicates that
                                        a SYSIB 1.1.1 was obtained from
                                        STSI and information reported in
```

```
                                        the following fields is valid:
                                        INFMTYPE, INFMMANU, INFMSEQ, and
                                        INFMPMAN.
          ..1. ....      INFMMNAM       X'20' INFMMNAM Machine Name
                                        Validity This bit being on
                                        indicates that the INFMNAME field
                                        is valid.
          ...1 ....      INFMPLNV       X'10' INFMPLNV Reserved for IBM
                                        use.
          .... 1...      INFMZIIPV      X'08' INFMZIIPV Machine zIIP
                                        reporting validity. When on, the
                                        INFMSZIIP and INFMDZIIP fields
                                        are valid.
0003    3 Bitstring    1 INFMVAL2       Machine Validity Byte 2 Reserved
                                        for IBM use
0004    4 Unsigned     2 INFMSCPS       Count of shared CPs configured in
                                        the machine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        (INFMPROC)
0006    6 Unsigned     2 INFMDCPS       Count of dedicated CPs configured
                                        in the machine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        (INFMPROC)
0008    8 Unsigned     2 INFMSIFL       Count of shared IFLs configured
                                        in the machine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        (INFMPROC)
000A   10 Unsigned     2 INFMDIFL       Count of dedicated IFLs
                                        configured in the machine or in
                                        the physical partition if the
                                        system is physically partitioned.
                                        (Valid if (INFMPROC)
000C   12 EBCDIC       8 INFMNAME       Machine Name, in EBCDIC format.
                                        This is the CPC name associated
                                        with the processor. (Valid if
                                        (INFMMNAM)
0014   20 EBCDIC       4 INFMTYPE       Machine Type, in EBCDIC format.
                                        This is the machine type reported
                                        by STSI 1.1.1 (Basic Machine
                                        Configuration). (Valid if
                                        (INFMMID)
0018   24 EBCDIC      16 INFMMANU       Machine Manufacturer, in EBCDIC
                                        format. This is the name of the
                                        manufacturer of the configuration
                                        reported by STSI 1.1.1. (Valid if
                                        (INFMMID)
0028   40 EBCDIC      16 INFMSEQ        Sequence Code, in EBCDIC format.
                                        This is the sequence code of the
                                        configuration reported by STSI
                                        1.1.1. (Valid if (INFMMID)
0038   56 EBCDIC       4 INFMPMAN       Plant of Manufacture, in EBCDIC
                                        format. This is the 4-byte code
                                        reported by STSI 1.1.1. (Valid if
                                        (INFMMID)
003C   60 Bitstring    4 *              Reserved for IBM use
0040   64 EBCDIC       8 INFMPLNM       Reserved for IBM use.
0048   72 Signed       2 INFMSZIIP      Count of shared zIIPs configured
                                        in the macine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        INFMZIIPV)
004A   74 Signed       2 INFMDZIIP      Count of dedicated zIIPs
                                        configured in the machine or in
                                        the physical partition if the
                                        system is physically partitioned.
                                        (Valid if INFMZIIPV)
004C   76 Bitstring    4 *              Reserved for IBM use
          00000050       INF0MSIZ       *-INF0MAC Size of Machine Section
                                        in bytes
           0000000A      INF0MSZD       (*-INF0MAC+7)/8 Size of Machine
                                        Section in doublewords
```

Function Code X'0000' Response Partition Section (INF0PAR DSECT)

```
     +------+------+------+------+-------------+-------------+
```

```
   0 |:PFLG1|:PFLG2|:PVAL1|:PVAL2|  INFPPNUM   |  INFPSCPS   |
     +------+------+------+------+-------------+-------------+
   8 |  INFPDCPS   |  INFPSIFL   |  INFPDIFL   |/////////////|
     +-------------+-------------+-------------+-------------+
  10 |                       INFPPNAM                        |
     +---------------------------+---------------------------+
  18 |         INFPWBCP          |         INFPABCP          |
     +---------------------------+---------------------------+
  20 |         INFPWBIF          |         INFPABIF          |
     +---------------------------+---------------------------+
  28 |                       INFPLGNM                        |
     +---------------------------+---------------------------+
  30 |         INFPLGCP          |         INFPLGIF          |
     +---------------------------+---------------------------+
  38 |                       INFPPLNM                        |
     +-------------+-------------+---------------------------+
  40 |  INFPSZIIP  | INFPDZIIP   |         INFPWBZIIP        |
     +-------------+-------------+---------------------------+
  48 |         INFPABZIIP        |         INFPLGZIIP        |
     +---------------------------+---------------------------+
  50
```

Hex	Dec	Type/Val	Lng	Label (dup)	Comments
0000	0	Structure		INF0PAR	Mappings for STHYI
0000	0	Bitstring	1	INFPFLG1	Partition Flag Byte 1
		1...		INFPMTEN	X'80' INFPMTEN Multithreading (MT) is enabled
		.1..		INFPPOOL	X'40' INFPPOOL Reserved for IBM use.
0001	1	Bitstring	1	INFPFLG2	Partition Flag Byte 2 reserved for IBM use.
0002	2	Bitstring	1	INFPVAL1	Partition Validity Byte 1
		1...		INFPPROC	X'80' INFPPROC Processor Count Validity This bit being on indicates that INFPSCPS, INFPDCPS, INFPSIFL, and INFPDIFL contain valid counts. When INFPZIIPV is on, INFPSZIIP and INFPDZIIP contain valid counts.
		.1..		INFPWBCC	X'40' INFPWBCC Partition weight-based capped capacity validity. This bit being on indicates that INFPWBCP and INFPWBIF are valid. When INFPZIIPV is also on, INFPWBZIIP is valid.
		..1.		INFPACC	X'20' INFPACC Partition absolute capped capacity validity. This bit being on indicates that INFPABCP and INFPABIF are valid. When INFPZIIPV is also on, INFPABZIIP is valid.
		...1		INFPPID	X'10' INFPPID Partition ID Validity. This bit being on indicates that a SYSIB 2.2.2 was obtained from STSI and information reported in the following fields is valid: INFPPNUM, INFPPNAM
		.... 1...		INFPLGVL	X'08' INFPLGVL LPAR Group Absolute Capacity Capping information validity This bit being on indicates that INFPLGNM, INFPLGCP and INFPLGIF are valid. When INFPZIIPV is also on, INFPLGZIIP is valid.
		.... .1..		INFPPLNV	X'04' INFPPLNV Reserved for IBM use.
		.... ..1.		INFPZIIPV	X'02' INFPZIIPV Partition zIIP reporting validity. When on, INFPSZIIP and INFPDZIIP are valid, and INFPWBZIIP, INFPABZIIP and INFPLGZIIP may be valid depending on additional validity bits.
0003	3	Bitstring	1	INFPVAL2	Partition Validity Byte 2 reserved for IBM use.
0004	4	Unsigned	2	INFPPNUM	Logical Partition Number. This is the Logical-Partition Number reported by STSI 2.2.2. (Valid if INFPPID)
0006	6	Unsigned	2	INFPSCPS	Count of shared logical CP cores

```
                                        configured for this partition.
                                        (Valid if INFPPROC)
0008    8 Unsigned     2 INFPDCPS       Count of dedicated logical CP
                                        cores configured for this
                                        partition. (Valid if INFPPROC)
000A   10 Unsigned     2 INFPSIFL       Count of shared logical IFL cores
                                        configured for this partition.
                                        (Valid if INFPPROC)
000C   12 Unsigned     2 INFPDIFL       Count of dedicated logical IFL
                                        cores configured for this
                                        partition. (Valid if INFPPROC)
000E   14 Bitstring    2 *              Reserved for IBM use
0010   16 EBCDIC       8 INFPPNAM       Logical Partition Name, in EBCDIC
                                        format. This is the Logical-
                                        Partition Name reported by STSI
                                        2.2.2. (Valid if INFPPID)
0018   24 Unsigned     4 INFPWBCP       Partition weight-based capped
                                        capacity for CPs, a scaled number
                                        where X'00010000' represents one
                                        CPU. Cap is applicable only to
                                        shared processors. Zero if not
                                        capped. (Valid if INFPWBCC)
001C   28 Unsigned     4 INFPABCP       Partition absolute capped
                                        capacity for CPs, a scaled number
                                        where X'00010000' represents one
                                        core. Cap is applicable only to
                                        shared processors. Zero if not
                                        capped. (Valid if INFPACC)
0020   32 Unsigned     4 INFPWBIF       Partition weight-based capped
                                        capacity for IFLs, a scaled
                                        number where X'00010000'
                                        represents one core. Cap is
                                        applicable only to shared
                                        processors. Zero if not capped.
                                        (Valid if INFPWBCC)
0024   36 Unsigned     4 INFPABIF       Partition absolute capped
                                        capacity for IFLs, a scaled
                                        number where X'00010000'
                                        represents one core. Cap is
                                        applicable only to shared
                                        processors. Zero if not capped.
                                        (Valid if INFPACC)
0028   40 EBCDIC       8 INFPLGNM       LPAR Group Name (Valid if
                                        INFPLGVL) EBCDIC, padded on right
                                        with blanks when in an LPAR group
                                        and valid. Binary zeros
                                        otherwise. Only reported when
                                        there is a group cap on CP, IFL, or
                                        zIIP CPU types and the partition has
                                        the capped CPU type.
0030   48 Unsigned     4 INFPLGCP       LPAR Group Absolute Capacity
                                        Value for the CP CPU type when
                                        nonzero. Nonzero only when
                                        INFPLGNM is nonzero and a cap is
                                        defined for this LPAR Group for
                                        the CP CPU type. When nonzero,
                                        contains a scaled number where
                                        X'00010000' represents one core.
                                        (Valid if INFPLGVL)
0034   52 Unsigned     4 INFPLGIF       LPAR Group Absolute Capacity
                                        Value for the IFL CPU type when
                                        nonzero. Nonzero only when
                                        INFPLGNM is nonzero and a cap is
                                        defined for this LPAR Group for
                                        the IFL CPU type. When nonzero,
                                        contains a scaled number where
                                        X'00010000' represents one core.
                                        (Valid if INFPLGVL)
0038   56 EBCDIC       8 INFPPLNM       Reserved for IBM use.
          00000040       INF0PSIZ       *-INF0PAR Size of Partition
                                        Section in bytes
          00000008       INF0PSZD       (*-INF0PAR+7)/8 Size of Partition
                                        Section in doublewords
0040   64 Signed       2 INFPSZIIP      Count of shared logical zIIP
                                        cores configured for this
                                        partition. (Valid if INFPZIIPV)
0042   66 Signed       2 INFPDZIIP      Count of dedicated logical zIIP
                                        cores configured for this
                                        partition. (Valid if INFPZIIPV)

0044   68 Signed       2 INFPWBZIIP     Partition weight-based capped
                                        capacity for zIIPs, a scaled
```

```
                                        number where X'00010000'
                                        represents one core. Cap is
                                        applicable only to shared
                                        processors. Zero if not capped.
                                        (Valid if INFPWBCC & INFPZIIPV)
0048   72 Signed     4 INFPABZIIP       Partition absolute capped
                                        capacity for zIIPs, a scaled
                                        number where X'00010000'
                                        represents one core. Cap is
                                        applicable only to shared
                                        processors. Zero if not capped.
                                        (Valid if INFPACC & INFPZIIPV)
004C   76 Signed     4 INFPLGZIIP       LPAR Group Absolute Capacity
                                        Value for the zIIP CPU type when
                                        nonzero. Nonzero only when
                                        INFPLGNM is nonzero and a cap is
                                        defined for this LPAR Group for
                                        the zIIP CPU type. When nonzero,
                                        contains a scaled number where
                                        X'00010000' represents one core.
                                        (Valid if INFPLGVL & INFPZIIPV)
          00000050     INF0PSIZ         *-INF0PAR Size of Partition
                                        Section in bytes
          0000000A     INF0PSZD         (*-INF0PAR+7)/8 Size of Partition
                                        Section in doublewords
```

Function Code X'0000' Response Hypervisor Section (INF0HYP DSECT)

```
     +------+------+------+------+------+------+------+------+
   0 |:YFLG1|:YFLG2|:YVAL1|:YVAL2|:YTYPE|//////|:YCPT |:YIFLT|
     +------+------+------+------+------+------+------+------+
   8 |                       INFYSYID                        |
     +-----------------------------------------------------+
  10 |                       INFYCLNM                        |
     +-------------+-------------+-------------+-------------+
  18 |  INFYSCPS   |  INFYDCPS   |  INFYSIFL   |  INFYDIFL   |
     +------+------+------+------+------+------+------+------+
  20 |:YINS0|:YINS1|:YINS2|:YINS3|:YINS4|:YINS5|:YINS6|:YINS7|
     +------+------+------+------+------+------+------+------+
  28 |:YAUT0|:YAUT1|:YAUT2|:YAUT3|:YAUT4|:YAUT5|:YAUT6|:YAUT7|
     +------+------+------+------+------+------+------+------+
  30 |:YZIIPT|/////|  :YSZIIP    |////////////////////////////|
     +------+------+-------------+------+------+------+------+
  38


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF0HYP        Mappings for STHYI
0000    0 Bitstring    1 INFYFLG1       Hypervisor Flag Byte 1
          1... ....      INFYLMCN       X'80' INFYLMCN Consumption method
                                        is used to enforce Limithard
                                        caps.
          .1.. ....      INFYLMPR       X'40' INFYLMPR If on, Limithard
                                        caps use prorated core time for
                                        capping. If off, raw CPU time is
                                        used.
          ..1. ....      INFYMTEN       X'20' INFYMTEN Hypervisor is
                                        MT-enabled.
0001    1 Bitstring    1 INFYFLG2       Hypervisor Flag Byte 2 Reserved
                                        for IBM use
0002    2 Bitstring    1 INFYVAL1       Hypervisor Validity Byte 1
          1... ....      INFYZIIPV      X'80' INFYZIIPV Hypervisor zIIP
                                        reporting validity. When on,
                                        INFYSZIIP, INFYZIIPT fields are
                                        valid.
0003    3 Bitstring    1 INFYVAL2       Hypervisor Validity Byte 2
                                        Reserved for IBM use
0004    4 Unsigned     1 INFYTYPE       Hypervisor type
          00000001       INFYTVM        X'01' INFYTVM z/VM is the
                                        hypervisor
          00000002       INFYTKVM       X'02' INFYTKVM KVM is the
                                        hypervisor
          00000003       INFYTZCX       X'03' INFYTZCX zCX is the
                                        hypervisor
0005    5 Bitstring    1 *              Reserved for IBM use
0006    6 Bitstring    1 INFYCPT        Threads in use per CP core. This
                                        value is reported for the current
```

```
                                        configuration settings even when
                                        the guest CPUs are not dispatched
                                        on CPs. The value is set only
                                        when SMT enabled as indicated by
                                        INFYFLG1.INFYMTEN.
0007    7 Bitstring    1 INFYIFLT       Threads in use per IFL core. This
                                        value is reported for the current
                                        configuration settings even when
                                        the guest CPUs are not dispatched
                                        on IFLs. The value is set only
                                        when SMT enabled as indicated by
                                        INFYFLG1.INFYMTEN.
0008    8 EBCDIC       8 INFYSYID       System Identifier, in EBCDIC
                                        format, left justified and padded
                                        with blanks. This is the value
                                        generally specified on the
                                        SYSTEM_IDentifier statement in
                                        the system configuration file.
                                        Blank if non-existent.
0010   16 EBCDIC       8 INFYCLNM       Cluster Name, in EBCDIC format,
                                        left justified and padded with
                                        blanks. This is the name on the
                                        SSI statement in the system
                                        configuration file. Blank if
                                        non-existent.
0018   24 Unsigned     2 INFYSCPS       Number of CP cores shared by
                                        non-dedicated CP-dispatched
                                        virtual CPUs of this hypervisor.
001A   26 Unsigned     2 INFYDCPS       Reserved for IBM use
001C   28 Unsigned     2 INFYSIFL       Number of IFL cores shared by
                                        non-dedicated IFL-dispatched
                                        virtual CPUs of this hypervisor.
001E   30 Unsigned     2 INFYDIFL       Reserved for IBM use
0020   32 Bitstring    8 INFYINSF (0)   Mask of installed function codes.
                                        Bit position corresponding to the
                                        function code number is on if the
                                        function code is supported by
                                        this hypervisor. Bits may be on
                                        even if the guest is not
                                        authorized.
0020   32 Bitstring    1 INFYINS0       Function codes 0-7.
          1... ....      INFYFCCP       X'80' INFYFCCP FC=0 - Obtain CPU
                                        Capacity Info.
          .1.. ....      INFYFHYP       X'40' INFYFHYP FC=1 - Hypervisor
                                        Environment Info
          ..1. ....      INFYFGLS       X'20' INFYFGLS FC=2 - Guest List
          ...1 ....      INFYFGST       X'10' INFYFGST FC=3 - Designated
                                        Guest Info
          .... 1...      INFYFPLS       X'08' INFYFPLS FC=4 - Resource
                                        Pool List
          .... .1..      INFYFPDS       X'04' INFYFPDS FC=5 - Designated
                                        Resource Pool Information
          .... ..1.      INFYFPML       X'02' INFYFPML FC=6 - Resource
                                        Pool Member List
0021   33 Bitstring    1 INFYINS1       Function codes 8-15.
0022   34 Bitstring    1 INFYINS2       Function codes 16-23.
0023   35 Bitstring    1 INFYINS3       Function codes 24-31.
0024   36 Bitstring    1 INFYINS4       Function codes 32-39.
0025   37 Bitstring    1 INFYINS5       Function codes 40-47.
0026   38 Bitstring    1 INFYINS6       Function codes 48-55.
0027   39 Bitstring    1 INFYINS7       Function codes 56-63.
0028   40 Bitstring    8 INFYAUTF (0)   Mask of authorized function
                                        codes. Bit position corresponding
                                        to the function code number is on
                                        if the function code is supported
                                        by this hypervisor and the guest
                                        has been authorized in the
                                        directory.
0028   40 Bitstring    1 INFYAUT0       Function codes 0-7.
          1... ....      INFYFCCP       X'80' INFYFCCP FC=0 - Obtain CPU
                                        Capacity Info.
          .1.. ....      INFYFHYP       X'40' INFYFHYP FC=1 - Hypervisor
                                        Environment Info
          ..1. ....      INFYFGLS       X'20' INFYFGLS FC=2 - Guest List
          ...1 ....      INFYFGST       X'10' INFYFGST FC=3 - Designated
                                        Guest Info
          .... 1...      INFYFPLS       X'08' INFYFPLS FC=4 - Resource
                                        Pool List
          .... .1..      INFYFPDS       X'04' INFYFPDS FC=5 - Designated
                                        Resource Pool Information
          .... ..1.      INFYFPML       X'02' INFYFPML FC=6 - Resource
                                        Pool Member List
```

```
0029   41 Bitstring    1 INFYAUT1       Function codes 8-15.
002A   42 Bitstring    1 INFYAUT2       Function codes 16-23.
002B   43 Bitstring    1 INFYAUT3       Function codes 24-31.
002C   44 Bitstring    1 INFYAUT4       Function codes 32-39.
002D   45 Bitstring    1 INFYAUT5       Function codes 40-47.
002E   46 Bitstring    1 INFYAUT6       Function codes 48-55.
002F   47 Bitstring    1 INFYAUT7       Function codes 56-63.
0030   48 Bitstring    1 INFYZIIPT      Threads in use per zIIP core.
                                        This value is reported for the
                                        current configuration settings
                                        even when the guest CPUs are not
                                        dispatched on zIIPs. The value is
                                        set only when SMT enabled as
                                        indicated by INFYFLG1.INFYMTEN.
                                        (Valid if INFYZIIPV)
0031   49 Bitstring    1 *              Reserved for IBM use.
0032   50 Signed       2 INFYSZIIP      Number of zIIP cores shared by
                                        non-dedicated zIIP-dispatched
                                        virtual CPUs of this hypervisor.
                                        (Valid if INFYZIIPV)
                                        Note: zIIP virtual CPUs are never
                                        dedicated.
0034   52 Signed       4 *              Reserved for IBM use.
          00000038       INF0YSIZ       *-INF0HYP Size of Hypervisor
                                        Section in bytes
          00000007       INF0YSZD       (*-INF0HYP+7)/8 Size of
                                        Hypervisor Section in doublewords
```

Function Code X'0000' Response Guest Section (INF0GST DSECT)

```
        All counts of CPs, IFLs, and zIIPs and capacity values
        are based on quantities of cores when multithreading is
        enabled. While virtual CPUs are thread dispatched
        they have the potential of consuming an entire core.
        Therefore virtual CP, IFL, and zIIP counts are counts of
        virtual CPUs but treated as counts of cores.

     +------+------+------+------+--------------------------+
   0 |:GFLG1|:GFLG2|:GVAL1|:GVAL2|        INFGUSID-          |
     +------+------+------+------+-------------+------------+
   8 |        -INFGUSID          |  INFGSCPS   |  INFGDCPS  |
     +------+--------------------+-------------+------------+
  10 |:GCPDT|////////////////////|         INFGCPCC          |
     +------+------+-------------+------+--------------------+
  18 |  INFGSIFL   |  INFGDIFL   |:GIFDT|////////////////////|
     +-------------+-------------+------+--------------------+
  20 |         INFGIFCC          |:GPFLG|////////////////////|
     +---------------------------+------+--------------------+
  28 |                       INFGPNAM                         |
     +---------------------------+---------------------------+
  30 |         INFGPCCC          |         INFGPICC          |
     +-------------+--------+----+---------------------------+
  38 | INFGSZIIP   |:GZIIPDT|////|        INFGZIIPCC         |
     +-------------+--------+----+---------------------------+
  40 |         INFGPZCC          |////////////////////////////|
     +---------------------------+---------------------------+
  38

Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF0GST        Mappings for STHYI
0000    0 Bitstring    1 INFGFLG1       Guest Flag Byte 1
          1... ....      INFGMOB        X'80' INFGMOB Guest mobility is
                                        enabled.
          .1.. ....      INFGMCPT       X'40' INFGMCPT Guest has multiple
                                        CPU types
          ..1. ....      INFGCPLH       X'20' INFGCPLH Guest CP dispatch
                                        type has LIMITHARD cap.
          ...1 ....      INFGIFLH       X'10' INFGIFLH Guest IFL dispatch
                                        type has LIMITHARD cap.
          .... 1...      INFGVCPT       X'08' INFGVCPT Virtual CPs are
                                        thread dispatched
          .... .1..      INFGVIFT       X'04' INFGVIFT Virtual IFLs are
                                        thread dispatched
          .... ..1.      INFGZIIPH      X'02' INFGZIIPH Guest zIIP
                                        dispatch type has LIMITHARD cap
                                        (Valid if INFGZIIPV).
          .... ...1      INFGVZIIPT     X'01' INFGVZIIPT Virtual zIIPs
                                        are thread dispatched (Valid if
```

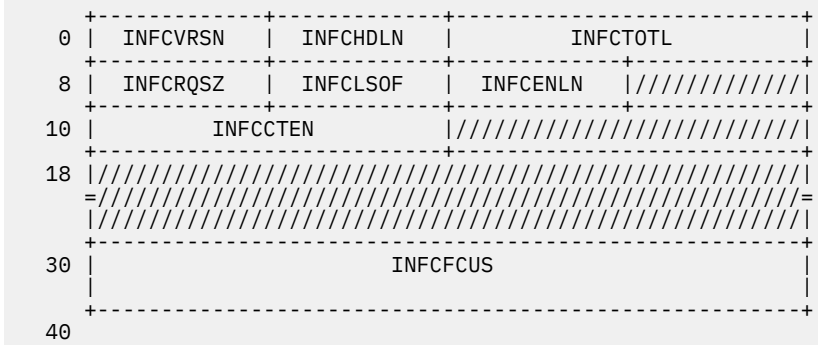
```
                                        INFGZIIPV).

0001    1 Bitstring    1 INFGFLG2       Guest Flag Byte 2 Reserved for
                                        IBM use
0002    2 Bitstring    1 INFGVAL1       Guest Validity Byte 1 Reserved
                                        for IBM use
          1... ....      INFGZIIPV      X'80' INFGZIIPV Guest zIIP
                                        reporting validity. When on,
                                        INFGSZIIP, INFGZIIPDT,
                                        INFGZIIPCC, INFGPZCC fields, and
                                        INFGZIIPH, INFGVZIIPT, INFGPZLH,
                                        and INFGPZPC flags are valid.
0003    3 Bitstring    1 INFGVAL2       Guest Validity Byte 2 Reserved
                                        for IBM use
0004    4 EBCDIC       8 INFGUSID       Guest's userid, in EBCDIC format
000C   12 Unsigned     2 INFGSCPS       Number of guest shared CPs
000E   14 Unsigned     2 INFGDCPS       Reserved for IBM use
0010   16 Unsigned     1 INFGCPDT       Dispatch type for guest CPs This
                                        field is valid if INFGSCPS or
                                        INFGDCPS is greater than zero.
                                        Always INGGPUCCP.
          00000000       INFGPUCCP      X'00' INFGPUCCP General Purpose
                                        (CP)
          00000003       INFGPUCIFL     X'03' INFGPUCIFL Integrated Fac
                                        for Linux (IFL).
          00000005       INFGPUCZIP     X'05' INFGPUCZIP zSeries
                                        Integrated Information Processor
                                        (zIIP).
          000000FF       INFGPUCZCP     X'FF' INFGPUCZCP May be
                                        dispatched on zIIP and CP
                                        Processors due to spillover.
0011   17 Bitstring    3 *              Reserved for IBM use
0014   20 Unsigned     4 INFGCPCC       Guest current capped capacity for
                                        CP-dispatched, shared vCPUs, a
                                        scaled number where X'00010000'
                                        represents one core. Zero if not
                                        capped.
0018   24 Unsigned     2 INFGSIFL       Number of guest shared IFLs
001A   26 Unsigned     2 INFGDIFL       Reserved for IBM use
001C   28 Unsigned     1 INFGIFDT       Dispatch type for guest IFLs.
                                        This field is valid if INFGSIFL
                                        or INFGDIFL is greater than zero.
                                        May be INFGPUCIFL or INFGPUCCP.
          00000000       INFGPUCCP      X'00' INFGPUCCP General Purpose
                                        (CP)
          00000003       INFGPUCIFL     X'03' INFGPUCIFL Integrated Fac
                                        for Linux (IFL).
          00000005       INFGPUCZIP      X'05' INFGPUCZIP zSeries
                                        Integrated Information Processor
                                        (zIIP).
          000000FF       INFGPUCZCP      X'FF' INFGPUCZCP May be
                                        dispatched on zIIP and CP
                                        Processors due to spillover.
001D   29 Bitstring    3 *              Reserved for IBM use
0020   32 Unsigned     4 INFGIFCC       Guest current capped capacity for
                                        IFL-dispatched, shared vCPUs, a
                                        scaled number where X'00010000'
                                        represents one core. Zero if not
                                        capped.
0024   36 Bitstring    1 INFGPFLG       Resource Pool Capping Flags
          1... ....      INFGPCLH       X'80' INFGPCLH Resource Pool's CP
                                        virtual type has LIMITHARD cap.
          .1.. ....      INFGPCPC       X'40' INFGPCPC Resource Pool's CP
                                        virtual type has CAPACITY cap.
          ..1. ....      INFGPILH       X'20' INFGPILH Resource Pool's
                                        IFL virtual type has LIMITHARD
                                        cap.
          ...1 ....      INFGPIFC       X'10' INFGPIFC Resource Pool's
                                        IFL virtual type has CAPACITY
                                        cap.
          .... 1...      INFPRCTM       X'08' INFPRCTM Resource Pool uses
                                        prorated core time.
          .... .1..      INFGPZLH        X'04' INFGPZLH Resource Pool's
                                        zIIP virtual type has LIMITHARD
                                        cap. (Valid if INFGZIIPV)
          .... ..1.      INFGPZPC        X'02' INFGPZPC Resource Pool's
                                        zIIP virtual type has CAPACITY
                                        cap. (Valid if INFGZIIPV)
0025   37 Bitstring    3 *              Reserved for IBM use
0028   40 EBCDIC       8 INFGPNAM       Resource pool name. Blanks if not
                                        Resource Pool.
0030   48 Unsigned     4 INFGPCCC       Resource pool capped capacity for
```

```
                                        shared virtual CPs, a scaled
                                        number where X'00010000'
                                        represents one core if capped.
                                        Zero if not capped.
0034   52 Unsigned     4 INFGPICC       Resource pool capped capacity for
                                        shared virtual IFLs, a scaled
                                        number where X'00010000'
                                        represents one core if capped.
                                        Zero if not capped.
0038   56 Signed       2 INFGSZIIP      Number of guest shared zIIPs.
                                        (Valid if INFGZIIPV)
                                        Note: zIIP virtual CPUs are never
                                        dedicated.
003A   58 Signed       1 INFGZIIPDT     Dispatch type for guest zIIPs.
                                        This field is set when INFGSZIIP
                                        > 0. May be INFGPUCZIIP,
                                        INFGPUCCP or INFGPUCZCP. (Valid
                                        if INFGZIIPV)
          00000000       INFGPUCCP      X'00' INFGPUCCP General Purpose
                                        (CP)
          00000003       INFGPUCIFL     X'03' INFGPUCIFL Integrated Fac
                                        for Linux (IFL).
          00000005       INFGPUCZIP     X'05' INFGPUCZIP zSeries
                                        Integrated Information Processor
                                        (zIIP).
          000000FF       INFGPUCZCP     X'FF' INFGPUCZCP May be
                                        dispatched on zIIP and CP
                                        Processors due to spillover.
003B   59 Bitstring    1 *              Reserved for IBM use.
003C   60 Signed       4 INFGZIIPCC     Guest current capped capacity for
                                        zIIP-dispatched, shared vCPUs, a
                                        scaled number where X'00010000'
                                        represents one core. Zero if not
                                        capped. (Valid if INFGZIIPV)
0040   64 Signed       4 INFGPZCC       Resource pool capped capacity for
                                        shared virtual zIIPs, a scaled
                                        number where X'00010000'
                                        represents one core if capped.
                                        Zero if not capped. (Valid if
                                        INFGZIIPV)
0044   68 Signed       4 *              Reserved for IBM use.
          00000048       INF0GSIZ       *-INF0GST Size of Guest Section
                                        in bytes
          00000009       INF0GSZD       (*-INF0GST+7)/8 Size of Guest
                                        Section in doublewords
```

Common Header Section (INFCHDR)

This common header (INFCHDR) is imbedded at the start of the response buffer for function codes X'0001' through X'0006'. The common header includes information about the size of the response, as well as list designation fields used to locate and traverse a list that may be included in the response.

Common Header Format (INFCHDR DSECT)




```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INFCHDR        Mappings for STHYI
      INFCHDR - INFBK Common Header of Response Buffer
```

```
         - This common header section defines the beginning
           of the response buffer for function codes that
           insert it at the start of the function-code-specific
           response buffer definition.
         - The INFxBK of each function-code-specific response
           buffer imbedding this common header describes the
           changes introduced for all version codes as a set
           of equates defined for the redefinition of INFCVRSN.
           The service level that introduced the change is also
           noted in the comment on the equates.
         - Function codes that don't return a list will have
           zero values in INFCLSOF, INFCENLN and INFCCTEN.
         - Function codes that return a list but have no list
           data to return will return zero values in INFCLSOF
           and INFCCTEN, but returns the entry length in INFCENLN.
         - If the definition of this common header changes then
           the version number for all function codes that use
           it will be increased to indicate the change in the
           response buffer format for that function code.
         - The length of the common header section (INFCHDR) will
           never change to prevent shifting offsets of fields
           in the function-code-specific block following the
           common header.
0000    0 Unsigned     2 INFCVRSN       Version number indicating the
                                        format of the response buffer for
                                        the requested function code. See
                                        response buffer for each function
                                        code for associated versions.
0002    2 Unsigned     2 INFCHDLN       Length of the full header that
                                        imbedded this common section.
0004    4 Unsigned     4 INFCTOTL       Total length of the actual data
                                        returned in all sections of the
                                        response buffer.
0008    8 Unsigned     2 INFCRQSZ       Required buffer size as a number
                                        of 4K pages needed to hold the
                                        complete response.
000A   10 Unsigned     2 INFCLSOF       Offset to start of list from the
                                        beginning of the INFxBK imbedding
                                        this common header DSECT. Entries
                                        are mapped by function-
                                        code-specific entry definitions.
000C   12 Unsigned     2 INFCENLN       Length of list entry.
000E   14 Unsigned     2 *              Reserved for IBM use
0010   16 Unsigned     4 INFCCTEN       Count of list entries.
0014   20 Unsigned     4 *              Reserved for IBM use
0018   24 Dbl-Word     8 * (3)          Reserved for IBM use
0030   48 Dbl-Word     8 INFCFCUS (2)   Function code-specific section.
                                        Used only if redefined within an
                                        INFxBK imbedding INFCHDR DSECT.
                                        It can be redefined to add
                                        section offsets and lengths.
          00000040       INFCSB1        *-INFCHDR Version 1 length in
                                        bytes of the Common Header.
          00000008       INFCSD1        (*-INFCHDR+7)/8 Version 1 length
                                        in doublewords of the Common
                                        Header.
          00000040       INFCSZB        *-INFCHDR Maximum supported
                                        version length in bytes of the
                                        Common Header.
          00000008       INFCSZD        (*-INFCHDR+7)/8 Maximum supported
                                        version length in doublewords of
                                        the Common Header.
```

Function Code X'0001' - Hypervisor Environment Information

Function code X'0001' returns hypervisor environment information including current CPU resources and utilization information available at the machine, logical partition, hypervisor, and guest levels. This information enables an authorized application to perform CPU performance monitoring and management of guests of the z/VM system.

Use of this function code is authorized in the user directory by OPTION STHYI-UTIL.

When the instruction completes with condition code 3, the response buffer located by the guest logical address in R_2 is unchanged. When the instruction completes with condition code 0, CPU capacity and utilization information will be stored into the buffer at the guest logical address specified by register R_2.

This response buffer contains information as described by HCPINFBK COPY in HCPGPI MACLIB. Refer to the INF1BK DSECT.

The response buffer returns one section of data for each of the following:

- The Header section. See "Function Code X'0001' Response Header (INF1BK DSECT)" on page 928.
- The Machine section. See "Function Code X'0001' Response Machine Section (INF1MAC DSECT)" on page 931.
- The Partition section. See "Function Code X'0001' Response Partition Section (INF1PAR DSECT)" on page 932.

The response buffer returns up to three sections of data for each of the following:

- The Hypervisor section. See "Function Code X'0001' Response Hypervisor Section (INF1HYP DSECT)" on page 937.
- The Guest section. See "Function Code X'0001' Response Guest Section (INF1GST DSECT)" on page 940.

Multiple levels can be reported for the Hypervisor and Guest sections when, for example, z/VM is run as a guest of z/VM. A maximum of three levels for Hypervisor and Guest sections is reported. When more than three levels exist, the three levels closest to the hardware that support the STHYI instruction are returned in the buffer. These sections are numbered 1-3, starting with the level reported which is closest to the hardware. The contents of the remaining unused area in the 4 KB response buffer are unpredictable.

If the guest issuing the STHYI is not authorized the STHYI instruction completes with CC3 and RC=8.

When running z/VM second level or higher, a complete response for function code 1 is available only if all hypervisor levels support function code 1 and the guest at each layer is authorized for function code 1.

Function Code X'0001' Response Buffer Format (INF1BK)

INF1BK DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

The version number (INF1VRSN) in the function code X'0001' response header can be used to clearly identify whether the function code is at the required service level and can be used by applications to show actual and required version in messages when required support is missing.

The header section is placed at the beginning of the response buffer and identifies the location and length of all other sections. Valid sections have nonzero offset and length values in the header. Each section provides information about validity of fields within that section.

The list designation fields in the common header are stored as zeroes.

Function Code X'0001' Response Header (INF1BK DSECT)

```
     +-------------------------------------------------------+
   0 |                                                       |
     =                       INF1CHDR                        =
     |                                                       |
     +------+------+------+------+--------------------+------+
  40 |:1HFL1|:1HFL2|:1HVL1|:1HVL2|////////////////////|:1HYGC|
     +------+------+------+------+--------------------+------+
  48 |  INF1MOFF   |  INF1MLEN   |  INF1POFF   |  INF1PLEN   |
     +-------------+-------------+-------------+-------------+
  50 |  INF1YOF1   |  INF1YLN1   |  INF1YVR1   |/////////////|
     +-------------+-------------+-------------+-------------+
  58 |  INF1GOF1   |  INF1GLN1   |  INF1GVR1   |/////////////|
     +-------------+-------------+-------------+-------------+
  60 |  INF1YOF2   |  INF1YLN2   |  INF1YVR2   |/////////////|
     +-------------+-------------+-------------+-------------+
  68 |  INF1GOF2   |  INF1GLN2   |  INF1GVR2   |/////////////|
     +-------------+-------------+-------------+-------------+
  70 |  INF1YOF3   |  INF1YLN3   |  INF1YVR3   |/////////////|
     +-------------+-------------+-------------+-------------+
  78 |  INF1GOF3   |  INF1GLN3   |  INF1GVR3   |/////////////|
     +-------------+-------------+-------------+-------------+
  80


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF1BK         Mappings for STHYI
        Function code x'0001': Hypervisor Environment Information
        This function code reports on the environment of the
        invoker of STHYI.
        The response buffer for function code x'0001' consists
        of five or more sections of data:
         Header Section       (INF1BK DSECT, imbeds INF1CHDR)
         Machine Section      (INF1MAC DSECT)
         Partition Section    (INF1PAR DSECT)
         Hypervisor Section 1 (INF1HYP DSECT)
         Guest Section 1      (INF1GST DSECT)
         Hypervisor Section 2 (may not be present)
         Guest Section 2      (may not be present)
         Hypervisor Section 3 (may not be present)
         Guest Section 3      (may not be present)
        One to three levels of hypervisors and guests may be
        reported. When the guest is running second level or
        higher, hypervisor/guest section 1 is filled in by the
        hypervisor running closest to the hardware, with its
        guest filling in the next section, etc. for a maximum
        of 3 levels.
        The format of this response buffer is determined by the
        hypervisor closest to the LPAR. When running directly
        in a partition, there is only one level of hypervisor
        involved in constructing the response so the version is
        the same for all sections. However when running second
        level the version of this block is generally determined
        by the first level hypervisor.
        The formats of the pairs of hypervisor and guest
        sections may differ. The specified version number
        describes the format of that section.
        Use of this function code is authorized in the User
        Directory by OPTION STHYI-UTIL.
        Error responses:
         - CC=3 RC=4  - Unsupported function code
                      - The response buffer is not modified.
         - CC=3 RC=8  - Not authorized for function code
                      - The response buffer is not modified.
        Update Log - see equates under INF1VRSN.
0000    0 Bitstring   64 INF1CHDR       Common section of header mapped
                                        by INFCHDR DSECT.
0000    0 Unsigned     2 INF1VRSN       Response buffer version.
          00000001       INF1V00001     X'0001' INF1V00001 Initial
                                        version. APAR VM66105.
0040   64 Bitstring    1 INF1HFL1       Header Flag Byte 1 These flag
                                        settings indicate the environment
                                        that the instruction was executed
                                        in and may influence the value of
                                        validity bits. The validity bits,
                                        not these flags, should be used
```

```
                                          to determine if a field contains
                                          valid data.
         1... ....      INF1GPDU          X'80' INF1GPDU Global Performance
                                          Data unavailable.
         .1.. ....      INF1STHY          X'40' INF1STHY One or more
                                          hypervisor levels below this
                                          level does not support the STHYI
                                          instruction. When this flag is
                                          set the value of INF1GPDU is not
                                          meaningful because the state of
                                          the Global Performance Data
                                          setting cannot be determined.
         ..1. ....      INF1VSI           X'20' INF1VSI Virtualization
                                          stack is incomplete. This bit
                                          indicates one of 2 cases: 1. One
                                          or more hypervisor levels do not
                                          support STHYI or the FC as
                                          indicated by INF1STHY, INF1FCFC
                                          or INF1FCAU. 2. There were more
                                          than 3 levels of guest/hypervisor
                                          information to report.
         ...1 ....      INF1BASC          X'10' INF1BASC Execution
                                          environment is not within a
                                          logical partition.
         .... 1...      INF1FCFC          X'08' INF1FCFC A lower level
                                          hypervisor supports STHYI but not
                                          this function code. When this
                                          flag is set the value of INF1GPDU
                                          is not meaningful because the
                                          state of the Global Performance
                                          Data setting cannot be
                                          determined.
         .... .1..      INF1FCAU          X'04' INF1FCAU A lower level
                                          hypervisor supports STHYI FC=1
                                          but the guest is not authorized.
                                          When this flag is set the value
                                          of INF1GPDU is not meaningful
                                          because the state of the Global
                                          Performance Data setting cannot
                                          be determined.
0041   65 Bitstring    1 INF1HFL2        Reserved for IBM use
0042   66 Bitstring    1 INF1HVL1        Reserved for IBM use
0043   67 Bitstring    1 INF1HVL2        Reserved for IBM use
0044   68 Unsigned     3 *               Reserved for IBM use
0047   71 Unsigned     1 INF1HYGC        Count of reported hypervisors/
                                          guests. This indicates how many
                                          hypervisor/guest sections are in
                                          the response buffer up to a
                                          maximum of 3.
         00000003       INF1YGMX         3,1,C'X' Maximum Hypervisor/Guest
                                          sections
0048   72 Unsigned     2 INF1MOFF        Offset from start of INF1BK to
                                          Machine Section mapped by INF1MAC
004A   74 Unsigned     2 INF1MLEN        Length of Machine Section in
                                          bytes
004C   76 Unsigned     2 INF1POFF        Offset from start of INF1BK to
                                          Partition Section mapped by
                                          INF1PAR
004E   78 Unsigned     2 INF1PLEN        Len of Partition Section in bytes
0050   80 Bitstring   16 INF1HYG1 (0)    Hypervisor/Guest Header 1
0050   80 Unsigned     2 INF1YOF1        Offset from start of INF1BK to
                                          Hypervisor Section 1, mapped by
                                          INF1HYP.
0052   82 Unsigned     2 INF1YLN1        Length of Hypervisor Section 1 in
                                          bytes.
0054   84 Unsigned     2 INF1YVR1        Version number of this hypervisor
                                          section. Corresponds to the FC=1
                                          version for this hypervisor.
0056   86 Unsigned     2 *               Reserved for IBM use.
0058   88 Unsigned     2 INF1GOF1        Offset from start of INF1BK to
                                          Guest Section 1 mapped by
                                          INF3GST.
005A   90 Unsigned     2 INF1GLN1        Length of Guest Section 1 in
                                          bytes
005C   92 Unsigned     2 INF1GVR1        Version number of this guest
                                          section. Corresponds to the FC=3
                                          version for this hypervisor.
005E   94 Unsigned     2 *               Reserved for IBM use.
0060   96 Bitstring   16 INF1HYG2 (0)    Hypervisor/Guest Header 2
0060   96 Unsigned     2 INF1YOF2        Offset from start of INF1BK to
                                          Hypervisor Section 2 mapped by
                                          INF1HYP.
```

```
0062   98 Unsigned     2 INF1YLN2       Length of Hypervisor Section 2 in
                                        bytes.
0064  100 Unsigned     2 INF1YVR2       Version number of this hypervisor
                                        section. Corresponds to the FC=1
                                        version for this hypervisor.
0066  102 Unsigned     2 *              Reserved for IBM use.
0068  104 Unsigned     2 INF1GOF2       Offset from start of INF1BK to
                                        Guest Section 2 mapped by
                                        INF3GST.
006A  106 Unsigned     2 INF1GLN2       Length of Guest Section 2 in
                                        bytes
006C  108 Unsigned     2 INF1GVR2       Version number of this guest
                                        section. Corresponds to the FC=3
                                        version for this hypervisor.
006E  110 Unsigned     2 *              Reserved for IBM use.
0070  112 Bitstring   16 INF1HYG3 (0)   Hypervisor/Guest Header 3
0070  112 Unsigned     2 INF1YOF3       Offset from start of INF1BK to
                                        Hypervisor Section 3 mapped by
                                        INF1HYP.
0072  114 Unsigned     2 INF1YLN3       Length of Hypervisor Section 3 in
                                        bytes.
0074  116 Unsigned     2 INF1YVR3       Version number of this hypervisor
                                        section. Corresponds to the FC=1
                                        version for this hypervisor.
0076  118 Unsigned     2 *              Reserved for IBM use.
0078  120 Unsigned     2 INF1GOF3       Offset from start of INF1BK to
                                        Guest Section 3 mapped by
                                        INF3GST.
007A  122 Unsigned     2 INF1GLN3       Length of Guest Section 3 in
                                        bytes
007C  124 Unsigned     2 INF1GVR3       Version number of this guest
                                        section. Corresponds to the FC=3
                                        version for this hypervisor.
007E  126 Unsigned     2 *              Reserved for IBM use.
          00000080       INF1BSB1      *-INF1BK Version 1 length in
                                        bytes of the Hypervisor
                                        Environment Information header.
          00000010       INF1BSD1      (*-INF1BK+7)/8 Version 1 length
                                        in doublewords of the Hypervisor
                                        Environment Information header.
          00000080       INF1BSZB      *-INF1BK Length in bytes for
                                        newest version of the Hypervisor
                                        Environment Information header.
          00000010       INF1BSZD      (*-INF1BK+7)/8 Length in
                                        doublewords for newest version of
                                        the Hypervisor Environment
                                        Information header.
```

Function Code X'0001' Response Hypervisor/Guest Entry (INF1YGHD DSECT)

```
     +-------------+-------------+-------------+-------------+
   0 |  INF1YOFF   |  INF1YLEN   |  INF1YVRS   |//////////////|
     +-------------+-------------+-------------+-------------+
   8 |  INF1GOFF   |  INF1GLEN   |  INF1GVRS   |//////////////|
     +-------------+-------------+-------------+-------------+
  10


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF1YGHD       Mappings for STHYI
        Function code X'0001' Response Buffer
        Hypervisor/Guest section description
0000    0 Unsigned     2 INF1YOFF       Offset from start of INF1BK to
                                        Hypervisor Section mapped by
                                        INF1HYP
0002    2 Unsigned     2 INF1YLEN       Length of Hypervisor Section in
                                        bytes
0004    4 Unsigned     2 INF1YVRS       Version number of this hypervisor
                                        section. Corresponds to the FC=1
                                        version for this hypervisor.
0006    6 Unsigned     2 *              Reserved for IBM use.
0008    8 Unsigned     2 INF1GOFF       Offset from start of INF1BK to
                                        Guest Section mapped by INF3GST.
000A   10 Unsigned     2 INF1GLEN       Length of Guest Section in bytes
000C   12 Unsigned     2 INF1GVRS       Version number of this guest
                                        section. Corresponds to the FC=3
                                        version for this hypervisor.
```

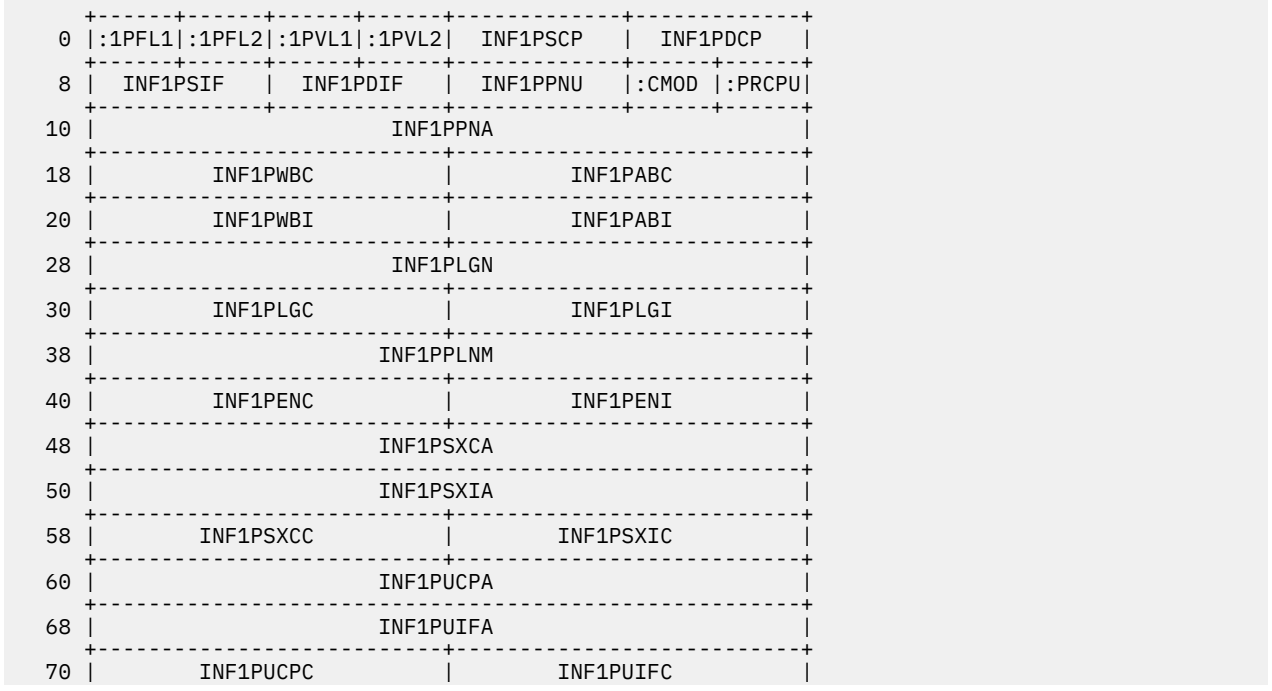
```
000E   14 Unsigned     2 *              Reserved for IBM use.
          00000010       INF1YGB1       *-INF1YGHD Version 1 length in
                                        bytes of the Hypervisor/Guest
                                        entry.
          00000002       INF1YGD1       (*-INF1YGHD+7)/8 Version 1 length
                                        in doublewords of the
                                        Hypervisor/Guest entry.
          00000010       INF1YGSB       *-INF1YGHD Length in bytes for
                                        newest version of the
                                        Hypervisor/Guest entry.
          00000002       INF1YGSD       (*-INF1YGHD+7)/8 Length in
                                        doublewords for newest version of
                                        the Hypervisor/Guest entry.
```

Function Code X'0001' Response Machine Section (INF1MAC DSECT)

```
     +------+------+------+------+-------------+-------------+
   0 |:1MFL1|:1MFL2|:1MVL1|:1MVL2|  INF1MSCP   |  INF1MDCP   |
     +------+------+------+------+-------------+-------------+
   8 |  INF1MSIF   |  INF1MDIF   |         INF1MTYP          |
     +-------------+-------------+---------------------------+
  10 |                       INF1MNAM                        |
     +-------------------------------------------------------+
  18 |                       INF1MMAN                        |
     |                                                       |
     +-------------------------------------------------------+
  28 |                       INF1MSEQ                        |
     |                                                       |
     +---------------------------+---------------------------+
  38 |         INF1MPMA          |///////////////////////////|
     +---------------------------+---------------------------+
  40 |                      INF1MPLNM                        |
     +-------------------------------------------------------+
  48


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF1MAC        Mappings for STHYI
        Function code X'0001' Response Buffer
        Machine Section
0000    0 Bitstring    1 INF1MFL1       Machine Flag Byte 1
          1... ....      INF1MPOOL      X'80' INF1MPOOL Reserved for IBM
                                        use.
0001    1 Bitstring    1 INF1MFL2       Machine Flag Byte 2 Reserved for
                                        IBM use.
0002    2 Bitstring    1 INF1MVL1       Machine Field Validity Byte 1
          1... ....      INF1MPROC      X'80' INF1MPROC Processor Count
                                        Validity When on, INF1MSCP,
                                        INF1MDCP, INF1MSIF, and INF1MDIF
                                        contain valid counts. Off when
                                        virtualization stack is
                                        incomplete (INF1HFL1.INF1VSI on).
          .1.. ....      INF1MIDV       X'40' INF1MIDV Machine ID
                                        Validity This bit being on
                                        indicates that a SYSIB 1.1.1 was
                                        obtained from STSI and
                                        information reported in the
                                        following fields is valid:
                                        INF1MTYP, INF1MMAN, INF1MSEQ, and
                                        INF1MPMA.
          ..1. ....      INF1MNMV       X'20' INF1MNMV Machine Name
                                        Validity This bit being on
                                        indicates that the INF1MNAM field
                                        is valid.
          ...1 ....      INF1MPLNV      X'10' INF1MPLNV Reserved for IBM
                                        use.
0003    3 Bitstring    1 INF1MVL2       Reserved for IBM use
0004    4 Unsigned     2 INF1MSCP       Count of shared CPs configured in
                                        the machine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        INF1MPROC)
0006    6 Unsigned     2 INF1MDCP       Count of dedicated CPs configured
                                        in the machine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        INF1MPROC)
```

```
0008    8 Unsigned     2 INF1MSIF       Count of shared IFLs configured
                                        in the machine or in the physical
                                        partition if the system is
                                        physically partitioned. (Valid if
                                        INF1MPROC)
000A   10 Unsigned     2 INF1MDIF       Count of dedicated IFLs
                                        configured in the machine or in
                                        the physical partition if the
                                        system is physically partitioned.
                                        (Valid if INF1MPROC)
000C   12 EBCDIC       4 INF1MTYP       Machine Type, in EBCDIC format.
                                        This is the machine type reported
                                        by STSI 1.1.1 (Basic Machine
                                        Configuration). (Valid if
                                        INF1MIDV)
0010   16 EBCDIC       8 INF1MNAM       Machine Name, in EBCDIC format.
                                        This is the CPC name associated
                                        with the processor. (Valid if
                                        INF1MNMV)
0018   24 EBCDIC      16 INF1MMAN       Machine Manufacturer, in EBCDIC
                                        format. This is the name of the
                                        manufacturer of the configuration
                                        reported by STSI 1.1.1. (Valid if
                                        INF1MIDV)
0028   40 EBCDIC      16 INF1MSEQ       Sequence Code, in EBCDIC format.
                                        This is the sequence code of the
                                        configuration reported by STSI
                                        1.1.1. (Valid if INF1MIDV)
0038   56 EBCDIC       4 INF1MPMA       Plant of Manufacture, in EBCDIC
                                        format. This is the 4-byte code
                                        reported by STSI 1.1.1. (Valid if
                                        INF1MIDV)
003C   60 Unsigned     4 *              Reserved for IBM use
0040   64 EBCDIC       8 INF1MPLNM      Reserved for IBM use.
          00000048       INF1MSB1      *-INF1MAC Version 1 length in
                                        bytes of the Machine Section.
          00000009       INF1MSD1      (*-INF1MAC+7)/8 Version 1 length
                                        in doublewords of the Machine
                                        Section.
          00000048       INF1MSZB      *-INF1MAC Length in bytes for
                                        newest version of the Machine
                                        Section.
          00000009       INF1MSZD      (*-INF1MAC+7)/8 Length in
                                        doublewords for newest version of
                                        the Machine Section.
```

Function Code X'0001' Response Partition Section (INF1PAR DSECT)



```
     +-------------------------+-------------------------+
  78 |                      INF1PGUCA                      |
     +-------------------------------------------------+
  80 |                      INF1PGUIA                      |
     +-------------------------+-------------------------+
  88 |       INF1PGUCC         |       INF1PGUIC         |
     +-------------------------+-------------------------+
  90 |                      INF1PUTOD                      |
     +-------------------------------------------------+
  98 |                      INF1PACTC                      |
     +-------------------------------------------------+
  A0 |                      INF1PLPTC                      |
     +-------------------------------------------------+
  A8 |                      INF1POLTC                      |
     +-------------------------------------------------+
  B0 |                      INF1PWTTC                      |
     +-------------------------------------------------+
  B8 |                      INF1PMTIC                      |
     +-------------------------------------------------+
  C0 |                      INF1PACTI                      |
     +-------------------------------------------------+
  C8 |                      INF1PLPTI                      |
     +-------------------------------------------------+
  D0 |                      INF1POLTI                      |
     +-------------------------------------------------+
  D8 |                      INF1PWTTI                      |
     +-------------------------------------------------+
  E0 |                      INF1PMTII                      |
     +-------------------------------------------------+
  E8


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF1PAR        Mappings for STHYI
        Function code X'0001' Response Buffer
        Partition Section
0000    0 Bitstring    1 INF1PFL1       Partition Flag Byte 1
          1... ....      INF1PMTE       X'80' INF1PMTE Multithreading
                                        (MT) is enabled
          .1.. ....      INF1PPOOL      X'40' INF1PPOOL Reserved for IBM
                                        use.
          ..1. ....      INF1PWTFL      X'20' INF1PWTFL Wait Completion
                                        flag is set. This flag is not
                                        reliable when INF1PVL1.INF1PUTV
                                        is not set.
0001    1 Bitstring    1 INF1PFL2       Partition Flag Byte 2 reserved
                                        for IBM use.
0002    2 Bitstring    1 INF1PVL1       Partition Field Validity Byte 1
          1... ....      INF1PPRC       X'80' INF1PPRC Processor
                                        Configuration Validity. This bit
                                        being on indicates that INF1PSCP,
                                        INF1PDCP, INF1PSIF, and INF1PDIF
                                        contain valid counts and
                                        INF1PRCPU and INF1PCMOD are
                                        known.
          .1.. ....      INF1PWCV       X'40' INF1PWCV Partition
                                        weight-based capped capacity
                                        validity. This bit being on
                                        indicates that INF1PWBC and
                                        INF1PWBI are valid.
          ..1. ....      INF1PACC       X'20' INF1PACC Partition absolute
                                        capped capacity validity. This
                                        bit being on indicates that
                                        INF1PABC and INF1PABI are valid.
          ...1 ....      INF1PIDV       X'10' INF1PIDV Partition ID
                                        Validity. This bit being on
                                        indicates that a SYSIB 2.2.2 was
                                        obtained from STSI and
                                        information reported in the
                                        following fields is valid:
                                        INF1PPNU, INF1PPNA.
          .... 1...      INF1PLGV       X'08' INF1PLGV LPAR Group
                                        Absolute Capacity Capping
                                        information validity This bit
                                        being on indicates that INF1PLGN,
                                        INF1PLGC, INF1PLGI, INF1PGUCA,
                                        INF1PGUIA, INF1PGUCC, and
                                        INF1PGUIC are valid.
          .... .1..      INF1PPLNV      X'04' INF1PPLNV Reserved for IBM
                                        use.
          .... ..1.      INF1PENV       X'02' INF1PENV Partition
```

```
                                        Entitlement and related
                                        information validity. This bit
                                        being on indicates that INF1PENC,
                                        INF1PENI, INF1PSXCA, INF1PSXIA,
                                        INF1PSXCC, INF1PSXIC, INF1PUCPA,
                                        INF1PUIFA, INF1PUCPC, INF1PUIFC
                                        are valid.
          .... ...1      INF1PUTV       X'01' INF1PUTV Partition Core
                                        Utilization and related
                                        information validity. This bit
                                        being on indicates that
                                        INF1PFL1.INF1PWTFL, INF1PUTOD,
                                        INF1PACTC, INF1PLPTC, INF1POLTC,
                                        INF1PWTTC, INF1PMTIC, INF1PACTI,
                                        INF1PLPTI, INF1POLTI, INF1PWTTI,
                                        and INF1PMTII are valid.
0003    3 Bitstring    1 INF1PVL2      Reserved for IBM use
0004    4 Unsigned     2 INF1PSCP      Count of shared logical CP cores
                                        configured for this partition.
                                        (Valid if INF1PPRC)
0006    6 Unsigned     2 INF1PDCP      Count of dedicated logical CP
                                        cores configured for this
                                        partition. (Valid if INF1PPRC)
0008    8 Unsigned     2 INF1PSIF      Count of shared logical IFL cores
                                        configured for this partition.
                                        (Valid if INF1PPRC)
000A   10 Unsigned     2 INF1PDIF      Count of dedicated logical IFL
                                        cores configured for this
                                        partition. (Valid if INF1PPRC)
000C   12 Unsigned     2 INF1PPNU      Logical Partition Number. This is
                                        the Logical-Partition Number
                                        reported by STSI 2.2.2. (Valid if
                                        INF1PIDV)
000E   14 Unsigned     1 INF1PCMOD     Logical Partition Config Mode
          00000080       INF1PCMGN     X'80' INF1PCMGN General (ESA390)
                                        logical configuration mode
          00000040       INF1PCMLI     X'40' INF1PCMLI Linux logical
                                        configuration mode
          00000020       INF1PCMVM     X'20' INF1PCMVM VM logical
                                        configuration mode
          00000010       INF1PCMCF     X'10' INF1PCMCF CF logical
                                        configuration mode
000F   15 Unsigned     1 INF1PRCPU     Partition primary CPU type. Valid
                                        if INF1PVL1.INF1PPRC is on.
          00000000       INF1PUCCP     X'00' INF1PUCCP General Purpose
                                        (CP)
          00000003       INF1PUCIFL    X'03' INF1PUCIFL Integrated Fac
                                        for Linux (IFL).
0010   16 EBCDIC       8 INF1PPNA      Logical Partition Name, in EBCDIC
                                        format. This is the Logical-
                                        Partition Name reported by STSI
                                        2.2.2. (Valid if INF1PIDV)
      Partition Capacity Caps
0018   24 Unsigned     4 INF1PWBC      Partition weight-based capped
                                        capacity for CPs, a scaled number
                                        where X'00010000' represents one
                                        CPU. Zero if not capped. Cap is
                                        applicable only to shared
                                        processors. (Valid if INF1PWCV)
001C   28 Unsigned     4 INF1PABC      Partition absolute capped
                                        capacity for CPs, a scaled number
                                        where X'00010000' represents one
                                        core. Zero if not capped. Cap is
                                        applicable only to shared
                                        processors. (Valid if INF1PACC)
0020   32 Unsigned     4 INF1PWBI      Partition weight-based capped
                                        capacity for IFLs, a scaled
                                        number where X'00010000'
                                        represents one core. Zero if not
                                        capped. Cap is applicable only to
                                        shared processors. (Valid if
                                        INF1PWCV)
0024   36 Unsigned     4 INF1PABI      Partition absolute capped
                                        capacity for IFLs, a scaled
                                        number where X'00010000'
                                        represents one core. Zero if not
                                        capped. Cap is applicable only to
                                        shared processors. (Valid if
                                        INF1PACC)
0028   40 EBCDIC       8 INF1PLGN      LPAR Group Name (Valid if
                                        INF1PLGV) EBCDIC, padded on right
                                        with blanks when in an LPAR group
```

```
                                        and valid. Binary zeros
                                        otherwise. Only reported when
                                        there is a group cap on CP or IFL
                                        CPU types and the par- tition has
                                        the capped CPU type.
0030   48 Unsigned     4 INF1PLGC       LPAR Group Absolute Capacity
                                        Value for the CP CPU type when
                                        nonzero. Nonzero only when
                                        INF1PLGN is nonzero and a cap is
                                        defined for this LPAR Group for
                                        the CP CPU type. When nonzero,
                                        contains a scaled number where
                                        X'00010000' represents one core.
                                        (Valid if INF1PLGV)
0034   52 Unsigned     4 INF1PLGI       LPAR Group Absolute Capacity
                                        Value for the IFL CPU type when
                                        nonzero. Nonzero only when
                                        INF1PLGN is nonzero and a cap is
                                        defined for this LPAR Group for
                                        the IFL CPU type. When nonzero,
                                        contains a scaled number where
                                        X'00010000' represents one core.
                                        (Valid if INF1PLGV)
0038   56 EBCDIC       8 INF1PPLNM      Reserved for IBM use.
      Processor Entitlement and Available Capacity
        Entitlement values are current values for the partition.  Share
        of extra capacity varies over time.  The sum of the calculated
        values at each hiperdispatch interval and the count of those
        intervals are provided so an average between two STHYI invocations
        can be calculated.  INF1PENC, INF1PENI, INF1PSXCA, INF1PSXCA,
        INF1PSXIA, INF1PSXCC and INF1PSXIC are monotonically increasing
        for the life of the system.
0040   64 Unsigned     4 INF1PENC       Partition Entitled Capacity Value
                                        for the CP CPU type when nonzero.
                                        It is a scaled number where
                                        X'00010000' represents one core.
                                        The value will have been reduced
                                        to the count of CP processors if
                                        the entitlement was larger.
                                        (Valid if INF1PENV)
0044   68 Unsigned     4 INF1PENI       Partition Entitled Capacity Value
                                        for the IFL CPU type when
                                        nonzero. It is a scaled number
                                        where X'00010000' represents one
                                        core. The value will have been
                                        reduced to the count of IFL
                                        processors if the entitlement was
                                        larger. (Valid if INF1PENV)
0048   72 Unsigned     8 INF1PSXCA      Partition calculated Share of
                                        Extra capacity in the interval:
                                        number of physical CP CPUs/cores
                                        which this partition could have
                                        used. A scaled number where
                                        X'00010000' represents one
                                        CPU/core. Aggregated for
                                        INF1PSXCC intervals. (Valid if
                                        INF1PENV)
0050   80 Unsigned     8 INF1PSXIA      Partition calculated Share of
                                        Extra capacity in the interval:
                                        number of physical IFL CPUs/cores
                                        which this partition could have
                                        used. A scaled number where
                                        X'00010000' represents one
                                        CPU/core. Aggregated for
                                        INF1PSXIC intervals. (Valid if
                                        INF1PENV)
0058   88 Unsigned     4 INF1PSXCC      Count of intervals aggregated in
                                        INF1PSXCA. (Valid if INF1PENV)
005C   92 Unsigned     4 INF1PSXIC      Count of intervals aggregated in
                                        INF1PSXIA. (Valid if INF1PENV)
      Partition Utilization
        Partition utilization varies over time. The sum of the
        calculated values at each hiperdispatch interval and the
        count of those intervals are provided so an average
        between two STHYI invocations can be calculated.
        INF1PUCPA, INF1PUIFA, INF1PUCPC and INF1PUIFC are
        monotonically increasing for the life of the system.
0060   96 Unsigned     8 INF1PUCPA      Partition Calculated Utilization
                                        in the interval; number of
                                        physical CP CPUs/cores of this
                                        type which were consumed, a
                                        scaled number where X'00010000'
```

```
                                        represents one CPU/core.
                                        Aggregated for INF1PUCPC
                                        intervals. (Valid if INF1PENV)
0068  104 Unsigned     8 INF1PUIFA      Partition Calculated Utilization
                                        in the interval; number of
                                        physical IFL CPUs/cores of this
                                        type which were consumed, a
                                        scaled number where X'00010000'
                                        represents one CPU/core.
                                        Aggregated for INF1PUIFC
                                        intervals. (Valid if INF1PENV)
0070  112 Unsigned     4 INF1PUCPC      Count of intervals aggregated in
                                        INF1PUCPA. (Valid if INF1PENV)
0074  116 Unsigned     4 INF1PUIFC      Count of intervals aggregated in
                                        INF1PUIFA. (Valid if INF1PENV)
      LPAR Groups Utilization
        LPAR group utilization varies over time. The sum of the
        calculated values at each hiperdispatch interval and the
        count of those intervals are provided so an average
        between two STHYI invocations can be calculated.
        LPAR Groups can be defined for a single CPU type and a
        partition can belong to only 1 group.
        INF1PGUCA, INF1PGUIA, INF1PGUCC and INF1PGUIC are
        monotonically increasing for the life of the system.
0078  120 Unsigned     8 INF1PGUCA      LPAR Group Utilization Value for
                                        the CP CPU type when nonzero.
                                        Nonzero only when INF1PLGN is
                                        nonzero and a cap is defined for
                                        this LPAR Group for the CP CPU
                                        type. When nonzero, contains a
                                        scaled number where X'00010000'
                                        represents one core. Aggregated
                                        over INF1PGUCC intervals. (Valid
                                        if INF1PLGV)
0080  128 Unsigned     8 INF1PGUIA      LPAR Group Utilization Value for
                                        the IFL CPU type when nonzero.
                                        Nonzero only when INF1PLGN is
                                        nonzero and a cap is defined for
                                        this LPAR Group for the IFL CPU
                                        type. When nonzero, contains a
                                        scaled number where X'00010000'
                                        represents one core. Aggregated
                                        over INF1PGUIC intervals. (Valid
                                        if INF1PLGV)
0088  136 Unsigned     4 INF1PGUCC      Count of intervals aggregated in
                                        INF1PGUCA. (Valid if INF1PLGV)
008C  140 Unsigned     4 INF1PGUIC      Count of intervals aggregated in
                                        INF1PGUIA. (Valid if INF1PLGV)
      Core utilization
        The utilization counts are monotonically increasing values.
        The delta between values from 2 STHYI invocations can be
        used to determine the change in the values during the
        interval.  The length of the interval is determined from
        the INF1PUTOD value delta. These fields are valid when
        INF1PVL1.INF1PUTV is set. The values of the fields in this
        section are based on data available at the time specified
        in the INF1PUTOD field.
0090  144 Unsigned     8 INF1PUTOD      Host TOD value at the time when
                                        the partition's core utilization
                                        information was last determined.
                                        (Valid if INF1PUTV)
0098  152 Unsigned     8 INF1PACTC      Aggregate number of microseconds
                                        physical CP cores were assigned
                                        to this partition's logical
                                        cores. (Valid if INF1PUTV)
00A0  160 Unsigned     8 INF1PLPTC      Aggregate number of microseconds
                                        excluding LPAR management time
                                        while physical CP cores were
                                        assigned to this partition's
                                        logical cores. (Valid if
                                        INF1PUTV)
00A8  168 Unsigned     8 INF1POLTC      Aggregate number of microseconds
                                        while logical CP cores of this
                                        partition were online. (Valid if
                                        INF1PUTV)
00B0  176 Unsigned     8 INF1PWTTC      Aggregate number of microseconds
                                        while logical CP cores of this
                                        partition were in wait state. If
                                        MT enabled, accumulates when no
                                        processors of core are running.
                                        Not updated if INF1PWTFL=1.
                                        (Valid if INF1PUTV)
```

```
00B8  184 Unsigned     8 INF1PMTIC      Aggregate number of microseconds
                                        while logical CP cores of this
                                        partition were active but some of
                                        the logical processors of the
                                        core are in any of these states:
                                        enabled wait, disabled wait,
                                        stopped state, check-stop state,
                                        or program interrupt loop. Zero
                                        when MT disabled. (Valid if
                                        INF1PUTV)
00C0  192 Unsigned     8 INF1PACTI      Aggregate number of microseconds
                                        physical IFL cores were assigned
                                        to this partition's logical
                                        cores. (Valid if INF1PUTV)
00C8  200 Unsigned     8 INF1PLPTI      Aggregate number of microseconds
                                        excluding LPAR management time
                                        while physical IFL cores were
                                        assigned to this partition's
                                        logical cores. (Valid if
                                        INF1PUTV)
00D0  208 Unsigned     8 INF1POLTI      Aggregate number of microseconds
                                        while logical IFL cores of this
                                        partition were online. (Valid if
                                        INF1PUTV)
00D8  216 Unsigned     8 INF1PWTTI      Aggregate number of microseconds
                                        while logical IFL cores of this
                                        partition were in wait state. If
                                        MT enabled, accumulates when no
                                        processors of core are running.
                                        Not updated if INF1PWTFL=1.
                                        (Valid if INF1PUTV)
00E0  224 Unsigned     8 INF1PMTII      Aggregate number of microseconds
                                        while logical IFL cores of this
                                        partition were active but some of
                                        the logical processors of the
                                        core are in any of these states:
                                        enabled wait, disabled wait,
                                        stopped state, check-stop state,
                                        or program interrupt loop. Zero
                                        when MT disabled. (Valid if
                                        INF1PUTV)
          000000E8       INF1PSB1      *-INF1PAR Version 1 length in
                                        bytes of the Partition Section.
          0000001D       INF1PSD1      (*-INF1PAR+7)/8 Version 1 length
                                        in doublewords of the Partition
                                        Section.
          000000E8       INF1PSZB      *-INF1PAR Length in bytes for
                                        newest version of the Partition
                                        Section.
          0000001D       INF1PSZD      (*-INF1PAR+7)/8 Length in
                                        doublewords for newest version of
                                        the Partition Section.
```

Function Code X'0001' Response Hypervisor Section (INF1HYP DSECT)

```
    +------+------+------+------+------+------+------+------+
  0 |:1YFL1|:1YFL2|:1YVL1|:1YVL2|:1YTYP|//////|:1YCPT|:1YIFT|
    +------+------+------+------+------+------+------+------+
  8 |                       INF1YSID                        |
    +-------------------------------------------------------+
 10 |                       INF1YCLN                        |
    +-------------+-------------+-------------+-------------+
 18 |  INF1YSCP   |  INF1YDCP   |  INF1YSIF   |  INF1YDIF   |
    +-------------+-------------+-------------+-------------+
 20 |         INF1YASC          |         INF1YASI          |
    +---------------------------+---------------------------+
 28 |         INF1YRSC          |         INF1YRSI          |
    +---------------------------+---------------------------+
 30 |         INF1YLCC          |         INF1YLCI          |
    +---------------------------+------+------+------+------+
 38 |        INF1YMONH          |:1YPKF|:XUSC |:XUSI |//////|
    +---------------------------+------+------+------+------+
 40 |        INF1YPADC          |        INF1YPADI          |
    +---------------------------+---------------------------+
 48 |                      INF1YUTOD                        |
    +-------------------------------------------------------+
 50 |                       INF1YUTC                        |
    +-------------------------------------------------------+
```

```
  58 |                       INF1YUTI                        |
     +-------------------------------------------------------+
  60 |                       INF1YSTC                        |
     +-------------------------------------------------------+
  68 |                       INF1YSTI                        |
     +-------------------------------------------------------+
  70 |                       INF1YWTC                        |
     +-------------------------------------------------------+
  78 |                       INF1YWTI                        |
     +-------------------------------------------------------+
  80 |                       INF1YPTC                        |
     +-------------------------------------------------------+
  88 |                       INF1YPTI                        |
     +-------------------------------------------------------+
  90


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF1HYP        Mappings for STHYI
        Function code X'0001' Response Buffer
        Hypervisor Section
0000    0 Bitstring    1 INF1YFL1       Hypervisor Flag Byte 1
          1... ....      INF1YLMC       X'80' INF1YLMC Consumption method
                                        is used to enforce Limithard
                                        caps.
          .1.. ....      INF1YLMP       X'40' INF1YLMP If on, Limithard
                                        caps use prorated core time for
                                        capping. If off, raw CPU time is
                                        used.
          ..1. ....      INF1YMTE       X'20' INF1YMTE Hypervisor is
                                        MT-enabled.
          ...1 ....      INF1YVRT       X'10' INF1YVRT Vertical
                                        polarization is in use. If off,
                                        horizontal topology is in use,
                                        and INF1YPKF, INF1YXUSC,
                                        INF1YXUSI, INF1YPADC, and
                                        INF1YPADI are not meaningful.
0001    1 Bitstring    1 INF1YFL2       Hypervisor Flag Byte 2 Reserved
                                        for IBM use
0002    2 Bitstring    1 INF1YVL1       Reserved for IBM use
0003    3 Bitstring    1 INF1YVL2       Reserved for IBM use
0004    4 Unsigned     1 INF1YTYP       Hypervisor type
          00000001       INF1YTVM       X'01' INF1YTVM z/VM is the
                                        hypervisor
0005    5 Bitstring    1 *              Reserved for IBM use
0006    6 Bitstring    1 INF1YCPT       Threads in use per CP core. This
                                        value is reported for the current
                                        configuration settings even when
                                        the guest CPUs are not dispatched
                                        on CPs. The value is set only
                                        when SMT enabled as indicated by
                                        INF1YFL1.INF1YMTE.
0007    7 Bitstring    1 INF1YIFT       Threads in use per IFL core. This
                                        value is reported for the current
                                        configuration settings even when
                                        the guest CPUs are not dispatched
                                        on IFLs. The value is set only
                                        when SMT enabled as indicated by
                                        INF1YFL1.INF1YMTE.
0008    8 EBCDIC       8 INF1YSID       System Identifier, in EBCDIC
                                        format, left justified and padded
                                        with blanks. This is the value
                                        generally specified on the
                                        SYSTEM_IDentifier statement in
                                        the system configuration file.
                                        Blank if non-existent.
0010   16 EBCDIC       8 INF1YCLN       Cluster Name, in EBCDIC format,
                                        left justified and padded with
                                        blanks. This is the name on the
                                        SSI statement in the system
                                        configuration file. Blank if the
                                        member isn't in an SSI cluster.
      Core counts
0018   24 Unsigned     2 INF1YSCP       Number of CP cores shared by
                                        non-dedicated CPUs of the guests
                                        of this hypervisor.
001A   26 Unsigned     2 INF1YDCP       Number of CP cores dedicated to
                                        guest CPUs of this hypervisor.
001C   28 Unsigned     2 INF1YSIF       Number of IFL cores shared by
                                        non-dedicated CPUs of the guests
                                        of this hypervisor.
```

```
001E   30 Unsigned     2 INF1YDIF       Number of IFL cores dedicated to
                                        guest CPUs of this hypervisor.
      Total shares
0020   32 Unsigned     4 INF1YASC       Sum of absolute shares for all
                                        logged on CP-dispatched VMDBKs
                                        (excluding dedicated VMDBKs).
                                        A scaled number where X'00010000'
                                        represents one CPU.
0024   36 Unsigned     4 INF1YASI       Sum of absolute shares for all
                                        logged on IFL-dispatched VMDBKs
                                        (excluding dedicated VMDBKs).
                                        A scaled number where X'00010000'
                                        represents one CPU.
0028   40 Unsigned     4 INF1YRSC       Sum of relative shares for all
                                        logged on CP-dispatched VMDBKs
                                        (excluding dedicated VMDBKs).
002C   44 Unsigned     4 INF1YRSI       Sum of relative shares for all
                                        logged on IFL-dispatched VMDBKs
                                        (excluding dedicated VMDBKs).
      Total adds to the limit list
0030   48 Unsigned     4 INF1YLCC       Approximate count of adds to the
                                        limit-list of CP dispatched
                                        VMDBKs. Monotonically increasing.
0034   52 Unsigned     4 INF1YLCI       Approximate count of adds to the
                                        limit-list of IFL dispatched
                                        VMDBKs. Monotonically increasing.
      High-frequency sampling period.
0038   56 Unsigned     4 INF1YMONH      Current Monitor high-frequency
                                        (HF) sampling interval in
                                        hundredths of seconds. Value is 0
                                        when high frequency sampling is
                                        not active.
      Hiperdispatch Settings
003C   60 Unsigned     1 INF1YPKF       HiperDispatch Unparking setting.
                                        Not used if INF1YFL1.INF1YVRT=0.
          00000000       INF1YPLG      X'00' INF1YPLG SRM UNPARKING
                                        LARGE (default)
          00000001       INF1YPMD      X'01' INF1YPMD SRM UNPARKING
                                        MEDIUM
          00000002       INF1YPSM      X'02' INF1YPSM SRM UNPARKING
                                        SMALL
003D   61 Unsigned     1 INF1YXUSC      SRM EXCESSUSE for General (CP)
                                        processor type. Not used if
                                        INF1YFL1.INF1YVRT=0.
          00000010       INF1YXHI      16 INF1YXHI -Indicates the system
                                        should be aggressive in using
                                        unentitled CPU/core capacity.
          00000008       INF1YXMD      8 INF1YXMD -Indicates the system
                                        should be moderately aggressive
                                        in using unentitled CPU/core
                                        capacity.
          00000001       INF1YXLO      1 INF1YXLO -Indicates the system
                                        should not be aggressive in using
                                        unentitled CPU/core capacity.
          00000002       INF1YXNO      2 INF1YXNO -Indicates the system
                                        should not use unentitled
                                        CPU/core capacity.
003E   62 Unsigned     1 INF1YXUSI      SRM EXCESSUSE for IFL processor
                                        type. Not used if
                                        INF1YFL1.INF1YVRT=0.
          00000010       INF1YXHI      16 INF1YXHI -Indicates the system
                                        should be aggressive in using
                                        unentitled CPU/core capacity.
          00000008       INF1YXMD      8 INF1YXMD -Indicates the system
                                        should be moderately aggressive
                                        in using unentitled CPU/core
                                        capacity.
          00000001       INF1YXLO      1 INF1YXLO -Indicates the system
                                        should not be aggressive in using
                                        unentitled CPU/core capacity.
          00000002       INF1YXNO      2 INF1YXNO -Indicates the system
                                        should not use unentitled
                                        CPU/core capacity.
003F   63 Bitstring    1 *              Reserved for IBM use
0040   64 Unsigned     4 INF1YPADC      SRM CPUPAD value, a scaled number
                                        where X'00010000' represents one
                                        General (CP) CPU/core. Not used
                                        if INF1YFL1.INF1YVRT=0.
0044   68 Unsigned     4 INF1YPADI      SRM CPUPAD value, a scaled number
                                        where X'00010000' represents one
                                        IFL CPU/core. Not used if
                                        INF1YFL1.INF1YVRT=0.
```

```
      Logical Processor (not core) Utilization
        The values in these fields are the aggregate of individual
        online processors. When a processor is varied online, the
        value is reset. The length of the interval is determined
        from the INF1YUTOD value delta. INF1YUTOD is set from the
        TOD clock at the start of the calculation of these values.
0048   72 Unsigned     8 INF1YUTOD      TOD value at the start of the
                                        hypervisor logical processor
                                        utilization calculations.
0050   80 Unsigned     8 INF1YUTC       Total CPU time in microseconds
                                        spent on online CP processors
                                        charged to guests. Monotonically
                                        increasing.
0058   88 Unsigned     8 INF1YUTI       Total CPU time in microseconds
                                        spent on online IFL processors
                                        charged to guests. Monotonically
                                        increasing.
0060   96 Unsigned     8 INF1YSTC       Total CPU time in microseconds
                                        spent on online CP processors
                                        charged to SYSTEM. Monotonically
                                        increasing.
0068  104 Unsigned     8 INF1YSTI       Total CPU time in microseconds
                                        spent on online IFL processors
                                        charged to SYSTEM. Monotonically
                                        increasing.
0070  112 Unsigned     8 INF1YWTC       Total elapsed time in
                                        microseconds spent on online CP
                                        processors while in system wait
                                        state. Does not include parked
                                        wait time. Monotonically
                                        increasing.
0078  120 Unsigned     8 INF1YWTI       Total elapsed time in
                                        microseconds spent on online IFL
                                        processors while in system wait
                                        state. Does not include parked
                                        wait time. Monotonically
                                        increasing.
0080  128 Unsigned     8 INF1YPTC       Total elapsed time in
                                        microseconds spent on online CP
                                        processors while in the parked
                                        state. Monotonically increasing.
0088  136 Unsigned     8 INF1YPTI       Total elapsed time in
                                        microseconds spent on online IFL
                                        processors while in the parked
                                        state. Monotonically increasing.
          00000090       INF1YSB1       *-INF1HYP Version 1 length in
                                        bytes of the Hypervisor Section.
          00000012       INF1YSD1       (*-INF1HYP+7)/8 Version 1 length
                                        in doublewords of the Hypervisor
                                        Section.
          00000090       INF1YSZB       *-INF1HYP Length in bytes for
                                        newest version of the Hypervisor
                                        Section.
          00000012       INF1YSZD       (*-INF1HYP+7)/8 Length in
                                        doublewords for newest version of
                                        the Hypervisor Section.
```

Function Code X'0001' Response Guest Section (INF1GST DSECT)

```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF1GST        Mappings for STHYI
        Function code X'0001' Response Buffer
        Guest Section - description of STHYI issuer.
        This matches the guest section of function code 3 as
        defined by the INF3GST DSECT.
0000    0 Bitstring  320 INF1GSTI       Mapped by INF3GST which lacks the
                                        FC=3 header.
          00000140       INF1GSB1       INF3GSB1 Version 1 length in
                                        bytes of the Guest Description.
```

```
        00000028       INF1GSD1       INF3GSD1 Version 1 length in
                                      doublewords of the Guest
                                      Description.
        00000140       INF1GSZB       INF3GSZB Length in bytes for
                                      newest version of the Guest
                                      Description.
        00000028       INF1GSZD       INF3GSZD Length in doublewords
                                      for newest version of the Guest
                                      Description.
```

Function Code X'0002' - Guest List

Function code X'0002' returns a list of logged on guests. Guest virtual machines in the process of logging on, logging off, being forced, or relocating to this system are excluded. Guests enabled for shutdown signals appear in the output during the shutdown signal timeout interval until the actual logoff process begins.

Use of this function code is authorized in the user directory by OPTION STHYI-GUEST.

When function code X'0002' is specified, general register R_2 contains the guest logical address of a response buffer, which must be on a 4 KB boundary or a specification exception is recognized. The size of the response buffer as an unsigned number of 4 KB pages must be specified in general register R_1 (bits 0-15 in ESA/390 or ESA/XC mode, or bits 32-47 in z/Architecture mode). If the value is zero, then a specification exception is recognized. No checking is done for address wrapping. No checking is done for access to pages of the buffer beyond the last page of the actual response data.

When the instruction completes with condition code 0, a list of logged on guests will be returned in the buffer at the guest logical address specified by register R_2.

The possible nonzero return codes for this function code are:

CC=3 RC=4
Unsupported function code. Buffer is unchanged.

CC=3 RC=8
Not authorized for the function code. Buffer is unchanged.

CC=3 RC=20
Response buffer is too small. INFCRQSZ specifies required buffer size. Values are also provided in INFCVRSN, INFCHDLN and INFCTOTL. Other response buffer contents are unpredictable.

Function Code X'0002' Response Buffer Format (INF2BK)

INF2BK DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

The common header section (INFCHDR DSECT) is placed at the beginning of the response buffer and identifies the length of the response.

The common header includes list designation fields for the following:

- List of logged on guests mapped by INF2GST DSECT

Valid lists have nonzero count, offset and length values in the header.

The version number (INF2VRSN) in the function code X'0002' response header can be used to clearly identify whether the function code is at the required service level and can be used by applications to show actual and required version in messages when required support is missing.

Function Code X'0002' Response Header (INF2BK DSECT)

```
    +-------------------------------------------------------+
  0 |                                                       |
    =                       INF2CHDR                        =
    |                                                       |
    +-------------------------------------------------------+
 40
```

```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF2BK         Mappings for STHYI
0000    0 Bitstring   64 INF2CHDR       Common section of header mapped
                                        by INFCHDR DSECT.
0000    0 Unsigned     2 INF2VRSN       Response buffer version.
          00000001       INF2V00001     X'0001' INF2V00001 Initial
                                        version. APAR VM66105.
          00000040       INF2BSB1       *-INF2BK Version 1 length in
                                        bytes of the Guest List header.
          00000008       INF2BSD1       (*-INF2BK+7)/8 Version 1 length
                                        in doublewords of the Guest List
                                        header.
          00000040       INF2BSZB       *-INF2BK Length in bytes for
                                        newest version of the Guest List
                                        header.
          00000008       INF2BSZD       (*-INF2BK+7)/8 Length in
                                        doublewords for newest version of
                                        the Guest List header.
```

Function Code X'0002' Response List Entry (INF2GST DSECT)

```
     +-------------------------------------------------------+
   0 |                       INF2GUID                        |
     +-------------------------------------------------------+
   8 |                       INF2GACN                        |
     +---------------------------+------+------+------+------+
  10 |         INF2GTOD          |:2GFLG|:CMOD |:AFFN |:PRTP |
     +------+--------------------+------+------+------+------+
  18 |:PRDT |////////////////////////////////////////////////|
     +------+------------------------------------------------+
  20


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF2GST        Mappings for STHYI
        Function code X'0002' Response Buffer
        Guest List Entry Description
0000    0 EBCDIC       8 INF2GUID       Guest's userid, in EBCDIC format.
0008    8 EBCDIC       8 INF2GACN       User accounting number in EBCDIC
                                        format.
0010   16 Unsigned     4 INF2GTOD       Bits 0-31 of host TOD at guest
                                        logon.
0014   20 Bitstring    1 INF2GFLG       Guest list entry flag byte 1.
          .... 1...      INF2GLINI      X'08' INF2GLINI Guest identified
                                        itself as running Linux using a
                                        control program identification
                                        interface.
          .... .1..      INF2GLINH      X'04' INF2GLINH Guest may be
                                        running Linux based on
                                        heuristics. Set only if INF2GLINI
                                        is not set.
0015   21 Unsigned     1 INF2GCMOD      Virtual Configuration Mode
          00000080       INF2GCMGN      X'80' INF2GCMGN General (ESA390)
                                        virtual configuration mode
          00000040       INF2GCMLI      X'40' INF2GCMLI Linux virtual
                                        configuration mode
          00000020       INF2GCMVM      X'20' INF2GCMVM VM virtual
                                        configuration mode
          00000010       INF2GCMCF      X'10' INF2GCMCF CF virtual
                                        configuration mode
0016   22 Bitstring    1 INF2GAFFN      Guest CPUAFFINITY settings.
          1... ....      INF2GAFON      X'80' INF2GAFON CPUAFFINITY is
                                        ON, but may be suppressed. If
                                        off, then INF2GAFSUP will not be
                                        on.
          .1.. ....      INF2GAFSUP     X'40' INF2GAFSUP CPUAFFINTY is
                                        suppressed.
0017   23 Bitstring    1 INF2GPRTP      Guest primary virtual CPU type.
          00000000       INF2PUCCP      X'00' INF2PUCCP General Purpose
                                        (CP)
          00000003       INF2PUCIFL     X'03' INF2PUCIFL Integrated Fac
                                        for Linux (IFL).
0018   24 Bitstring    1 INF2GPRDT      Guest primary vCPU dispatch type.
          00000000       INF2PUCCP      X'00' INF2PUCCP General Purpose
                                        (CP)
```

```
          00000003       INF2PUCIFL     X'03' INF2PUCIFL Integrated Fac
                                        for Linux (IFL).
0019   25 Bitstring    7 *              Reserved for IBM use.
          00000020       INF2GSB1       *-INF2GST Version 1 length in
                                        bytes of the Guest List entry.
          00000004       INF2GSD1       (*-INF2GST+7)/8 Version 1 length
                                        in doublewords of the Guest List
                                        entry.
          00000020       INF2GSZB       *-INF2GST Length in bytes for
                                        newest version of the Guest List
                                        entry.
          00000004       INF2GSZD       (*-INF2GST+7)/8 Length in
                                        doublewords for newest version of
                                        the Guest List entry.
```

Function Code X'0003' - Designated Guest Information

Function code X'0003' returns guest CPU resource information useful to an authorized application performing CPU performance monitoring and management.

Use of this function code is authorized in the user directory by OPTION STHYI-GUEST.

When function code X'0003' is specified, general register R_2 contains the guest logical address of a 4 KB response buffer, which must be on a 4 KB boundary or a specification exception is recognized.

This function code provides information for the Guest specified in the 8-byte buffer located at the guest logical address in R_{1+1}. If the buffer is not aligned on a doubleword boundary a specification exception is recognized.

When the instruction completes with condition code 3, the response buffer located by the guest logical address in R_2 is unchanged. When the instruction completes with condition code 0, guest information will be stored into the buffer at the guest logical address specified by register R_2.

The possible nonzero return codes for this function code are:

CC=3 RC=4
Unsupported function code. Buffer is unchanged.

CC=3 RC=8
Not authorized for the function code. Buffer is unchanged.

CC=3 RC=12
Missing or invalid guest name. Buffer is unchanged.

CC=3 RC=16
Named guest is not logged on. Buffer is unchanged.

Function Code X'0003' Response Buffer Format (INF3BK)

INF3BK DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

The common header section (INFCHDR DSECT) is placed at the beginning of the response buffer and identifies the length of the response.

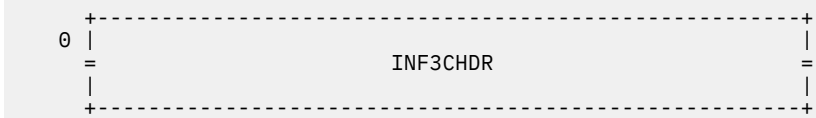
The list designation fields in the common header are stored as zeroes.

The guest information follows the common header at INF3GSTI and is mapped by INF3GST DSECT.

The high-frequency counters starting at field INF3GIWSC report data collected when Monitor high-frequency sampling is active. The values are reset to zero when Monitor commands are issued that deactivate high-frequency sampling, and remain zero while it is inactive. MONITOR SAMPLE RATE is used to set the frequency of the guest state sampling which should be more frequent than the sampling interval. The high-frequency sampling is activated using MONITOR SAMPLE ENABLE USER for the users of interest.

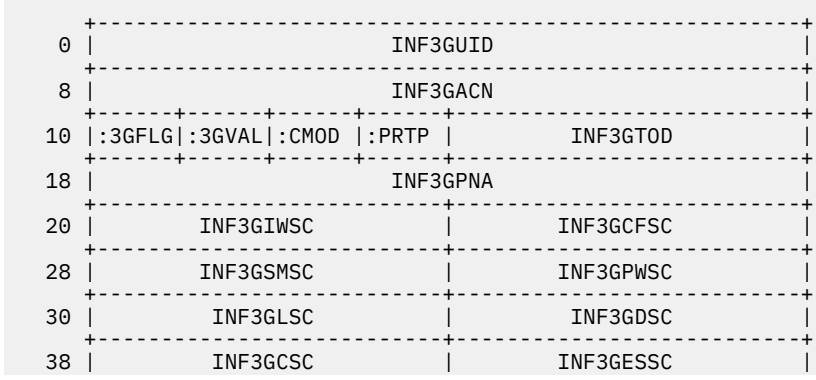
The version number (INF3VRSN) in the function code X'0003' response header can be used to clearly identify whether the function code is at the required service level and can be used by applications to show actual and required version in messages when required support is missing.

Function Code X'0003' Response Header (INF3BK DSECT)




```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF3BK         Mappings for STHYI
        Function code x'0003': Designated Guest Information
        This function code returns the description of the
        specified guest.
        Use of this function code is authorized in the User
        Directory by OPTION STHYI-GUEST.
        Error responses:
         - CC=3 RC=4  - Unsupported function code
                      - The response buffer is not modified.
         - CC=3 RC=8  - Not authorized for function code
                      - The response buffer is not modified.
         - CC=3 RC=12 - Missing or invalid guest name.
                      - The response buffer is not modified.
         - CC=3 RC=16 - Named guest is not logged on.
                      - The response buffer is not modified.
        Update Log - see equates under INF3VRSN.
        Function code X'0003' Response Buffer
0000    0 Bitstring   64 INF3CHDR       Common section of header mapped
                                        by INFCHDR DSECT.
0000    0 Unsigned     2 INF3VRSN       Response buffer version.
          00000001       INF3V00001     X'0001' INF3V00001 Initial
                                        version. APAR VM66105.
0040   64 Bitstring  320 INF3GSTI       Guest Information section mapped
                                        by INF3GST DSECT.
          00000180       INF3BSB1       *-INF3BK Version 1 length in
                                        bytes of the Guest Information
                                        response.
          00000030       INF3BSD1       (*-INF3BK+7)/8 Version 1 length
                                        in doublewords of the Guest
                                        Information response.
          00000180       INF3BSZB       *-INF3BK Length in bytes for
                                        newest version of the Guest
                                        Information response.
          00000030       INF3BSZD       (*-INF3BK+7)/8 Length in
                                        doublewords for newest version of
                                        the Guest Information response.
```

Function Code X'0003' Response Guest Information (INF3GST DSECT)



```
     +--------------------------+--------------------------+
  40 |        INF3GLDSC         |        INF3GDLSC         |
     +--------------------------+--------------------------+
  48 |        INF3GDSSC         |        INF3GIASC         |
     +--------------------------+--------------------------+
  50 |        INF3GTISC         |        INF3GTSSC         |
     +--------------------------+--------------------------+
  58 |        INF3GPASC         |         INF3GOSC         |
     +--------------------------+--------------------------+
  60 |         INF3GTSC         |        INF3GIWSI         |
     +--------------------------+--------------------------+
  68 |        INF3GCFSI         |        INF3GSMSI         |
     +--------------------------+--------------------------+
  70 |        INF3GPWSI         |         INF3GLSI         |
     +--------------------------+--------------------------+
  78 |         INF3GDSI         |         INF3GCSI         |
     +--------------------------+--------------------------+
  80 |        INF3GESSI         |        INF3GLDSI         |
     +--------------------------+--------------------------+
  88 |        INF3GDLSI         |        INF3GDSSI         |
     +--------------------------+--------------------------+
  90 |        INF3GIASI         |        INF3GTISI         |
     +--------------------------+--------------------------+
  98 |        INF3GTSSI         |        INF3GPASI         |
     +--------------------------+--------------------------+
  A0 |         INF3GOSI         |         INF3GTSI         |
     +------+------+-------------+--------------------------+
  A8 |:3CFLG|:AFFN | INF3CMCPU   |//////////////////////////|
     +------+------+-------------+--------------------------+
  B0 |                      INF3CTCPP                       |
     +------------------------------------------------------+
  B8 |                      INF3CTCPS                       |
     +------------------------------------------------------+
  C0 |                      INF3CTCRP                       |
     +------------------------------------------------------+
  C8 |                      INF3CTCRS                       |
     +-------------+-------------+-------------+-------------+
  D0 |  INF3CSCP   |  INF3CDCP   |  INF3CRCP   |//////////////|
     +------+------+------+------+-------------+-------------+
  D8 |:3CCDT|:CSCF |:ISCF |//////|        INF3CCNSC         |
     +------+------+------+------+--------------------------+
  E0 |        INF3CCASC         |        INF3CCMSC         |
     +--------------------------+--------------------------+
  E8 |        INF3CINSC         |        INF3CIASC         |
     +--------------------------+--------------------------+
  F0 |        INF3CIMSC         |//////////////////////////|
     +--------------------------+--------------------------+
  F8 |                      INF3CTIPP                       |
     +------------------------------------------------------+
 100 |                      INF3CTIPS                       |
     +------------------------------------------------------+
 108 |                      INF3CTIRP                       |
     +------------------------------------------------------+
 110 |                      INF3CTIRS                       |
     +-------------+-------------+-------------+-------------+
 118 |  INF3CSIF   |  INF3CDIF   |  INF3CRIF   |//////////////|
     +------+------+------+------+-------------+-------------+
 120 |:3CIDT|:CSIF |:ISIF |//////|        INF3CCNSI         |
     +------+------+------+------+--------------------------+
 128 |        INF3CCASI         |        INF3CCMSI         |
     +--------------------------+--------------------------+
 130 |        INF3CINSI         |        INF3CIASI         |
     +--------------------------+--------------------------+
 138 |        INF3CIMSI         |//////////////////////////|
     +--------------------------+--------------------------+
 140


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF3GST        Mappings for STHYI
        Function code X'0003' Response Buffer
        Guest Description section
        This DSECT is also used for the guest section of the
        function code 1 response buffer.
0000    0 EBCDIC       8 INF3GUID       Guest's userid, in EBCDIC format.
0008    8 EBCDIC       8 INF3GACN       User accounting number in EBCDIC
                                        format.
0010   16 Bitstring    1 INF3GFLG       Guest Flag Byte
          1... ....      INF3GMOB       X'80' INF3GMOB Guest mobility is
                                        enabled.
          .... 1...      INF3GLINI      X'08' INF3GLINI Guest identified
```

```
                                        itself as running Linux using a
                                        control program identification
                                        interface.
          .... .1..      INF3GLINH      X'04' INF3GLINH Guest may be
                                        running Linux based on
                                        heuristics. Set only if INF3GLINI
                                        is not set.
0011   17 Bitstring    1 INF3GVAL       Reserved for IBM use
0012   18 Unsigned     1 INF3GCMOD      Virtual Configuration Mode
          00000080       INF3GCMGN      X'80' INF3GCMGN General (ESA390)
                                        virtual configuration mode
          00000040       INF3GCMLI      X'40' INF3GCMLI Linux virtual
                                        configuration mode
          00000020       INF3GCMVM      X'20' INF3GCMVM VM virtual
                                        configuration mode
          00000010       INF3GCMCF      X'10' INF3GCMCF CF virtual
                                        configuration mode
0013   19 Bitstring    1 INF3GPRTP      Guest primary virtual CPU type.
          00000000       INF3PUCCP      X'00' INF3PUCCP General Purpose
                                        (CP)
          00000003       INF3PUCIFL     X'03' INF3PUCIFL Integrated Fac
                                        for Linux (IFL).
0014   20 Unsigned     4 INF3GTOD       Bits 0-31 of host TOD at guest
                                        logon.
0018   24 EBCDIC       8 INF3GPNA       Resource pool name. Blanks if not
                                        in a Resource Pool.
      High frequency sampler counts
        These counts are the number of times that CP and IFL CPUs of the
        guest are observed in each state.  These counts are reset to zero
        when Monitor commands are issued that disable high-frequency
        sampling.  Otherwise they are monotonically increasing values.
        INF1YMONH provides the high-frequency sampling interval for the
        state testing.  For each virtual CPU type, the order of the count
        fields is the order that the states are tested so that if a vCPU
        is in multiple states, it is counted with the first state that
        matches so that it is counted in one and only one state.  The one
        exception is that when a guest is on the dormant list and SVM wait
        they are counted in both INF3GDLSC and INF3GDSSC for virtual CPs
        and for IFLs, both INF3GDLSI and INF3GDSSI.  The values are
        aggregate values for all guest virtual CPUs of that type.  See
        Monitor record MRUSEINT D4 R4 for additional information.
0020   32 Unsigned     4 INF3GIWSC      I/O Wait Samples for CPs. Times
                                        vCPU found in I/O wait for
                                        asynchronous I/O.
0024   36 Unsigned     4 INF3GCFSC      Console Function Wait samples
                                        for CPs. Times vCPU found in
                                        console function wait.
0028   40 Unsigned     4 INF3GSMSC      Simulation Wait samples for CPs.
                                        Times vCPU found using CPU.
002C   44 Unsigned     4 INF3GPWSC      Page Wait Samples for CPs. Times
                                        vCPU found in page wait.
0030   48 Unsigned     4 INF3GLSC       Limit list Samples for CPs. Times
                                        vCPU found on limit list.
0034   52 Unsigned     4 INF3GDSC       CPU Delay Samples for CPs. Times
                                        vCPU found waiting for CPU.
0038   56 Unsigned     4 INF3GCSC       CPU Using Samples for CPs. Times
                                        vCPU found using CPU.
003C   60 Unsigned     4 INF3GESSC      E-list SVM Wait Samples for CPs.
                                        Times vCPU found on the eligible
                                        list and in SVM wait.
0040   64 Unsigned     4 INF3GLDSC      Loading User Samples for CPs.
                                        Times vCPU considered to be a
                                        loading user and not on the
                                        dormant list.
0044   68 Unsigned     4 INF3GDLSC      Dormant User Samples for CPs.
                                        Times vCPU was found to be on the
                                        dormant list.
0048   72 Unsigned     4 INF3GDSSC      SVM Wait Samples for CPs. Times
                                        vCPU found in the dormant list
                                        and in SVM wait. Also counted in
                                        INF3GDLSC.
004C   76 Unsigned     4 INF3GIASC      I/O Active Samples for CPs. Times
                                        vCPU found with asynchronous I/O
                                        outstanding, causing it to be
                                        left in the Dispatch List.
0050   80 Unsigned     4 INF3GTISC      Test Idle Samples for CPs. Times
                                        vCPU found in test idle and not
                                        in SVM wait.
0054   84 Unsigned     4 INF3GTSSC      Test Idle & SVM Wait Samples for
                                        CPs. Times vCPU found in test
                                        idle and SVM wait.
0058   88 Unsigned     4 INF3GPASC      Page Fault Active Samples for
```

```
                                          CPs. Times vCPU had active page
                                          fault requests active but was not
                                          in page wait.
005C   92 Unsigned     4 INF3GOSC        Other Samples for CPs. Times vCPU
                                          found in other state.
0060   96 Unsigned     4 INF3GTSC        Total Samples for CPs. Times vCPU
                                          state sampled.
0064  100 Unsigned     4 INF3GIWSI       I/O Wait Samples for IFLs. Times
                                          vCPU found in I/O wait for
                                          asynchronous I/O.
0068  104 Unsigned     4 INF3GCFSI       Console Function Wait samples
                                          for IFLs. Times vCPU found in
                                          console function wait.
006C  108 Unsigned     4 INF3GSMSI       Simulation Wait samples for IFLs.
                                          Times vCPU found using CPU.
0070  112 Unsigned     4 INF3GPWSI       Page Wait Samples for IFLs. Times
                                          vCPU found in page wait.
0074  116 Unsigned     4 INF3GLSI        Limit list Samples for IFLs.
                                          Times vCPU found on limit list.
0078  120 Unsigned     4 INF3GDSI        CPU Delay Samples for IFLs. Times
                                          vCPU found waiting for CPU.
007C  124 Unsigned     4 INF3GCSI        CPU Using Samples for IFLs. Times
                                          vCPU found using CPU.
0080  128 Unsigned     4 INF3GESSI       E-list SVM Wait Samples for IFLs.
                                          Times vCPU found on the eligible
                                          list and in SVM wait.
0084  132 Unsigned     4 INF3GLDSI       Loading User Samples for IFLs.
                                          Times vCPU considered to  be a
                                          loading user and not on the
                                          dormant list.
0088  136 Unsigned     4 INF3GDLSI       Dormant User Samples for IFLs.
                                          Times vCPU was found to be on the
                                          dormant list.
008C  140 Unsigned     4 INF3GDSSI       SVM Wait Samples for IFLs. Times
                                          vCPU found in the dormant list
                                          and in SVM wait. Also counted in
                                          INF3GDLSC.
0090  144 Unsigned     4 INF3GIASI       I/O Active Samples for IFLs.
                                          Times vCPU found with
                                          asynchronous I/O outstanding,
                                          causing it to be left in the
                                          Dispatch List.
0094  148 Unsigned     4 INF3GTISI       Test Idle Samples for IFLs. Times
                                          vCPU found in test idle and not
                                          in SVM wait.
0098  152 Unsigned     4 INF3GTSSI       Test Idle & SVM Wait Samples for
                                          IFLs. Times vCPU found in test
                                          idle and SVM wait.
009C  156 Unsigned     4 INF3GPASI       Page Fault Active Samples for
                                          IFLs. Times vCPU had active page
                                          fault requests active but was not
                                          in page wait.
00A0  160 Unsigned     4 INF3GOSI        Other Samples for IFLs. Times
                                          vCPU found in other state.
00A4  164 Unsigned     4 INF3GTSI        Total Samples for IFLs. Times
                                          vCPU state sampled.
      CPU resources information
00A8  168 Bitstring    1 INF3CFLG        Guest CPU Flag Byte
          .1.. ....      INF3CMCT        X'40' INF3CMCT Guest has multiple
                                          CPU types
          ..1. ....      INF3CVCT        X'20' INF3CVCT Virtual CPs are
                                          thread dispatched
          ...1 ....      INF3CVIT        X'10' INF3CVIT Virtual IFLs are
                                          thread dispatched
00A9  169 Bitstring    1 INF3CAFFN       Guest CPUAFFINITY settings.
          1... ....      INF3CAFON       X'80' INF3CAFON CPUAFFINITY is
                                          ON, but may be suppressed. If
                                          off, then INF3CAFSUP will not be on.
          .1.. ....      INF3CAFSUP      X'40' INF3CAFSUP CPUAFFINTY is
                                          suppressed.
00AA  170 Unsigned     2 INF3CMCPU       Maximum number of guest CPUs
                                          based on user directory setting.
00AC  172 Unsigned     4 *               Reserved for IBM use.
00B0  176 Unsigned     8 INF3CTCPP       Total virtual and simulation time
                                          while running a virtual CP on a
                                          primary CPU, in prorated core
                                          time microseconds. Only provided
                                          when SMT is enabled
                                          (INF1YFL1.INF1YMTE).
                                          Monotonically increasing.
00B8  184 Unsigned     8 INF3CTCPS       Total virtual and simulation time
                                          while running a virtual CP on a
```

```
                                        2ndary CPU, in prorated core time
                                        microseconds. Only provided when
                                        SMT is enabled
                                        (INF1YFL1.INF1YMTE).
                                        Monotonically increasing.
00C0  192 Unsigned     8 INF3CTCRP      Total virtual and simulation time
                                        while running a virtual CP on a
                                        primary CPU, in raw core time
                                        microseconds. Monotonically
                                        increasing.
00C8  200 Unsigned     8 INF3CTCRS      Total virtual and simulation time
                                        while running a virtual CP on a
                                        secondary CPU, in raw core time
                                        microseconds. Monotonically
                                        increasing.
00D0  208 Unsigned     2 INF3CSCP       Number of guest shared CPs.
00D2  210 Unsigned     2 INF3CDCP       Number of guest dedicated CPs.
00D4  212 Unsigned     2 INF3CRCP       Number of non-stopped guest CPs.
00D6  214 Unsigned     2 *              Reserved for IBM use.
00D8  216 Unsigned     1 INF3CCDT       Dispatch type for guest CPs This
                                        field is valid if INF3SCPS,
                                        INF3CDCP or INF3CRCP is greater
                                        than zero. Always INF3PUCCP.
00D9  217 Bitstring    1 INF3CCSCF      Current share flags for CPs.
          .1.. ....      INF3CCLHC      X'40' INF3CCLHC Current max share
                                        for CP-dispatched vCPUs is
                                        LIMITHARD if on. Max share for
                                        type CP is LIMITSOFT if off.
          ..1. ....      INF3CCNAC      X'20' INF3CCNAC Current normal
                                        share for CP-dispatched vCPUs is
                                        ABSOLUTE if on. Normal share for
                                        type CP is RELATIVE if off.
          ...1 ....      INF3CCMAC      X'10' INF3CCMAC Current max share
                                        for CP-dispatched vCPUs is
                                        ABSOLUTE if on. Max share for
                                        type CP is RELATIVE if off.
00DA  218 Bitstring    1 INF3CISCF      Initial share flags for CPs.
          .1.. ....      INF3CILHC      X'40' INF3CILHC Initial max share
                                        for CP-dispatched vCPUs is
                                        LIMITHARD if on. Max share for
                                        type CP is LIMITSOFT if off. This
                                        is the setting at logon.
          ..1. ....      INF3CINAC      X'20' INF3CINAC Initial normal
                                        share for CP-dispatched vCPUs is
                                        ABSOLUTE if on. Normal share for
                                        type CP is RELATIVE if off. This
                                        is the setting at logon.
          ...1 ....      INF3CIMAC      X'10' INF3CIMAC Initial max share
                                        for CP-dispatched vCPUs is
                                        ABSOLUTE if on. Max share for
                                        type CP is RELATIVE if off. This
                                        is the setting at logon.
00DB  219 Bitstring    1 *              Reserved for IBM use
00DC  220 Unsigned     4 INF3CCNSC      Current normal relative share for
                                        CP-dispatched vCPUs. The value is
                                        zero when the user has an ABSOLUTE
                                        share value for CPs, or has a
                                        dedicated virtual CPU, or is in
                                        the process of being logged off.
00E0  224 Unsigned     4 INF3CCASC      Current ABSOLUTE share
for                                         CP-dispatched vCPUs. Unit
of                                     value is a hexadecimal
factor                                     scaled 16 bits. For
example,                                     X'00010000' =
1.00(100%),                                     X'0000C000' =
0.75(75%),                                    X'00008000' = 0.50(50%),
etc.                                     Set to zero when the user
has                                         a RELATIVE share value for CPs.
00E4  228 Unsigned     4 INF3CCMSC      Current Max share for CP-
dispatched                                        vCPUs. This value is set
to                                     zero if no Max share exists
for                                     CPs. If the Max share is
Absolute,                                     then the units of the
value                                    are the same as for
INFCCASC.                                    Otherwise as for INF3CCNSC.
00E8  232 Unsigned     4 INF3CINSC      Initial (logon) normal relative
                                        share for CP-dispatched vCPUs.
                                        This value will be zero when the
                                        initial (logon) share is ABSOLUTE
                                        for CPs, or the user has a
                                        dedicated virtual CPU, or is in
                                        the process of being logged off.
```

```
00EC  236 Unsigned     4 INF3CIASC      Initial (logon) ABSOLUTE share
                                        for CP-dispatched vCPUs. Unit of
                                        value is a hexadecimal factor
                                        scaled 16 bits. For example,
                                        X'00010000' = 1.00(100%),
                                        X'0000C000' = 0.75(75%),
                                        X'00008000' = 0.50(50%), etc.
                                        This value will be zero when the
                                        initial (logon) share for CPs
                                        is relative.
00F0  240 Unsigned     4 INF3CIMSC      Initial (logon) Max share for
                                        CP-dispatched vCPUs. This value
                                        will be zero if no initial
                                        (logon) Max share exists for CPs.
                                        If the Max share is Absolute,
                                        then the units of the value
                                        are the same as for INF3CIASC.
                                        Otherwise as for INF3CINSC.
00F4  244 Unsigned     4 *              Reserved for IBM use.
00F8  248 Unsigned     8 INF3CTIPP      Total virtual and simulation time
                                        while running a virtual IFL on a
                                        primary CPU, in prorated core
                                        time microseconds. Only provided
                                        when SMT is enabled
                                        (INF1YFL1.INF1YMTE).
                                        Monotonically increasing.
0100  256 Unsigned     8 INF3CTIPS      Total virtual and simulation time
                                        while running a virtual IFL on a
                                        2ndary CPU, in prorated core time
                                        microseconds. Only provided when
                                        SMT is enabled
                                        (INF1YFL1.INF1YMTE).
                                        Monotonically increasing.
0108  264 Unsigned     8 INF3CTIRP      Total virtual and simulation time
                                        while running a virtual IFL on a
                                        primary CPU, in raw core time
                                        microseconds. Monotonically
                                        increasing.
0110  272 Unsigned     8 INF3CTIRS      Total virtual and simulation time
                                        while running a virtual IFL on a
                                        secondary CPU, in raw core time
                                        microseconds. Monotonically
                                        increasing.
0118  280 Unsigned     2 INF3CSIF       Number of guest shared IFLs.
011A  282 Unsigned     2 INF3CDIF       Number of guest dedicated IFLs.
011C  284 Unsigned     2 INF3CRIF       Number of non-stopped guest IFLs.
011E  286 Unsigned     2 *              Reserved for IBM use.
0120  288 Unsigned     1 INF3CIDT       Dispatch type for guest IFLs.
                                        This field is valid if INF3SCPS,
                                        INF3CDCP or INF3CRCP is greater
                                        than zero.
          00000000       INF3PUCCP      X'00' INF3PUCCP General Purpose
                                        (CP)
          00000003       INF3PUCIFL     X'03' INF3PUCIFL Integrated Fac
                                        for Linux (IFL).
0121  289 Bitstring    1 INF3CCSIF      Current share flags for IFLs.
          .1.. ....      INF3CCLHI      X'40' INF3CCLHI Current max share
                                        for IFL-dispatched vCPUs is
                                        LIMITHARD if on. Max share for
                                        type IFL is LIMITSOFT if off.
          ..1. ....      INF3CCNAI      X'20' INF3CCNAI Current normal
                                        share for IFL-dispatched vCPUs is
                                        ABSOLUTE if on. Normal share for
                                        type IFL is RELATIVE if off.
          ...1 ....      INF3CCMAI      X'10' INF3CCMAI Current max share
                                        for IFL-dispatched vCPUs is
                                        ABSOLUTE if on. Max share for
                                        type IFL is RELATIVE if off.
0122  290 Bitstring    1 INF3CISIF      Initial share flags for IFLs.
          .1.. ....      INF3CILHI      X'40' INF3CILHI Initial max share
                                        for IFL-dispatched vCPUs is
                                        LIMITHARD if on. Max share for
                                        type IFL is LIMITSOFT if off.
                                        This is the setting at logon.
          ..1. ....      INF3CINAI      X'20' INF3CINAI Initial normal
                                        share for IFL-dispatched vCPUs is
                                        ABSOLUTE if on. Normal share for
                                        type IFL is RELATIVE if off. This
                                        is the setting at logon.
          ...1 ....      INF3CIMAI      X'10' INF3CIMAI Initial max share
                                        for IFL-dispatched vCPUs is
                                        ABSOLUTE if on. Max share for
```

```
                                        type IFL is RELATIVE if off. This
                                        is the setting at logon.
0123  291 Bitstring    1 *              Reserved for IBM use
0124  292 Unsigned     4 INF3CCNSI      Current normal relative share
for                                         IFL-dispatched vCPUs. The value
is                                        zero when the user has an
ABSOLUTE                                        share value for IFLs, or has
a                                       dedicated virtual CPU, or is
in                                       the process of being logged off.
0128  296 Unsigned     4 INF3CCASI      Current ABSOLUTE share
for                                         IFL-dispatched vCPUs. Unit
of                                        value is a hexadecimal
factor                                        scaled 16 bits. For
example,                                    X'00010000' =
1.00(100%),                                    X'0000C000' =
0.75(75%),                                   X'00008000' = 0.50(50%),
etc.                                    The value is zero when the
user                                    has a RELATIVE share value for IFLs.
012C  300 Unsigned     4 INF3CCMSI      Current Max share for IFL-
dispatched                                     vCPUs. This value will be
zero                                    if no Max share exists for
IFLs.                                    If the Max share is
Absolute,                                    then the units of the
value                                   are the same as for
INF3CCASI.                                   Otherwise as for INF3CCNSI.
0130  304 Unsigned     4 INF3CINSI      Initial (logon) normal relative
                                        share for IFL-dispatched vCPUs.
                                        This value will be zero when the
                                        initial (logon) share is ABSOLUTE
                                        for IFLs, or the user has a
                                        dedicated virtual CPU, or is in
                                        the process of being logged off.
0134  308 Unsigned     4 INF3CIASI      Initial (logon) ABSOLUTE share
                                        for IFL-dispatched vCPUs. Unit of
                                        value is a hexadecimal factor
                                        scaled 16 bits. For example,
                                        X'00010000' = 1.00(100%),
                                        X'0000C000' = 0.75(75%),
                                        X'00008000' = 0.50(50%), etc.
                                        This value will be zero when the
                                        initial (logon) share is relative
                                        for IFLs.
0138  312 Unsigned     4 INF3CIMSI      Initial (logon) Max share for
                                        IFL-dispatched vCPUs. This value
                                        will be zero if no initial
                                        (logon) Max share exists for IFLs.
                                        If the Max share is Absolute,
                                        then the units of the value
                                        are the same as for INF3CIASI.
                                        Otherwise as for INF3CINSI.
013C  316 Unsigned     4 *              Reserved for IBM use.
          00000140       INF3GSB1       *-INF3GST Version 1 length in
                                        bytes of the Guest Description.
          00000028       INF3GSD1       (*-INF3GST+7)/8 Version 1 length
                                        in doublewords of the Guest
                                        Description.
          00000140       INF3GSZB       *-INF3GST Length in bytes for
                                        newest version of the Guest
                                        Description.
          00000028       INF3GSZD       (*-INF3GST+7)/8 Length in
                                        doublewords for newest version of
                                        the Guest Description.
```

Function Code X'0004' - Resource Pool List

Function code X'0004' returns a list of defined resource pools.

Use of this function code is authorized in the user directory by OPTION STHYI-RESPOOL.

When function code X'0004' is specified, general register R_2 contains the guest logical address of a response buffer, which must be on a 4 KB boundary or a specification exception is recognized. The size of the response buffer as an unsigned number of 4 KB pages must be specified in general register R_1 (bits 0-15 in ESA/390 or ESA/XC mode, or bits 32-47 in z/Architecture mode). If the value is zero, then a specification exception is recognized. No checking is done for address wrapping. No checking is done for access to pages of the buffer beyond the last page of the actual response data.

When the instruction completes with condition code 0, a list of defined Resource Pools will be returned in the buffer at the guest logical address specified by register R_2.

The possible nonzero return codes for this function code are:

CC=3 RC=4
Unsupported function code. Buffer is unchanged.

CC=3 RC=8
Not authorized for the function code. Buffer is unchanged.

CC=3 RC=20
Response buffer is too small. INFCRQSZ specifies required buffer size. Values are also provided in INFCVRSN, INFCHDLN and INFCTOTL. Other response buffer contents are unpredictable.

Function Code X'0004' Response Buffer Format (INF4BK)

INF4BK DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

The common header section (INFCHDR DSECT) is placed at the beginning of the response buffer and identifies the length of the response.

The common header includes list designation fields for the following:

- List of defined Resource Pools mapped by INF4POOL DSECT

Valid lists have nonzero count, offset and length values in the header.

The version number (INF4VRSN) in the function code X'0004' response header can be used to clearly identify whether the function code is at the required service level and can be used by applications to show actual and required version in messages when required support is missing.

Function Code X'0004' Response Header (INF4BK DSECT)

```
     +-------------------------------------------------------+
   0 |                                                       |
     =                       INF4CHDR                        =
     |                                                       |
     +-------------------------------------------------------+
  40


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF4BK         Mappings for STHYI
        Function code x'0004': Resource Pool List
        This function code returns a list of all Resource
        Pools defined on the system.
        This function code requires the length of the buffer as
        a number of 4K pages to be specified. See the STHYI
        documentation for details of how this is specified.
        The response buffer for function code x'0004' consists
        of 2 sections of data:
        - Header Section (INF2BK DSECT, imbeds INFCHDR DSECT)
          - use offset to Resource Pool list, length of entry,
            and count of entries to process the list.
          - The offset in INFCLSOF is from the beginning
            of the INF4BK.
          - if the count is zero there is no list so the offset
            and length will be zero.
        - Resource Pool list. The list is returned as an array
          with the count of entries, offset to the first entry
          and the entry length defined by the common header.
        - The array entry structure is defined by INF4POOL DSECT.
        Use of this function code is authorized in the User
        Directory by OPTION STHYI-RESPOOL.
        Error responses:
         - CC=3 RC=4  - Unsupported function code
                      - The response buffer is not modified.
         - CC=3 RC=8  - Not authorized for function code
                      - The response buffer is not modified.
         - CC=3 RC=20 - Response buffer is too small.
                      - INFCRQSZ specifies required buffer size.
```

```
                      - Values are also provided in INFCVRSN,
                        INFCHDLN and INFCTOTL. Other response
                        buffer contents are unpredictable.
        Update Log - see equates under INF4VRSN.
        Function code X'0004' Response Buffer
        Header Section
0000    0 Bitstring   64 INF4CHDR       Common section of header mapped
                                        by INFCHDR DSECT.
0000    0 Unsigned     2 INF4VRSN       Response buffer version.
          00000001       INF4V00001     X'0001' INF4V00001 Initial
                                        version. APAR VM66105.
          00000040       INF4BSB1       *-INF4BK Version 1 length in
                                        bytes of the Resource Pool List
                                        header.
          00000008       INF4BSD1       (*-INF4BK+7)/8 Version 1 length
                                        in doublewords of the Resource
                                        Pool List header.
          00000040       INF4BSZB       *-INF4BK Length in bytes for
                                        newest version of the Resource
                                        Pool List header.
          00000008       INF4BSZD       (*-INF4BK+7)/8 Length in
                                        doublewords for newest version of
                                        the Resource Pool List header.
```

Function Code X'0004' Response List Entry (INF4POOL DSECT)

```
     +-------------------------------------------------------+
   0 |                       INF4PNAM                        |
     +-------------------------------------------------------+
   8 |                       INF4PCRE                        |
     +-------------------------------------------------------+
  10


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF4POOL       Mappings for STHYI
        Function code X'0004' Response Buffer
        Resource Pool List Entry Description
0000    0 EBCDIC       8 INF4PNAM       Resource Pool name.
0008    8 EBCDIC       8 INF4PCRE       Resource Pool creator userid.
          00000010       INF4PSB1       *-INF4POOL Version 1 length in
                                        bytes of the Resource Pool List
                                        entry.
          00000002       INF4PSD1       (*-INF4POOL+7)/8 Version 1 length
                                        in doublewords of the Resource
                                        Pool List entry.
          00000010       INF4PSZB       *-INF4POOL Length in bytes for
                                        newest version of the Resource
                                        Pool List entry.
          00000002       INF4PSZD       (*-INF4POOL+7)/8 Length in
                                        doublewords for newest version of
                                        the Resource Pool List entry.
```

Function Code X'0005' - Designated Resource Pool Information

Function code X'0005' returns a resource pool description for the specified resource pool.

Use of this function code is authorized in the user directory by OPTION STHYI-RESPOOL.

When function code X'0005' is specified, general register R_2 contains the guest logical address of a 4 KB response buffer, which must be on a 4 KB boundary or a specification exception is recognized.

This function code provides information for the resource pool specified in the 8-byte buffer located at the guest logical address in R_{1+1}. If the buffer is not aligned on a doubleword boundary a specification exception is recognized.

When the instruction completes with condition code 3, the response buffer located by the guest logical address in R_2 is unchanged. When the instruction completes with condition code 0, guest CPU resource information will be stored into the buffer at the guest logical address specified by register R_2.

The possible nonzero return codes for this function code are:

CC=3 RC=4
Unsupported function code. Buffer is unchanged.

CC=3 RC=8
Not authorized for the function code. Buffer is unchanged.

CC=3 RC=12
Missing or invalid Resource Pool Name. Buffer is unchanged.

CC=3 RC=16
Named Resource Pool does not exist. Buffer is unchanged.

Function Code X'0005' Response Buffer Format (INF5BK)

INF5BK DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

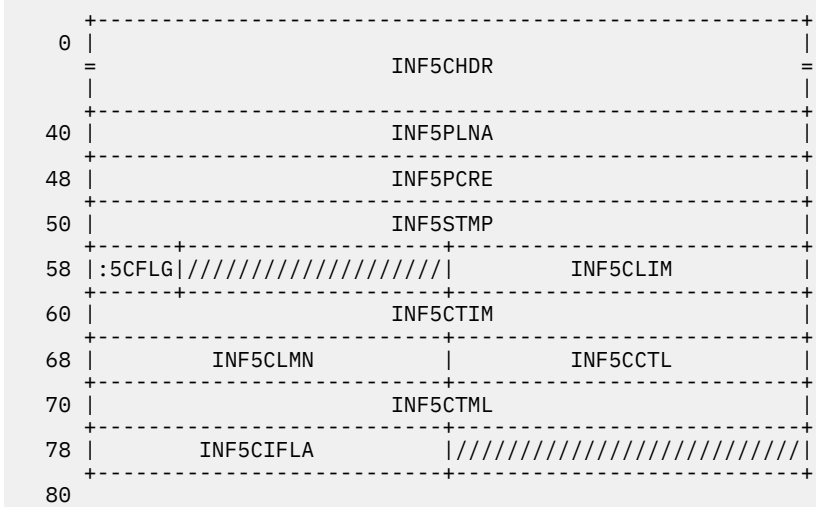
The common header section (INFCHDR DSECT) is placed at the beginning of the response buffer and identifies the length of the response.

The list designation fields in the common header are stored as zeroes.

The resource pool information follows the common header as described by the INF5BK DSECT.

The version number (INF5VRSN) in the function code X'0005' response header can be used to clearly identify whether the function code is at the required service level and can be used by applications to show actual and required version in messages when required support is missing.

Function Code X'0005' Response Header (INF5BK DSECT)




```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF5BK         Mappings for STHYI
        Function code x'0005': Designated Resource Pool Info
        This function code returns the description of the
        specified Resource Pool.
        Use of this function code is authorized in the User
        Directory by OPTION STHYI-RESPOOL.
        Error responses:
         - CC=3 RC=4  - Unsupported function code
                      - The response buffer is not modified.
         - CC=3 RC=8  - Not authorized for function code
                      - The response buffer is not modified.
         - CC=3 RC=12 - Missing or invalid Resource Pool Name.
                      - The response buffer is not modified.
         - CC=3 RC=16 - Named Resource Pool does not exist.
                      - The response buffer is not modified.
        Update Log - see equates under INF5VRSN.
        Function code X'0005' Response Buffer
0000    0 Bitstring   64 INF5CHDR       Common section of header mapped
                                        by INFCHDR DSECT.
```

```
0000    0 Unsigned     2 INF5VRSN       Response buffer version.
          00000001       INF5V00001     X'0001' INF5V00001 Initial
                                        version. APAR VM66105.
0040   64 EBCDIC       8 INF5PLNA       Resource pool name.
0048   72 EBCDIC       8 INF5PCRE       Resource Pool creator's user ID.
0050   80 Unsigned     8 INF5STMP       Host TOD value at the time of the
                                        last change to the Resource Pool
                                        definition.
      Resource Pool CPU Limits and usage information.
        A CPU can have a capacity limit for either CP or IFL
        virtual CPU type. That limit can be specified as either
        a CAPACITY or LIMITHARD limit. Therefore never more than
        one of INF5CCCL,INF5CCCC,INF5CICL or INF5CICC will be on.
0058   88 Bitstring    1 INF5CFLG       Resource Pool CPU Capping Flags
          1... ....      INF5CCLC       X'80' INF5CCLC Resource Pool's CP
                                        virtual type has LIMITHARD cap.
          .1.. ....      INF5CCCC       X'40' INF5CCCC Resource Pool's CP
                                        virtual type has CAPACITY cap.
          ..1. ....      INF5CILC       X'20' INF5CILC Resource Pool's
                                        IFL virtual type has LIMITHARD
                                        cap.
          ...1 ....      INF5CICC       X'10' INF5CICC Resource Pool's
                                        IFL virtual type has CAPACITY
                                        cap.
          .... 1...      INF5CPCT       X'08' INF5CPCT Resource Pool uses
                                        prorated core time.
          .... .1..      INF5CSUP       X'04' INF5CSUP IFL CPUAFFINITY is
                                        suppressed. Not set for CP
                                        resource pools.
0059   89 Bitstring    3 *              Reserved for IBM use.
005C   92 Unsigned     4 INF5CLIM       Resource pool maximum share for
                                        shared virtual CPUs. Scaled
                                        number where X'00010000'
                                        represents 1 core if the cap is a
                                        CAPACITY cap, or 100% of the real
                                        processors if the cap is a
                                        LIMITHARD cap. Zero if not
                                        capped. INF5CFLG specifies the
                                        capped CPU type.
0060   96 Unsigned     8 INF5CTIM       Total Time consumed by guests
                                        vCPUs in the Resource Pool since
                                        it was created. Monotonically
                                        increasing in microseconds. If MT
                                        is enabled, this field contains
                                        prorated core time; Otherwise,
                                        this field contains raw CPU time.
0068  104 Unsigned     4 INF5CLMN       Total number of times the
                                        Resource Pool was limited since
                                        it was created. Monotonically
                                        increasing. Incremented at end of
                                        group limit.
006C  108 Unsigned     4 INF5CCTL       Total number of times vCPUs of
                                        members of the resource pool were
                                        limited since it was created.
                                        Monotonically increasing. Updated
                                        at the start of the vCPU limit.
0070  112 Unsigned     8 INF5CTML       Total time limited for vCPUs of
                                        all members of the Resource Pool
                                        since it was created.
                                        Monotonically increasing in
                                        microseconds. Updated at the
                                        start of the vCPU limit.
0078  120 Unsigned     4 INF5CIFLA      IFL CPUAFFINITY toggle sequence
                                        number. It is incremented when
                                        IFL CPUAFFINITY suppression is
                                        turned on or off for pools with
                                        members.
007C  124 Unsigned     4 *              Reserved for IBM use.
          00000080       INF5BSB1       *-INF5BK Version 1 length in
                                        bytes of the Resource Pool
                                        Information response.
          00000010       INF5BSD1       (*-INF5BK+7)/8 Version 1 length
                                        in doublewordss of the Resource
                                        Pool Information response.
          00000080       INF5BSZB       *-INF5BK Length in bytes for
                                        newest version of the Resource
                                        Pool Information response.
          00000010       INF5BSZD       (*-INF5BK+7)/8 Length in
                                        doublewords for newest version of
                                        the Resource Pool Information
                                        response
```

Function Code X'0006' - Resource Pool Member List

Function code X'0006' returns a list of resource pool members for the specified resource pool.

Use of this function code is authorized in the user directory by OPTION STHYI-RESPOOL.

When function code X'0006' is specified, general register R_2 contains the guest logical address of a response buffer, which must be on a 4 KB boundary or a specification exception is recognized. The size of the response buffer as an unsigned number of 4 KB pages must be specified in general register R_1 (bits 0-15 in ESA/390 or ESA/XC mode, or bits 32-47 in z/Architecture mode). If the value is zero, then a specification exception is recognized. No checking is done for address wrapping. No checking is done for access to pages of the buffer beyond the last page of the actual response data.

This function code provides information for the resource pool specified in the 8-byte buffer located at the guest logical address in R_{1+1}. If the buffer is not aligned on a doubleword boundary a specification exception is recognized.

When the instruction completes with condition code 0, a list of members of the specified resource pool will be returned in the buffer at the guest logical address specified by register R_2.

The possible nonzero return codes for this function code are:

CC=3 RC=4
Unsupported function code. Buffer is unchanged.

CC=3 RC=8
Not authorized for the function code. Buffer is unchanged.

CC=3 RC=12
Missing or invalid Resource Pool Name. Buffer is unchanged.

CC=3 RC=16
Named Resource Pool does not exist. Buffer is unchanged.

CC=3 RC=20
Response buffer is too small. INFCRQSZ specifies required buffer size. Values are also provided in INFCVRSN, INFCHDLN and INFCTOTL. Other response buffer contents are unpredictable.

Function Code X'0006' Response Buffer Format (INF6BK)

INF6BK DSECT in HCPINFBK COPY of HCPGPI MACLIB describes the response buffer format and might include information regarding usage of the fields in the DSECT.

The common header section (INFCHDR DSECT) is placed at the beginning of the response buffer and identifies the length of the response.

The common header includes list designation fields for the following:

- List of members of the defined Resource Pool mapped by INF6MEM DSECT

Valid lists have nonzero count, offset and length values in the header.

The version number (INF6VRSN) in the function code X'0006' response header can be used to clearly identify whether the function code is at the required service level and can be used by applications to show actual and required version in messages when required support is missing.

Function Code X'0006' Response Header (INF6BK DSECT)

```
Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF6BK         Mappings for STHYI
        Function code x'0006': Resource Pool Member List
        This function code returns the list of members of the
        specified Resource Pool.
        This function code requires the length of the buffer as
        a number of 4K pages to be specified. See the STHYI
        documentation for details of how this is specified.
        The response buffer for function code x'0006' consists
        of 2 sections of data:
        - Header Section (INF2BK DSECT, imbeds INFCHDR DSECT)
          - use offset to Resource Pool member list, length of
            entry, and count of entries to process the list.
          - The offset in INFCLSOF is from the beginning
            of the INF6BK.
          - if the count is zero there is no list so the offset
            and length will be zero.
        - Resource Pool Member list is returned as an array
          with the count of entries, offset to the first entry
          and the entry length defined by the common header.
        - The array entry structure is defined by INF6MEM DSECT.
        Use of this function code is authorized in the User
        Directory by OPTION STHYI-RESPOOL.
        Error responses:
         - CC=3 RC=4  - Unsupported function code
                      - The response buffer is not modified.
         - CC=3 RC=8  - Not authorized for function code
                      - The response buffer is not modified.
         - CC=3 RC=12 - Missing or invalid Resource Pool Name.
                      - The response buffer is not modified.
         - CC=3 RC=16 - Named Resource Pool does not exist.
                      - The response buffer is not modified.
         - CC=3 RC=20 - Response buffer is too small.
                      - INFCRQSZ specifies required buffer size.
                      - Values are also provided in INFCVRSN,
                        INFCHDLN and INFCTOTL. Other response
                        buffer contents are unpredictable.
        Update Log - see equates under INF6VRSN.
        Function code X'0006' Response Buffer
        Header Section
0000    0 Bitstring   64 INF6CHDR       Common section of header mapped
                                        by INFCHDR DSECT.
0000    0 Unsigned     2 INF6VRSN       Response buffer version.
          00000001       INF6V00001     X'0001' INF6V00001 Initial
                                        version. APAR VM66105.
0040   64 EBCDIC       8 INF6PLNA       Resource pool name.
          00000048       INF6BSB1       *-INF6BK Version 1 length in
                                        bytes of the Resource Pool Member
                                        List header.
          00000009       INF6BSD1       (*-INF6BK+7)/8 Version 1 length
                                        in doublewords of the Resource
                                        Pool Member List header.
          00000048       INF6BSZB       *-INF6BK Length in bytes for
                                        newest version of the Resource
                                        Pool Member List header.
          00000009       INF6BSZD       (*-INF6BK+7)/8 Length in
                                        doublewords for newest version of
                                        the Resource Pool Member List
                                        header.
```

Function Code X'0006' Response List Entry (INF6MEM DSECT)

```
     +-------------------------------------------------------+
   0 |                       INF6MNAM                        |
     +-------------------------------------------------------+
   8


Hex   Dec Type/Val   Lng Label (dup)    Comments
---- ---- --------- ---- -------------- --------
0000    0 Structure      INF6MEM        Mappings for STHYI
        Function code X'0006' Response Buffer
        Resource Pool Member List Entry Description
0000    0 EBCDIC       8 INF6MNAM       Resource Pool member name.
          00000008       INF6MSB1       *-INF6MEM Version 1 length in
                                        bytes of the Resource Pool Member
                                        List entry.
```

```
00000001       INF6MSD1       (*-INF6MEM+7)/8 Version 1 length
                              in doublewords of the Resource
                              Pool Member List entry.
00000008       INF6MSZB       *-INF6MEM Length in bytes for
                              newest version of the Resource
                              Pool Member List entry.
00000001       INF6MSZD       (*-INF6MEM+7)/8 Length in
                              doublewords for newest version of
                              the Resource Pool Member List
                              entry.
```

Special Conditions, Exceptions, and Usage Notes

Special Conditions

A specification exception is recognized and no other action is taken if any of the following occurs:

- The R_1 or R_2 field designates an odd-numbered register.
- The R_1 and R_2 fields designate the same register.
- The response buffer is not on a 4 KB boundary.
- For function codes 2, 4 and 6, the value for the number of pages is zero.
- For function codes 3, 5 and 6, if the search key buffer address specified in R_{1+1} is not aligned on a doubleword boundary.

Program Exceptions

- Access (store, response buffer). When an access exception occurs while storing into any part of the response buffer, the contents of any accessible portion of the response buffer are unpredictable. This is true even if the exception is defined to suppress or nullify execution of the instruction.
- Operation (if the store-hypervisor-information facility is not installed)
- Specification

Usage Notes

1. Support for function codes 1-6 is available on z/VM 6.4 with APAR VM66105, or on later releases of z/VM. Each Hypervisor section in the function code 0 response contains a mask of the supported function codes and a mask of function codes that the guest is authorized to use. When the response to function code 0 is too short to contain the masks, then only function code 0 is supported and authorized.
2. DirMaint support for the user directory options for STHYI authorization is provided with APAR VM66109 for function level 640, or with later releases of z/VM.
3. Running z/VM second level or higher, a complete response for function code 1 requires that all hypervisor levels support function code 1, and the guest at each level is authorized for function code 1 by that hypervisor level.
4. Use of the STHYI instruction within a transaction will cause the transaction to abort with either a restricted-instruction transaction-abort code or a transaction-constraint exception.
5. If the partition where z/VM is running has Global Performance Data off, function codes 0 and 1 will not be able to return certain information because it is not available to z/VM. Validity bits will be off for sets of information that are not available. The state of the validity bits should be checked before the contents of any associated fields are used to ensure the contents are valid.
6. All pages of the response buffer must be resident in guest memory for the STHYI instruction to complete. A guest access exception is presented each time a page of the buffer is found to be non-resident, and the instruction must be re-executed completely. Therefore performance might be improved by ensuring each 4 KB block of the response buffer is initialized before issuing STHYI.

7. Use of STHYI in an SSI cluster may encounter problems if members do not have the same level of support. For example:
   - If a member that is authorized for STHYI function codes 1-6 is relocated to a member that does not support those function codes and then is relocated back to a member that does have the support, they will no longer be authorized.
   - Guest initial share settings reported by STHYI function code 1 and function code 3 are not accurate if the guest had at any time relocated to a member without support for those function codes.
8. Counts of dedicated cores in the machine and partition layers refer to cores in use in dedicated partitions. The counts of dedicated cores in the hypervisor and guest layers refer to processors dedicated to guest virtual CPUs. z/VM hypervisors no longer support dedication of processors to guest virtual CPUs as of 7.1.0.
9. zIIP support necessary for z/OS guests running zCX is added by APAR VM66329. When zCX is running in a z/OS guest of z/VM, there are two layers of hypervisor. For correct reporting of zIIP configuration information, the zIIP validity bits must be on in each reported layer. Fields were not added for counts of dedicated zIIP cores in the hypervisor and guest layers because guest dedication is no longer supported.
10. In most cases, virtual CPUs are dispatched on processors of a matching CPU type. z/VM CPUAFFINITY settings could change the dispatch type of a specialty CPU type so that they are dispatched on CPs. In either case, for each CPU type the dispatch type field indicates on which processor type the virtual CPUs are dispatched. However, z/OS supports a spillover function that allows virtual zIIPs to be dispatched on CP processors (general CPUs) if zIIP processors are unavailable. In this case, the dispatch type will be INFGPUCZCP indicating that the virtual CPUs may be dispatched on either zIIP or CP processors.
11. Resource pool related fields are used to report z/OS tenant resource group settings when the hypervisor is zCX.

Part 7. Symptom Record Reporting

This part contains the following chapter:

- Chapter 30, “Symptom Record Reporting,” on page 961

© Copyright IBM Corp. 1991, 2020

Chapter 30. Symptom Record Reporting

This document, with its emphasis on application programming, describes symptom record reporting within this chapter as well as in the material covering DIAGNOSE code X'94' on "DIAGNOSE Code X'94' – VMDUMP and Symptom Record Service" on page 113. For more information on the VMDUMP command, refer specifically to *z/VM: CP Commands and Utilities Reference*, and for more information about symptom records, see *z/VM: System Operation* and *z/VM: Dump Viewing Facility*.

Reporting Software Error Symptoms (Symptom Records)

DIAGNOSE code X'94' provides a mechanism by which a program running in a virtual machine may provide a symptom record to be recorded by CP, either with or independent of, a virtual machine dump. Refer to Chapter 22, "Symptom System Service (*SYMPTOM)," on page 773 for details on how symptom records are recorded by CP.

The program-created symptom record contains a description of some programming failure, along with a description of the environment in which the failure occurred. The format of this information is indicated in the ADSR macro, which is in HCPGPI MACLIB. CP uses the same format to document symptom record information.

The VM Dump Tool's VMDUMPTL command and SYMPTOM subcommand can be used to view symptom record information. For more information on this command and subcommand, see *z/VM: VM Dump Tool*.

The Format of the Symptom Record

The symptom record consists of six sections that are structured according to the format shown in the ADSR macro. These sections are numbered 1 through 5, including an additional section that is numbered 2.1. Because sections 2.1, 3, 4, and 5 of the symptom record are variable in length, they do not need to be sequentially ordered within the record. In section 2, the application (the program running in the virtual machine) supplies the offset and the length of the nonfixed sections. The purpose of each section, including material showing its format, is described in the following sections.

Section 1 (Environmental Data)

Section 1 contains the record header with basic environmental data. The application initializes this area to binary zeros and stores the characters *SR* in the record header. The environmental data of section 1 is filled in automatically when DIAGNOSE code X'94' SR processing is invoked. The environmental data that is stored in this section provides a system context within which the software errors can be viewed. Section 1 includes items such as the:

- CPU model and serial number
- Date and time, with a time zone conversion factor
- Customer-assigned system name
- Product ID of the control program.

Section 2 (Control Data)

Section 2 contains control information with the lengths and offsets of the sections that follow. The application must initialize the control information before invoking DIAGNOSE code X'94'. Section 2 immediately follows section 1 in the symptom record structure.

© Copyright IBM Corp. 1991, 2020

Section 2.1 (Component Data)

Section 2.1 contains the name of the component in which the error occurred, as well as other specific component-related data. The application must also initialize section 2.1 before invoking the DIAGNOSE code.

Section 3 (Primary SDB—Structured Data Base—Symptoms)

Section 3 contains the **primary symptom string** of problem symptoms, which may be used to perform tasks such as duplicate problem recognition. When an application detects an error, it must store a string of symptoms in section 3, and this string becomes the primary symptom for the error. This string should be a unique and complete description of the error. All incidences of that error should produce the same string in section 3. When problems are analyzed, problems that have identical strings in section 3 represent the same error. Note that an application does not store any primary symptom string or invoke DIAGNOSE code X'94' unless it detects an error in its own processing.

Section 4 (Secondary SDB Symptoms)

Section 4 contains an optional **secondary symptom string**. The purpose of the secondary string is to provide additional symptoms that may supplement the symptoms in section 3.

Section 5 (Free-Format Data)

Section 5 contains logical segments of optional problem-related data to aid in problem diagnosis. However, the data in section 5 is not in the SDB format, which is found only in sections 3 and 4. Each logical segment in section 5 is structured in a **key-length-data** format.

Symptom Strings — SDB Format

The symptom strings placed in sections 3 and 4 of the symptom record must be in the SDB (structured database) format. In this format, the individual symptoms in sections 3 and 4 of the symptom record consist of a prefix and its associated data. Examples of typical prefixes are:

Prefix
Data

PIDS/
A component name

RIDS/
A routine name

AB/
An abend code

PRCS/
A return code

Notes for Applications Using DIAGNOSE Code X'94' SR Option

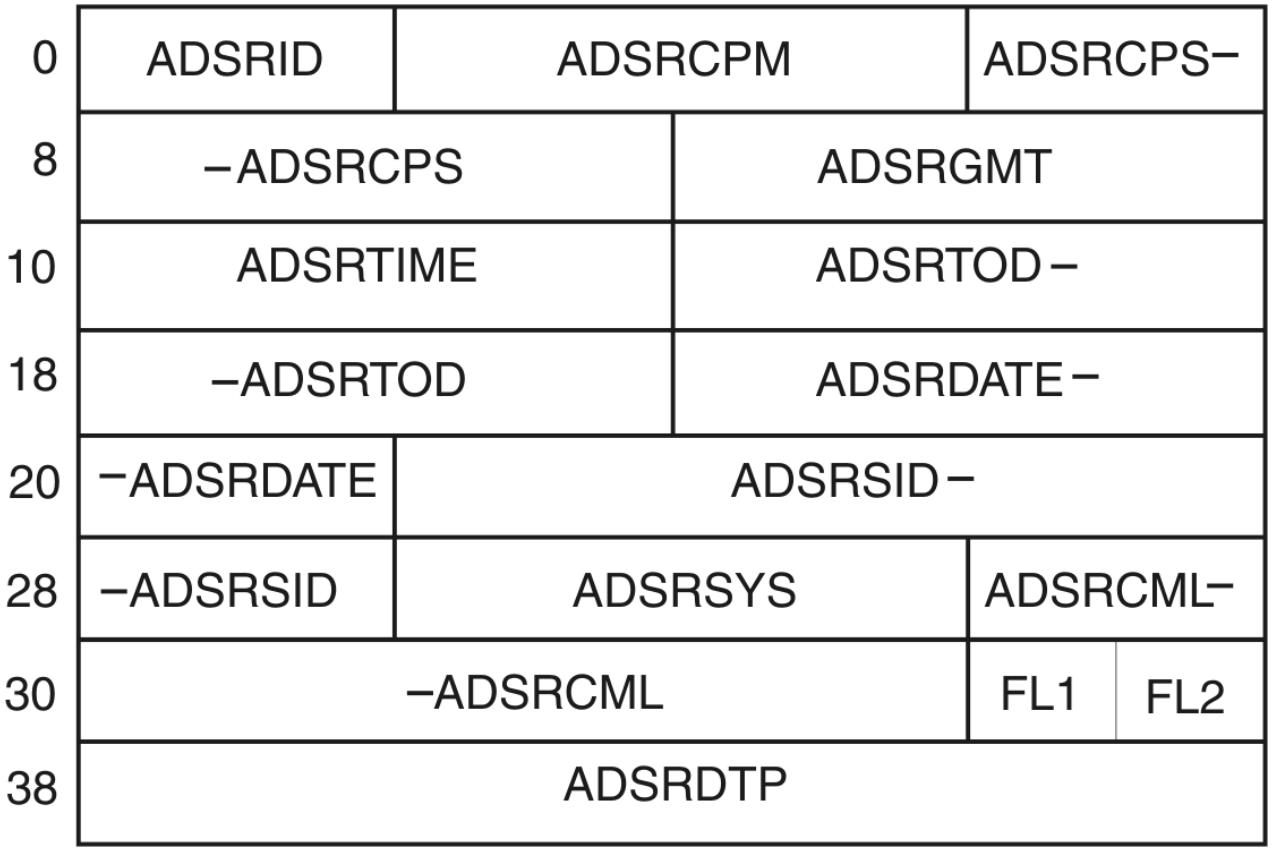
This section contains programming notes on how the various fields of the ADSR data area (symptom record) are set. Some fields of this area must be set by the caller of the DIAGNOSE code, and other fields are set by the system when the application invokes the DIAGNOSE code. The fields that the caller must always set are indicated by an RC code in the following sections. The fields that are set by CP are indicated by an RS code.

The RA code designates certain flag fields that need to be set only when certain kinds of alterations and substitutions are made in the symptom record after the incident occurs. These alterations and substitutions must be obvious to the user who interprets the data. Setting these flag fields is the responsibility of the program that alters or substitutes the data. If a program changes a symptom record that has already been recorded, it should set the appropriate RA-designated flag-bit fields as an indication of how the record has been altered.

The remaining fields, those not marked by RC, RS, or RA, are optionally set by the caller of DIAGNOSE code X'94'. When DIAGNOSE code X'94' is invoked, it checks that all the required input fields in the symptom record are set by the caller. If the required input fields are not set, DIAGNOSE code X'94' issues appropriate return and reason codes.

Programming Notes for Section 1

Notes in this section pertain to the following fields, which are in section 1 of the ADSR data area.



ADSRID	Record header (*SR*)	(RC)
ADSRCPM	CPU model number	
ADSRCPS	CPU serial number	
ADSRGMT	Local time zone conversion factor	
ADSRTIME	Time stamp	(RS)
ADSRTOD	Time stamp (HHMMSSTH)	
ADSRDATE	Date (YYMMDD)	
ADSRSID	Customer-assigned system/node name	(RS)
ADSRSYS	Product ID of the base system (BCP)	(RS)
ADSRCML	Feature and level of Symrec Service	(RS)
ADSRTRNC	Truncated flag	(RS)
ADSRPMOD	Section 3 modified flag	(RA)
ADSRSGEN	Surrogate record flag	(RA)
ADSRSMOD	Section 4 modified flag	
ADSRNOTD	ADSRTOD and ADSRDATE not-computed flag	(RS)

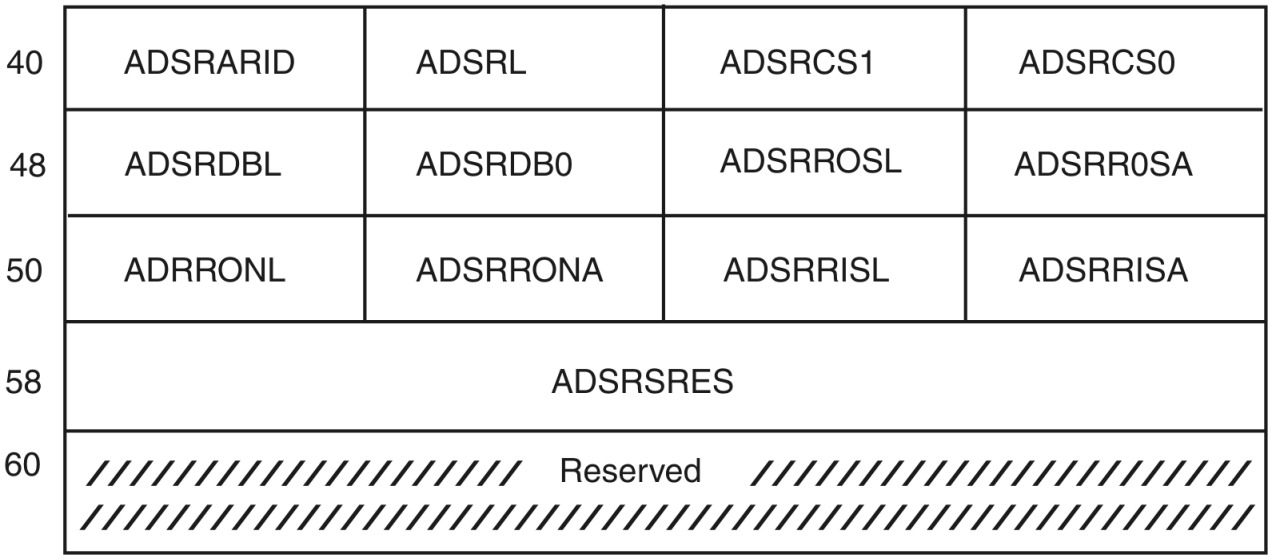
ADSRASYN	Asynchronous event flag	(RA)
ADSRDTP	Name of dump	

Notes:

1. The application invoking DIAGNOSE code X'94' must provide the space for the entire symptom record, and must initialize that space to binary zeros. The application must also store the value *SR* into ADSRID.
2. DIAGNOSE code X'94' stores the TOD clock value into ADSRTIME when the incident occurs. However, it does not compute ADSRTOD and ADSRDATE when the incident occurs, but afterward, when it formats the output. When the incident occurs, DIAGNOSE code X'94' also sets ADSRNOTD to 1 as an indication that ADSRTOD and ADSRDATE have not been computed.
3. DIAGNOSE code X'94' stores the customer-assigned system node name into ADSRSID.
4. DIAGNOSE code X'94' stores the first four digits of the base control program component ID into ADSRSYS.
5. If DIAGNOSE code X'94' truncates the symptom record, it sets ADSRTRNC to 1. This can happen when the size of the symptom record provided by the invoking application exceeds 3500 decimal bytes.
6. ADSRSGEN, if 1, indicates that the symptom record was not provided as *first time data capture* by the invoking application. Another program created the symptom record. The identification of the other program might be included with other optional information, for example, in section 5.
7. If some application creates the record asynchronously, that application should set ADSRASYN to 1. This means that the data is derived from sources outside the normal execution environment, such as human analysis or some type of machine post-processing.

Programming Notes for Section 2

Notes in this section pertain to the following fields, which are in section 2 of the ADSR data area.



ADSRARID	Architectural level designation	(RS)
ADSRL	Length of section 2	(RC)
ADSRCSL	Length of section 2.1	(RC)
ADSRCSO	Offset of section 2.1	(RC)
ADSRDBL	Length of section 3	(RC)
ADSRDBO	Offset of section 3	(RC)
ADSRROSL	Length of section 4	

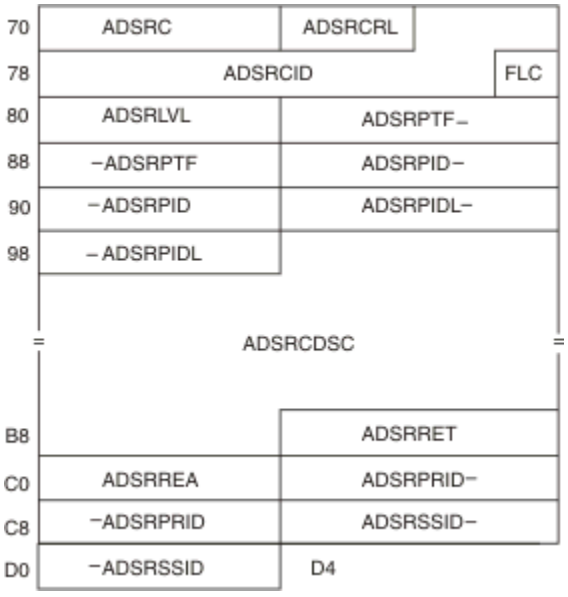
ADSRROSA	Offset of section 4
ADSRRONL	Length of section 5
ADSRRONA	Offset of section 5
ADSRRISL	Length of section 6
ADSRRISA	Offset of section 6
ADSRSRES	Reserved for system use

Notes:

1. The invoking application must ensure that the actual lengths of supplied data agree with the lengths indicated in the ADSR data area. The application should not assume that DIAGNOSE code X'94' validates these lengths and offsets.
2. The lengths and offsets in section 2 are intended to make the indicated portions of the record indirectly addressable. Invoking applications should not depend on the sections following one another, and thus should not use computed absolute offsets into the data area.
3. The value of the ADSRARID field is the architectural level at which the DIAGNOSE code X'94' SR support is operating. This field is supplied by DIAGNOSE code X'94'.
4. Section 2 has a fixed length of 48 bytes. Optional fields (not marked with RC, RS, or RA) contain binary zeros when the invoking application provides no values for them.

Programming Notes for Section 2.1

Notes in this section pertain to the following fields, which are in section 2.1 of the ADSR data area.



ADSRC	C'SR21' Section 2.1 Identifier	(RC)
ADSRCRL	Architectural level of record	(RC)
ADSRCID	Component identifier	
ADSRFLC	Component status flags	
ADSRNIBM	Non-IBM program flag	(RC)
ADSRLVL	Component release level	(RC)
ADSRPTF	Service level	
ADSRPID	PID number	(RC)

ADSRPIDL	PID release level	(RC)
ADSRCDSC	Text description	
ADSRRET	Return code	(RS)
ADSRREA	Reason code	(RS)
ADSRPRID	Problem identifier	
ADSRSSID	Subsystem identifier	

Notes:

1. This section has a fixed length of 100 bytes, and cannot be truncated. Optional fields (not marked with RC, RS, or RA) appear as binary zeros, if no values are provided.
2. ADSRCRL is the architectural level of the record. Note that ADSRARID (section 2) is the architectural level of DIAGNOSE code X'94'.
3. ADSRCID is the component ID of the application that incurred the error.

   Under some circumstances, there can be more than one component ID involved. For ADSRCID, select the component ID that is most indicative of the source of the error. Additional and clarifying data (such as, another component ID involved) is optional, and may be placed in optional entries, such as ADSRCDSC of section 2.1, section 4, or section 5.

   ADSRCID is not a required field in this section, although it is required in section 3 after the PIDS/ prefix of the symptom string. Repeating this value in section 2.1 is desirable but not required. Where the component ID is not given in section 2.1, this field must contain binary zeros.
4. ADSRNIBM is a flag indicating that a non-IBM program originated the symptom record.
5. ADSRLVL is the release level of the component indicated in ADSRCID, or zero if ADSRCID is zero.
6. ADSRPTF is the service level. It may differ from ADSRLVL because the program may be at a higher level than the release. ADSRPTF can contain any number indicative of the service level. For example, a PTF, FMID, APAR number, or user modification number. This field is not required, but it should be provided if possible.

   ADSRPID is the program identification number assigned to the program that incurred the error. When the symptom record is being created by an IBM program, ADSRPID must be provided only if it does not have an assigned component ID. Therefore, ADSRCID contains binary zeros if ADSRPID is provided.
7. ADSRPIDL is the release level of the program designated by ADSRPID.
8. ADSRCDSC is a 32-byte area that contains text, and it is only provided at the discretion of the reporting component. It provides clarifying information.
9. ADSRRET is the return code, and ADSRREA is the reason code from the execution of DIAGNOSE code X'94'. DIAGNOSE code X'94' places these values in registers 0 and 15 and in these two fields as part of its execution. The fields are right-justified, and identical to the contents of registers 0 and 15.
10. ADSRPRID is a value that can be used to associate the symptom record with other symptom records. This value must be in EBCDIC, but it is not otherwise restricted.
11. ADSRSSID is the name of a subsystem. A zero value is interpreted as no name.

Programming Notes for Section 3

Section 3 of the symptom record contains the primary symptoms associated with the error, and is provided by the application that incurred the error, or by some program that acts on its behalf. The internal format of the data in section 3 is the SDB format, with a blank separating each entry. Once this data has been passed to DIAGNOSE code X'94' by the invoker, it may not be added to or modified without setting ADSRPMOD (in Section 1) to 1. The data in this section is EBCDIC, and no binary zeros may appear. The symptoms are in the form K/D where K is a keyword of one to eight characters and D is at least one character. D can only be an alphanumeric or @, $, and #.

Notes:

1. The symptom K/D can have no imbedded blanks, but the pound sign, #, can be used to substitute for desired blanks. Each symptom (K/D combination) must be separated by at least one blank. The first symptom may start at ADSRRSCS with no starting blank, but the final symptom must have at least one trailing blank. The total length of each symptom (K/D combination) cannot exceed 15 characters.
2. This section is provided by the component that reports the failure to the system.
3. The PIDS/ entry is required, with the component ID following the slash, from all programs that originate a symptom record and have a component ID assigned. Further, it must be identical to the value in ADSRCID (section 2.1) if that is provided. (ADSRCID is not a required field).

Programming Notes for Section 4

Section 4 of the symptom record contains the secondary symptoms associated with the error incident, and it is provided by the application that incurred the error, or by some program that acts in its behalf. The internal format of the data in section 4 is the SDB format, with a single blank separating each entry. Once this data has been passed to DIAGNOSE code X'94' by the invoker, it may not be added to or modified without setting ADSRSMOD to 1.

Programming Notes for Section 5

Section 5 of the symptom record contains logical segments of data that are provided by the application or by some program that acts in its behalf. The application stores data in section 5 before DIAGNOSE code X'94' is invoked.

Notes:

1. The first segment must be stored at symbolic location ADSR5ST. In each segment, the first two characters are a hex key value, and the second two characters are the hexadecimal length of the data string, which must immediately follow the 2-byte length field. Adjacent segments must be packed together. The length of section 5 is in the ADSRRONL field in section 2, and this field should be correctly updated as a result of all additions or deletions to section 5.
2. There are 64K key values grouped in 13 ranges representing 13 potential categories. The data type (that is, hexadecimal, EBCDIC, and so forth) of section 5 is indicated by the category of the key value. Thus, the key value indicates both the user category and the data type that are associated with the information in section 5. Because the component ID is a higher order qualifier of the key, it is only necessary to control the assignment of keys within each component ID or, if a component ID is not assigned, within each PID number.

Key Value
Category and Data Type

0001-00FF
Reserved

0100-0FFF
MVS system programs

1000-18FF
VM system programs

1900-1FFF
DOS/VSE system programs

2000-BFFF
Reserved

C000-CFFF
Product programs and nonprintable hex data

D000-DFFF
Product programs and printable EBCDIC data

E000-EFFF
Reserved

F000
Any program and printable EBCDIC data

F001-F0FF
Not assignable

F100-FEFF
Reserved

FF00
Any program and nonprintable hex data

FF01-FFFF
Not assignable

Appendix A. Data Areas Used by DIAGNOSE Codes

This appendix contains descriptions of data areas and control blocks referred to by DIAGNOSE codes X'14', X'24', X'D8', X'B4' and X'B8', and X'210'.

Data Areas Used by DIAGNOSE Codes X'24' and X'210'

The following information is returned as output from DIAGNOSE codes X'24' and X'210'. Information returned from DIAGNOSE code X'210' only is indicated by an asterisk (*). Symbolic names can be used by including DEVTYPES COPY from the HCPGPI library in your program.

Note: DIAGNOSE code X'24' will no longer be upgraded for new device support. Applications using DIAGNOSE code X'24' should use DIAGNOSE code X'210' to take advantage of new device support. z/VM includes DIAGNOSE code X'24' primarily for VM/SP, VM/SP HPO, and VM/ESA (370 Feature) compatibility.

CP370 Device Classes

These are the virtual and real device classes returned in byte 0 of Ry and Ry+1, respectively, for DIAGNOSE code X'24' and bytes 4 and 8, respectively, of the VRDCBLOK for DIAGNOSE code X'210'.

```
CLASTERM EQU   X'80'          TERMINAL DEVICE CLASS
CLASGRAF EQU   X'40'          GRAPHICS DEVICE CLASS
CLASURI  EQU   X'20'          UNIT RECORD INPUT DEVICE CLASS
CLASURO  EQU   X'10'          UNIT RECORD OUTPUT DEVICE CLASS
CLASTAPE EQU   X'08'          MAGNETIC TAPE DEVICE CLASS
CLASDASD EQU   X'04'          CKD DIRECT ACCESS STORAGE DEVICE
CLASSPEC EQU   X'02'          SPECIAL DEVICE CLASS
CLASFBA  EQU   X'01'          FIXED BLOCK STORAGE DEVICE CLASS
```

CP370 Device Types

These are the virtual and real device types within each class returned in byte 1 of Ry and Ry+1, respectively, for DIAGNOSE code X'24' and bytes 5 and 9, respectively, of the VRDCBLOK for DIAGNOSE code X'210'.

Codes for devices that are marked with an asterisk (*) are returned only by using DIAGNOSE code X'210'.

CLASTERM (Terminals):

```
TYPBSC         X'80'         BISYNC LINE FOR 3270 REMOTE
TYP2700        X'40'         2700 BISYNC LINE
TYP2955        X'40'         2955 COMMUNICATIONS LINE
TYPTELE2       X'20'         TELEGRAPH TERM CONTROL TYPE II
TYPTTY         X'20'         TELETYPE TERMINAL
TYP2741        X'18'         2741 COMMUNICATIONS TERMINAL
TYPIBM1        X'10'         IBM TERMINAL CONTROL TYPE I
TYPSDLC        X'08'         synchronous data link control(ICA)
TYP3210        X'00'         3210 CONSOLE
TYP3215        X'00'         3215 CONSOLE
TYP2150        X'00'         2150 CONSOLE
TYP1052        X'00'         1052 CONSOLE
```

CLASGRAF (Graphics Devices):

```
TYPHFGD        X'.C0'         5080 HIGH FUNCTION GRAPHICS
TYP2250        X'.80'         2250 DISPLAY UNIT
TYP3277        X'.04'         3277 DISPLAY STATION
TYP3138        X'.04'         3138 DISPLAY CONSOLE
TYP3148        X'.04'         3148 DISPLAY CONSOLE
TYP3158        X'.04'         3158 CONSOLE
TYP3284        X'.02'         3284 PRINTER
TYP3286        X'.02'         3286 PRINTER
TYP3287        X'.02'         3287 PRINTER
```

© Copyright IBM Corp. 1991, 2020

```
  TYP3288       X'.02'        3288 PRINTER
  TYP3278       X'.01'        3278 DISPLAY STATION
```

CLASURI (Unit Record Input Devices):

```
  TYP3505       X'84'        3505 CARD READER
  TYP2540R      X'82'        2540 CARD READER
  TYP2501       X'81'        2501 CARD READER
```

CLASURO (Unit Record Output Devices):

```
  TYP3525       X'84'        3525 CARD PUNCH
  TYP2540P      X'82'        2540 CARD PUNCH
  TYP3820       X'4F'        3820 PRINTER
  TYPAFP1       X'4E'        ADVANCED FUNCTION PRINTER
  TYP38008      X'4D'        3800 MODEL 8 PRINTER
  TYP4248       X'4B'        4248 PRINTER
  TYP4245       X'4A'        4245 PRINTER
  TYP38003      X'49'        3800 MODEL 3 PRINTER
  TYPVAFP       X'48'        VAFP PRINTER
  TYP3262       X'47'        3262 PRINTER MODEL 5
  TYP3800       X'45'        3800 MODEL 1 PRINTER
  TYP3203       X'43'        3203 PRINTER
  TYP3211       X'42'        3211 PRINTER
  TYP1403       X'41'        1403 PRINTER
```

CLASTAPE (Tape Devices):

```
  TYP3422       X'82'        3422 TAPE DRIVE
* TYP3490       X'81'        3490 TAPE DRIVE
  TYP3420       X'10'        3420 TAPE DRIVE
  TYP3430       X'02'        3430 TAPE DRIVE
  TYP3480       X'01'        3480 TAPE DRIVE
  TYP3424       X'42'        3424 TAPE DRIVE
  TYP3590       X'83'        3590 TAPE DRIVE
  TYP9348       X'44'        9348 TAPE DRIVE
```

CLASDASD (DASD Devices):

```
* TYP3390       X'82'        3390 DISK STORAGE FACILITY
* TYP9345       X'81'        9345 DISK STORAGE FACILITY
  TYP3380       X'20'        3380 DISK STORAGE FACILITY
  TYP3330       X'10'        3330 DISK STORAGE FACILITY
  TYP3333       X'10'        3333 DISK STORAGE FACILITY
  TYP3350       X'08'        3350 DISK STORAGE FACILITY
  TYP3375       X'04'        3375 DISK STORAGE FACILITY
  TYP2305       X'02'        2305 FIXED HEAD STORAGE DEVICE
  TYP3340       X'01'        3340 DISK STORAGE FACILITY
```

CLASFBA (FBA Devices):

```
  TYP3370       X'02'        FBA Disk Storage Device
  TYP9336       X'40'        FBA Disk Storage Device
  TYP9332       X'08'        FBA Disk Storage Device
  TYP9335       X'04'        FBA Disk Storage Device
  TYPFBA        X'00'        FBA Disk Storage Device
```

CLASSPEC (Special Devices):

```
  TYPCTCA       X'80'        CHANNEL TO CHANNEL ADAPTER
  TYP3704       X'40'        3704 PROGRAMMABLE COMM. CTL UNIT
  TYP3705       X'40'        3705 PROGRAMMABLE COMM. CTL UNIT
* TYPOSA        X'20'        OPEN SYSTEMS ADAPTER DEVICE
* TYP9033       X'08'        9033 DYNAMIC SWITCH
* TYP9032       X'02'        9032 DYNAMIC SWITCH
  TYPUNSUP      X'01'        DEVICE UNSUPPORTED BY VM/370
```

CP370 Device Features

These are the real device feature codes returned in byte 3 of Ry+1 for DIAGNOSE code X'24' and byte 11 of the VRDCBLOK for DIAGNOSE code X'210'.

Codes for features that are marked with an asterisk (*) are returned only by using DIAGNOSE code X'210'.

Terminals:

```
  FTRDIAL       X'01'          3275 WITH SWITCHED LINE SUPPORT
  FTR3270       X'02'          3270 MODE, VIRTUAL 3215 DEVICE
```

Graphics Devices:

```
  FTROPRDR      X'80'          OPERATOR ID CARD READER
```

Unit Record Output Devices:

```
  FTRUCS        X'01'          UCS FEATURE
  FTR4WCGM      X'80'          3800, 4 Writable Generation Char Mods
  FTREXTSN      X'40'          IS ALSO USED FOR A 3800 PRINTER
```

Tape Devices:

```
  FTR7TRK       X'80'          7-TRACK FEATURE
  FTRDLDNS      X'40'          DUAL DENSITY FEATURE
  FTRTRANS      X'20'          TRANSLATE FEATURE
  FTRDCONV      X'10'          DATA CONVERSION FEATURE
* FTRCMPCT      X'08'          3480 DATA COMPACTION FEATURE
```

DASD Devices:

```
  FTRRPS        X'80'      ROTATIONAL POSITIONAL SENSING (RPS)
  FTRFH         X'80'      FIXED HEAD DEVICE
  FTREXTSN      X'40'      EXTENDED SENSE BYTES (24 BYTES)
* FTRDYNP       X'40'      DYNAMIC PATHING
  FTR2311T      X'20'      TOP HALF OF 2314 USED AS 2311
  SYSVIRT       X'20'      DEVICE IS A 3330V 'SYSVIRT'
* FTRVUA        X'20'      3330V THAT MAY BE DEDICATED TO
                           A VIRTUAL MACHINE
  FTR2311B      X'10'      BOTTOM HALF OF 2314 USED AS 2311
  FTR35MB       X'08'      35 MB DATA MODULE MOUNTED (3340)
  FTR70MB       X'04'      70 MB DATA MODULE MOUNTED (3340)
  FTRRSRL       X'02'      RESERVE/RELEASE ARE VALID CCW OP CODES
  VIRTUAL       X'01'      DEVICE IS A 3330V 'VIRTUAL'
  FTRVIRT       X'01'      3330 VIRTUAL (MSS) VOLUME
* FTRCOMP       X'01'      3350 in 3330 COMPATIBLE MODE
```

Special Devices:

```
  FTR3088       X'40'     CTCA IS TYPE 3088
  FTRTYP1       X'10'     TYPE ONE CHANNEL ADAPTER   (3704/5)
  FTRTYP4       X'10'     TREAT AS TYPE1 CHAN ADAPT  (370X)
  FTR3088       X'40'     CTCA IS TYPE 3088
* FTRTERM       X'80'     Unsupported Terminal Device
* FTRGRAF       X'40'     Unsupported Graphic Display
* FTRSPOOL      X'20'     Unsupported Unit Record Spooling Device
* FTRTAPE       X'08'     Unsupported Tape Device
* FTRDASD       X'04'     Unsupported DASD Device
* FTRSWCH       X'01'     Unsupported Dynamic Switch
```

CP370 Virtual Device Status

This is the virtual device status returned in byte 2 of Ry for DIAGNOSE code X'24' and byte 6 of the VRDCBLOK for DIAGNOSE code X'210'.

```
  X'20'         VIRTUAL DEVICE IS BUSY
  X'04'         VIRTUAL DEVICE IS NOT READY
  X'01'         DEVICE IS DEDICATED
```

CP370 Virtual Device Flags

These are the virtual device flags returned in byte 3 of Ry for DIAGNOSE code X'24' and byte 7 of the VRDCBLOK for DIAGNOSE code X'210'.

```
   X'80'         DASD - READ ONLY
   X'80'         IF VIRTUAL 270X, LINE ENABLED
   X'40'         IF DASD, TDISK OR VIRTUAL DISK IN STORAGE SPACE ALLOCATED BY CP
   X'40'         IF VIRTUAL 270X, LINE CONNECTED
   X'10'         IF CONS/SPOOLING, PROCESSING 1ST CCW
   X'08'         IF DASD, VIRTUAL DISK IN STORAGE SPACE ALLOCATED BY CP
   X'02'         IF DASD, RESERVE/RELEASE ARE VALID CCW OP CODES
 * X'01'         DEVICE SUPPORTS MIDAWs
```

Data Areas Used by DIAGNOSE Codes X'14' and X'D8'

You may need to refer to one of the following areas when using DIAGNOSE code X'14' or X'D8':

- SFBLOK — VM/SP 370 spool file control block
- SPLINK — VM/SP 370 spool file data block
- Extended spool file block for DIAGNOSE code X'D8'.

SFBLOK - VM/SP 370 Spool File Control Block

SFBLOK is in HCPGPI MACLIB as SFBLOK COPY.

```
NAME: SFBLOK
DESCRIPTIVE NAME: VM/SP 370 SPOOL FILE CONTROL BLOCK
DSECT NAME - SFBLOK
FUNCTION: THIS DSECT IS USED TO WHEN SPOOL FILES ARE TO BE
          TRANSLATED TO VM/SP FORMAT. (IT IS ANALOGOUS TO
          THE VM/XA SPFBK.)
CREATED BY:
      HCPSXSFB FOR SPFBK TO SFBLOK TRANSLATIONS. THIS IS
      DONE FOR DIAGNOSE X'14' AND *SPL OUTPUT.
DELETED BY:
      NOT APPLICABLE
```

Offsets		Type	Len	Name (Dim)	Description
Dec	Hex				
0	(0)	SIGNED	4		RESERVED FOR IBM USE
4	(4)	SIGNED	4		RESERVED FOR IBM USE
8	(8)	CHARACTER	8	SFBUSER	VMUSER IDENTIFICATION OF FILE OWNER
16	(10)	CHARACTER	8	SFBORIG	VMUSER IDENTIFICATION OF FILE ORIGIN
24	(18)	SIGNED	4	SFBRECNO	NUMBER OF DATA RECORDS IN FILE
28	(1C)	SIGNED	2	SFBRECSZ	LOGICAL RECORD SIZE - EXCLD. CCWS
30	(1E)	SIGNED	2	SFBFILID	BINARY SYSTEM FILE NUMBER
BITS DEFINED IN SFBFLAG:					
32	(20)	BITSTRING	1	SFBFLAG	S 1 SFBLOK CONTROL FLAGS
		`1...  ....`		SFBINUSE	X'80' FILE BEING PROCESSED
		`.1..  ....`		SFBRECOK	X'40' ALLOCATION RECORDS COMPLETE
		`..1.  ....`		SFBUHOLD	X'20' FILE IN USER HOLD STATUS

Offsets Dec	Hex	Type	Len	Name (Dim)	Description
		...1		SFBDUMP	X'10' FILE IS A CP SYSTEM DUMP
		.... 1...		SFBOPEN	X'08' INPUT FILE HAS BEEN OPENED
		.... .1..		SFBSHOLD	X'04' FILE IN SYSTEM HOLD STATUS
		.... ..1.		SFBEOF	X'02' INPUT FILE HAS REACHED EOF
		1		SFBRECER	X'01' SFBREC CHAIN INCOMPLETE
33	(21)	BITSTRING	1	SFBTYPE	DEVICE TYPE FOR SPOOL OUTPUT
34	(22)	SIGNED	2		RESERVED FOR IBM USE
36	(24)	SIGNED	4		RESERVED FOR IBM USE
40	(28)	CHARACTER	12	SFBFNAME	FILE NAME
52	(34)	CHARACTER	12	SFBFTYPE	FILE TYPE
64	(40)	CHARACTER	8	SFBDATE	CREATION DATE OF SPOOL FILE
72	(48)	CHARACTER	8	SFBTIME	CREATION TIME OF SPOOL FILE
80	(50)	SIGNED	4		RESERVED FOR IBM USE
84	(54)	SIGNED	2	SFBCOPY	NUMBER OF COPIES REQUESTED
86	(56)	BITSTRING	1	SFBCLAS	SPOOL FILE CLASS CHARACTER
BITS DEFINED IN SFBFLAG2:					
87	(57)	BITSTRING	1	SFBFLAG2	SFBLOK CONTROL FLAGS - BYTE 2
		1...		SFBHOLD	X'80' SAVE INPUT FILE; HOLD OUTPUT FILE
		.1..		SFBNOHLD	X'40' DELETE INPUT FILE; NOHOLD OUTPUT
88	(58)	CHARACTER	8	SFBDIST	DISTRIBUTION CODE
96	(60)	CHARACTER	4		RESERVED FOR IBM USE
100	(64)	BITSTRING	1	SFBSTCPY	CURRENT STARTING COPY NUMBER
BITS DEFINED IN SFBFLAG3:					
101	(65)	BITSTRING	1	SFBFLAG3	SFBLOK CONTROL FLAGS - BYTE 3
		1...		SFBLDBEG	X'80' 3800 LOAD CCWS AT BEGINNING
		.1..		SFBLDMID	X'40' 3800 LOAD CCWS ALL THRU FILE
		..1.		SFBFCB	X'20' INDICATE FCB CCWS NOW IN FILE
		.... .1..		SFBACNT	X'04' ACCOUNTING TYPE FILE
		.... ..1.		SFBSEEN	X'02' 'FILE PREVIOUSLY SEEN' FLAG
		1		SFBXFER	X'01' 'FILE TRANSFERRED' FLAG

Offsets Dec	Hex	Type	Len	Name (Dim)	Description
102	(66)	BITSTRING	1		RESERVED FOR IBM USE
BITS DEFINED IN SFBFLAG4:					
103	(67)	BITSTRING	1	SFBFLAG4	MORE STATUS FLAGS - BYTE 4
		1...		SFBINVS	X'80' SFBLOK IS IN SYSSPOOL's VIRTUAL STORAGE
		.1..		SFBTUSE	X'40' FILE IN TEMPORARY USE BY SYSTEM
		..1.		SFBNORET	X'20' NORETURN FLAG
		...1		SFBVLEN	X'10' Original record length available
		.... 1...		SFBPURGD	X'08' File is 'to be purged'
		.... .1..		SFBCONV	X'04' File has been converted
		.... ..1.		SFBBCONV	X'02' File being converted
		1		SFBXABER	X'01' AN XAB DISK RECORD IS INCORRECT SFBFLASH CONTAINS THE DISK ADDRESS OF THE INCORRECT RECORD.
		.... 11.1		SFBDSIZE	(*-SFBLOK)/8 DEFAULT SIZE, NON EXTENDED
104	(68)	DBL WORD	8	SFBUFORM	USER SPECIFIED FORM NUMBER
112	(70)	DBL WORD	8	SFBOFORM	OPERATOR SPECIFIED FORM NUMBER
		 1111		SFBR2SIZ	(*-SFBLOK)/8 VM/SP RELEASE 2 SIZE IN DBL WDS
120	(78)	SIGNED	2	SFBFCBNL	LONGEST IMBEDDED FCB (3211-TYPE)
122	(7A)	SIGNED	2	SFBFCBXL	LONGEST IMBEDDED FCB (EXTENDED)
					The following 4 bytes are reserved for compatibility when Diagnose X'14' and *SPL return an SFBLOK to the user. In particular, the bits that are useful to Diagnose X'14' and *SPL are the ones which indicate the KEEP/NOKEEP status (SFBXKEEP) and the MSG/NOMSG status (SFBXMSG) of the file.
124	(7C)	BITSTRING	1	SFBXSPT(4)	RESERVED FOR Diag X'14' and *SPL use
128	(80)	DBL WORD	8	SFBDEST	USER SPECIFIED DESTINATION
136	(88)	SIGNED	4		RESERVED FOR IBM USE
140	(8C)	SIGNED	2	SFBXABL	LENGTH OF XAB EXTENDED ATTRIBUTE BUFFER
BITS DEFINED IN SFBFLG4A:					
142	(8E)	BITSTRING	1	SFBFLG4A	SFBLOK FLAG
		1...		SFBREAD	X'80' INPUT SPOOL FILE HAS BEEN READ
		.1..		SFBPCHEK	X'40' ALREADY CHECKED FOR PURGE
		..1.		SFBORIGN	X'20' DIAGNOSE X'F8' ORIGINATING NODE AND USERID ARE STORED IN THE FIRST SPLINK

Offsets Dec	Hex	Type	Len	Name (Dim)	Description
		...1		SFBVAFP	FILE CREATED ON VAFP PRINTER
143	(8F)	BITSTRING	1	SFBCENT	HEXADECIMAL REPRESENTATION OF THE CENTURY PORTION OF THE YEAR IN SFBDATE
144	(90)	SIGNED	4		RESERVED FOR IBM USE
BITS DEFINED IN SFBFLAG5:					
148	(94)	BITSTRING	1	SFBFLAG5	FLAG
		1...		SFBCDMP	X'80' INDICATE CURRENT DUMP SFBLOK
		.1..		SFBCONTO	X'40' CONSOLE SPOOLED TO ANOTHER VM
		..1.			X'20' RESERVED FOR IBM USE
		...1		SFBOPFRE	X'10' COPY OF AN OPEN READER FILE
149	(95)	ADDRESS	3		RESERVED FOR IBM USE
152	(98)	CHARACTER	1		RESERVED FOR IBM USE
152	(98)	BITSTRING	1		RESERVED FOR IBM USE
153	(99)	BITSTRING	1	SFBORGIX	BITMAP INDICATING WHICH MEMBER OF AN SSI CLUSTER CREATED THIS FILE: X'00000001' - MEMBER #1, X'00000010 - MEMBER #2, ETC.
154	(9A)	SIGNED	2		RESERVED FOR IBM USE
156	(9C)	BITSTRING	1		RESERVED FOR IBM USE
157	(9D)	BITSTRING	1		RESERVED FOR IBM USE
158	(9E)	BITSTRING	1		RESERVED FOR IBM USE
159	(9F)	BITSTRING	1		RESERVED FOR IBM USE
160	(A0)	CHARACTER	1	(0)	RESERVED FOR IBM USE
160	(A0)	DBL WORD	8		RESERVED FOR FUTURE USE
184	(B8)	CHARACTER	8	SFBSCLAB	SPOOL FILE SECLABEL
		...1 1...		SFBSIZE	(*-SFBLOK)/8 SIZE IN DOUBLE WORDS
		...1 1...		SFBFNFT	L'SFBFNAME+L'SFBFTYPE
124	(7C)	BITSTRING	1	SFBXQUE	SPFQUEUE saved here for Diag X'14'
125	(7D)	BITSTRING	1	SFBXSYTY	SPFSYSTY saved here for Diag X'14'
BITS DEFINED IN SFBXSTAT:					
126	(7E)	BITSTRING	1	SFBXSTAT	SPFSTAT saved here for Diag X'14'
		1...		SFBXINUS	X'80' SPFINUSE (always turned off because SPTAPE LOAD restored the entire byte and should never load a file with the in use flag on. SPTAPE is no longer supported but this flag is saved for compatibility in Diag X'14' and *SPL output.) The real in-use status of the file is recorded in SFBFLAG.SFBINUSE flag.

Offsets		Type	Len	Name (Dim)	Description
Dec	Hex				
		.... 1...		SFBXKEEP	X'08' SPFKEEP
		.... .1..		SFBXMSG	X'04' SPFMSG
127	(7F)	BITSTRING	1	SFBXTYPE	SPFTYPE saved here for Diag X'14' and *SPL output compatibility

SPLINK - VM/SP 370 Spool File Data Block

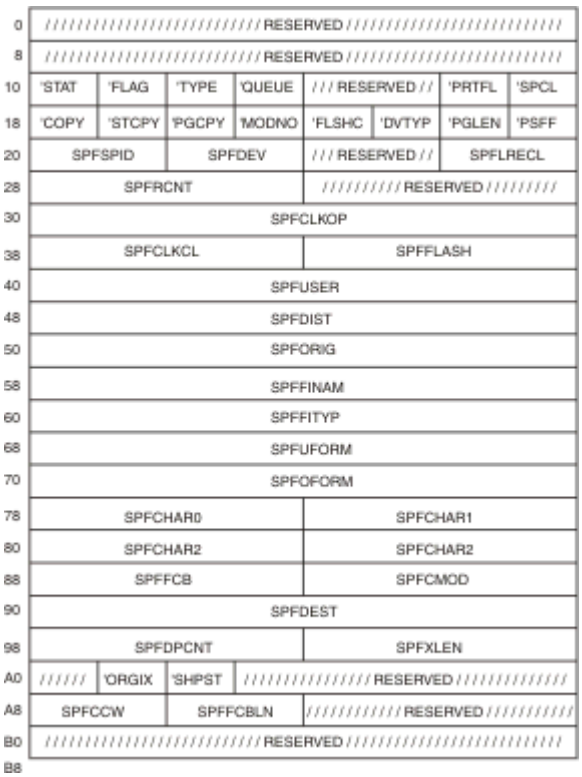
SPLINK is in HCPGPI MACLIB as SPLINK COPY.

NAME: SPLINK
DESCRIPTIVE NAME: VM/SP 370 SPOOL FILE DATA BLOCK
DSECT NAME - SPLINK
FUNCTION: THIS DSECT WHEN SPOOL FILES NEED TO BE TRANS-
LATED TO VM/SP FORMAT. (IT IS ANALOGOUS TO
THE VM/XA SPDBK.)
CREATED BY:
HCPSXSPL FOR DIAGNOSE X'14' AND *SPL OUTPUT.
DELETED BY:
NOT APPLICABLE

Offsets		Type	Len	Name (Dim)	Description
Dec	Hex				
0	(0)	SIGNED	4		RESERVED FOR IBM USE
4	(4)	SIGNED	4		RESERVED FOR IBM USE
8	(8)	SIGNED	4		RESERVED FOR IBM USE
12	(C)	SIGNED	4	SPRECNUM	NUMBER OF DATA RECORDS IN BUFFER
16	(10)	BITSTRING	1	SPLKDATA(0)	START OF SPLINK DATA AREA
		...1		SPSIZE	(*-SPLINK) SIZE IN BYTES
16	(10)	SIGNED	4016		AREA FOR CCW'S AND DATA
4032	(FC0)			SPSIZORG	(*-SPLINK) SIZE WITH ORIGINAL NODE/USERID
4032	(FC0)	CHARACTER	8	SPORIGID	DIAGNOSE X'F8' ORIGINATING USERID
4040	(FC8)	CHARACTER	8	SPNODEID	DIAGNOSE X'F8' ORIGINATING NODE
4048	(FD0)			SPDATLEN	(*-SPLINK) SIZE WITHOUT ORIGINAL NODE/ USERID
4048	(FD0)	CHARACTER	4	SPCHAR	3800 CHAR ARR TABLE 0 FOR FILE
4052	(FD4)	SIGNED	4	SPSPLNKC	COUNT OF SPLINKS FOR THIS FILE
4056	(FD8)	SIGNED	2	SPRECMAX	MAX CCW DATA LENGTH IN FILE
4058	(FDA)	CHARACTER	6		RESERVED FOR FUTURE USE
4064	(FE0)	CHARACTER	4	SPFCB	3800 FCB FOR FILE
4068	(FE4)	CHARACTER	4	SPCMOD	3800 COPY MOD FOR FILE
4072	(FE8)	CHARACTER	4	SPCHAR1	3800 ARR TABLE 1 FOR FILE
4076	(FEC)	CHARACTER	4	SPCHAR2	3800 ARR TABLE 2 FOR FILE
4080	(FF0)	CHARACTER	4	SPCHAR3	3800 ARR TABLE 3 FOR FILE

Offsets Dec	Offsets Hex	Type	Len	Name (Dim)	Description
4084	(FF4)	BITSTRING	1	SPFLSHC	S 1 3800 FLASH COUNT
BITS DEFINED IN SPFLAG1:					
4085	(FF5)	BITSTRING	1	SPFLAG1	S 2 3800 FLAG BYTE
		1...		SPCOPYFG	X'80' MULT COPIES IN ONE TRANSMISSION
		.1..		SPBTSTAC	X'40' 3800 BTS SPECIFIED
4086	(FF6)	BITSTRING	1	SPCMCHR	S 3 COPY MODIFICATION TRANSLATE NUM
4087	(FF7)	BITSTRING	1	SPPGLEN	S 4 PAPER LENGTH (1/2 - INCHES)
4088	(FF8)			SPNDATLN	(*-SPLINK) PTR END OF DATA UNLESS FIRST SPLINK
4088	(FF8)	SIGNED	2	SPFILID	FILID USED FOR VERIFICATION
4090	(FFA)	CHARACTER	6	SPTIME	SFBTIME- USED FOR VERIFICATION
		..11		SPENDSIZ	*-SPCHAR END OF BUFFER SIZE IN BYTES
4090	(FFA)			SPLKDSZ	*-SPLKDATA Length of the data area
4	(4)	SIGNED	2		RESERVED FOR IBM USE
6	(6)	SIGNED	2		RESERVED FOR IBM USE
		1... 1...		SPTAGSZ	136 VM/SP 370 TAG RECORD LENGTH
		.... 11..		SPTAG	12 OFFSET BEYOND SPDATA TO TAG DATA

Extended Spool File Block for DIAGNOSE Code X'D8'




```
10   SPFSTAT   1X   SPOOL FILE STATUS FLAGS
        X'80' = CLOSED FILE IS IN USE.
        X'40' = FILE IS OPEN (BEING CREATED)
        X'20' = FILE HAS USER HOLD
        X'10' = FILE HAS SYSTEM HOLD
        X'08' = FILE HAS 'KEEP' OPTION SET
        X'04' = FILE HAS 'MSG' OPTION SET
11   SPFFLAG   1X   SPOOL FILE ACTION FLAGS
        X'40' =  LAST RECORD PROCESSED (USUALLY RDR FILES)
12   SPFTYPE   1X   SPOOL FILE ORIGINATING DEVICE TYPE
        X'80' = CAME FROM REAL READER
        X'40' = CAME FROM VIRTUAL PUNCH
        X'22' = CAME FROM VIRTUAL PRINTER
        X'23' = CAME FROM VIRTUAL 3800 MODEL 3 PRINTER
        X'20' = CAME FROM VIRTUAL CONSOLE
        X'10' = SYSTEM CREATED SPOOL FILE
13   SPFQUEUE  1X   SPOOL FILE QUEUE LOCATION
        X'80' = FILE IS ON THE RDR QUEUE
        X'40' = FILE IS ON THE PUNCH QUEUE
        X'20' = FILE IS ON THE PRINTER QUEUE
16   SPFPRTFL  1X   FLAGS FOR ADVANCED FUNCTION PRINTERS
        X'80' =  3800 LOAD CCWS APPEAR AT BEGINNING
        X'40' =  3800 LOAD CCWS APPEAR THROUGHOUT FILE
        X'20' =  LOAD WCGM OR GRAPHMOD CCWS APPEAR
        X'10' =  FILE IS NOT EMPTY FLAG
        X'08' =  FLASH ALL COPIES OF THE FILE
        X'04' =  FILE CONTAINS X'5A' CCWS - AFP CONTROL
        X'02' =  FILE HAS AN XAB ASSOCIATED WITH IT
        X'01' =  FILE TRANSFERRED
17   SPFSPCL   1C   SPOOL FILE CLASS
18   SPFCOPY   1X   FILE COPY COUNT
19   SPFSTCPY  1X   NUMBER OF COPIES AT PRINT START
1A   SPFPGPY   1X   PAGE COPY COUNT (USED ONLY FOR 3800)
1B   SPFMODNO  1X   COPY MOD CHARACTER SET NUMBER (0-3)
1C   SPFFLSHC  1X   FLASH COUNT
1D   SPFDVTYP  1X   BITS DEFINED FOR SPFDVTYP AS FOLLOWS:
        X'80' =  TYPRDR    SPOL - CARD READER DEVICE
        X'81' =  TYP2501   SPOL - 2501 CARD READER
        X'82' =  TYP2540R  SPOL - 2540 CARD READER
        X'84' =  TYP3505   SPOL - 3505 CARD READER
        X'40' =  TYPPUN    SPOL - CARD PUNCH DEVICE
        X'42' =  TYP2540P  SPOL - 2540 CARD PUNCH
        X'44' =  TYP3525   SPOL - 3525 CARD PUNCH
        X'20' =  TYPPRT    SPOL - PRINTER TYPE DEVICE
```

```
        X'21' =  TYP1403   SPOL - 1403 PRINTER
        X'22' =  TYP32XX   SPOL - 3203 OR 3211 PRINTER
        X'26' =  TYP3203   SPOL - 3203 PRINTER
        X'22' =  TYP3211   SPOL - 3211 PRINTER
        X'28' =  TYP3800   SPOL - 3800 PRINTER
        X'23' =  TYP3262   SPOL - 3262 PRINTER
        X'24' =  TYP4245   SPOL - 4245 PRINTER
        X'29' =  TYP4248   SPOL - 4248 PRINTER
        X'25' =  TYP3820   SPOL - 3820 PRINTER - dedicated only
        X'27' =  TYPAFP1   SPOL - AFP1 PRINTER - dedicated only
        X'2A' =  TYPVAFP   SPOL - VAFP PRINTER - simulated only
```

```
1E   SPFPGLEN  1X   PAPER LENGTH
1F   SPFPSFF   1X   FLAGS FOR PRINT SERVICES FACILITY
        X'80' =  FILE IS BEING CONVERTED (CAN BE STOPPED)
        X'40' =  FILE HAS BEEN CONVERTED (CANNOT DELETE)
        X'20' =  FILE HAS BEEN CONVERTED ON ANOTHER SYSTEM IN THE CSE
                 COMPLEX.  DO NOT SELECT THIS FILE FOR CONVERSION.
        X'10' =  3800 BTS SPECIFIED
20   SPFSPID   1H   USER SPOOL FILE ID NUMBER (NOT UNIQUE)
22   SPFDEV    1H   REAL OR VIRT NUMBER OF DEVICE PROCESSING FILE
26   SPFLRECL  1H   LENGTH OF SPOOL FILE RECORDS
28   SPFRCNT   1F   TOTAL NUMBER OF LOGICAL RECORDS
30   SPFCLKOP  1D   TOD (FULL) AT 'OPEN' TIME
38   SPFCLKCL  1F   TOD HIGH ORDER WORD AT 'CLOSE' TIME
3C   SPFFLASH  CL4  FORMS OVERLAY (FLASH) NAME
40   SPFUSER   CL8  USER IDENTIFICATION OF FILE OWNER
48   SPFDIST   CL8  DISTRIBUTION CODE
50   SPFORIG   CL8  USER IDENTIFICATION OF FILE ORIGINATOR
58   SPFFINAM  CL8  FILE NAME
60   SPFFITYP  CL8  FILE TYPE
68   SPFUFORM  CL8  USER FORM NAME
70   SPFOFORM  CL8  OPERATOR FORM NUMBER
78   SPFCHAR0  CL4  CHARACTER SET NAME - FIRST
7C   SPFCHAR1  CL4  CHARACTER SET NAME - SECOND
80   SPFCHAR2  CL4  CHARACTER SET NAME - THIRD
84   SPFCHAR3  CL4  CHARACTER SET NAME - FOURTH
88   SPFFCB    CL4  FCB NAME OR LINES/INCH
8C   SPFCMOD   CL4  COPY MODIFICATION MODULE NAME
90   SPFDEST   1D   DESTINATION VALUE
98   SPFDPCNT  1F   COUNT OF DATA PAGES
9C   SPFXLEN   1F   LENGTH OF THE XAB IF ANY
A1   SPFORGIX  1X   BITMAP INDICATING WHICH MEMBER OF AN
                    SSI CLUSTER CREATED THIS FILE:
                    X'00000001' -> MEMBER #1,
                    X'00000010  -> MEMBER #2, ETC.
A2   SPFSHPST  1X   CSE STATUS
        X'01' =  EXTENDED FCB IS IMBEDDED
A8   SPFCCW    XL2  MAX CCW DATA LENGTH IN FILE
AA   SPFFCBLN  1H   MAX LOAD FCB CCW LENGTH
```

External Attribute Buffer Used by DIAGNOSE Codes X'B4', X'B8', and X'290'

The external attribute buffer (XAB) is a control block that contains data you create to specify additional information about a print file. Each print file has its own XAB, and CP has the facilities to maintain XABs.

Suggested Format for an External Attribute Buffer

Figure 106 on page 979 shows the suggested format for data contained in an external attribute buffer. The basic design of the format is to create or locate a specific block without affecting other blocks within the XAB.

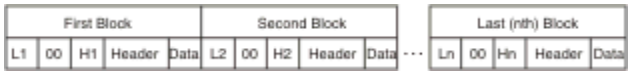


Figure 106. Suggested Format of an External Attribute Buffer

Lx
is the length of a block. This is a 2-byte field that specifies the total number of bytes for this block.

00
is reserved.

Hx
is the length of the header. This is a 2-byte field that specifies the total number of bytes used for header information. The value is the size of the header + 2 (for the size of the Hx field itself).

header
is the header data. This is a variable size field which identifies the block.

For multiple independent blocks to be contained in the XAB, it is necessary for each block to have a unique header.

- The first part of the header should be a name or character string that uniquely identifies who or what is defining the block. For example, a company name or trademark.
- The second part of the header should be the name of the product associated with the company or trademark.
- The third part of the header should be the name of the block.
- The fourth part of the header should be format level for the block. If a change is made to a defined block, it is reflected in the format level for that block.

The following are examples of headers.

Blocks defined for products from IBM might use a header like:

```
IBM - VM - BLOCK XYZ - LEVEL 0.0.0
```

And if IBM changed this block, the header might look like:

```
IBM - VM - BLOCK XYZ - LEVEL 1.0.0
```

Blocks defined for products from company JJKKLL might use a header like:

```
JJKKLL - PROD1 - BLOCK OPQ - LEVEL 0.0.0
```

data
is the actual data for the block. This is a variable size field.

Note: CP has no restrictions on the content of an XAB except that the total size of the XAB cannot exceed 32KB-1 bytes (32,767 bytes). Although CP does not check to see that the standard format has been followed, we recommend that each user of an XAB use the suggested format for multiple uses of the XAB.

Appendix B. Sample Programs Using DASD Block I/O System Service

Following are two sample programs using the DASD Block I/O system service (*BLOCKIO). The first program verifies writing to a DASD device using *BLOCKIO and the second program verifies reading a DASD device using *BLOCKIO. Both programs use CMS IUCV support. For more information on CMS IUCV support, see *z/VM: CMS Macros and Functions Reference* and *z/VM: CMS Application Development Guide for Assembler*.

Also, for more information on *BLOCKIO itself, see Chapter 14, “DASD Block I/O System Service (*BLOCKIO),” on page 721.

Note: These programs work, however, they could be easily enhanced. They are presented as samples and should be viewed as such.

Write Program

```
*************************************************************************
*                                                                       *
*    SAMPLE PROGRAM ID - BLKWRITE                                       *
*                                                                       *
*    TYPE SYSTEM - CPIUCV/BLOCKIO                                       *
*                                                                       *
*    PURPOSE OF SAMPLE PROGRAM -                                        *
*                  VERIFY WRITING TO A DASD DEVICE USING THE            *
*                  CP BLOCK I/O INTERFACE                               *
*                                                                       *
*    METHOD -                                                           *
*              CONNECTION IS MADE TO A DASD DEVICE THRU BLOCK           *
*              I/O.THE ENDING BLOCK NUMBER IS OBTAINED AND A LOOP       *
*              IS SET UP TO START WRITING AT BLOCK 1 THRU THE ENDING    *
*              BLOCK.                                                   *
*                                                                       *
*    EXPECTED RESULTS (NORMAL) -                                        *
*                  MSGS - CONNECTION COMPLETE                           *
*                       - SEND DONE                                     *
*                                                                       *
*    EXPECTED RESULTS (ABNORMAL) - ANYTHING OTHER THAN ABOVE.           *
*                                                                       *
*    EXECUTION INSTRUCTIONS:  BEFORE LOADING  AND STARTING THE          *
*             FOLLOWING STEPS MUST BE PREFORMED:                        *
*                                                                       *
*                     - FORMAT THE DASD(512,1K,2K,OR 4K)                *
*                     - ISSUE THE RESERVE COMMAND FOR THE DASD          *
*                     - ISSUE 'FILEDEF BLKPGM DISK XXX' WHERE XXX       *
*                       IS THE VIRTUAL ADDRESS OF THE DASD              *
*                                                                       *
*    NOTE: THIS PROGRAM CAN BE RUN AGAINST ALL DASD                     *
*          AND ALL BLOCKSIZES.                                          *
*                                                                       *
*************************************************************************
IUCV01 CSECT
       BALR 12,0
       USING *,12
       USING NUCON,0
       ST    14,SAVE
       LA    1,BDISK          ISSUE DISK ID
       SVC   202
       DC    AL4(1)
       LA    6,SMPPGM
       IMMCMD SET,NAME='GO',EXIT=OUT       IMMCMD FOR WAIT
*
       HNDIUCV SET,EXIT=BCON,NAME=(6)       ENABLE IUCV
       LTR    15,15                            TEST
       BNZ    ERRSET                        BRANCH ON ERROR
       LA    5,PLIST                        GET PLIST
       USING IPARML,5                       ADDRESSABILITY
       XC   0(IPSIZE*8,5),0(5)              CLEAR PLIST
```

© Copyright IBM Corp. 1991, 2020

```
       MVC   VDEVA(2),DSKAD                VIRT.DEV ADDRESS
       MVC   PACK01+2(2),DBLK              BLOCKSIZE
       MVC   OFFSET(4),DSKOFF              DISK OFFSET
*
       IUCV  CONNECT,PRMLIST=(5),USERID=ID,USERDTA=PACK01,PRMDATA=YES, *
               MSGLIM=TEN,MF=L              ISSUE CONNECT
       CMSIUCV CONNECT,PRMLIST=(5),NAME=SMPPGM,EXIT=BCON
       LTR 15,15                            CONNECT OK
       BZ     OKS                           YES BRANCH
       LR      7,5                          NO SHOW WHY
       LINEDIT  TEXT='BAD CONNECT PLIST FOLLOWS'
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = .....................................*
               ............................',SUB=(HEX4A,(7)),        *
               DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       B        EXIT
*
OKS    EQU   *
       WRTERM  'CONNECT OK ENTER GO TO CONTINUE'
*
WT     NI     ECB1,X'00'               WAIT FOR CONNECT
       WAIT   ECB=ECB1
       LA     11,1                     SET UP TO WRITE RECORDS
       ST     11,MGSS                  STORE BLK NUMBER FOR PRMMSG
       ST     11,NXB                   STORE IT FOR COMPARE
*
SND    LH      7,PATH
       XC     IPARML(40),IPARML        CLEAR PARMLIST
       IUCV SEND,PRMLIST=(5),DATA=PRMMSG,PRMMSG=MGSS,                  *
               TYPE=2WAY,PATHID=(7),TRGCLS=WRT     SEND IT
       BZ      OKS2          SEND OK BRANCH
       LR      7,5           OTHER WISE PRINT OUT THE PLIST
       DMSKEY NUCLEUS
       LINEDIT TEXT='BAD SEND PLIST     ',                             *
               DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = .....................................*
               ............................',SUB=(HEX4A,(7)),        *
               DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
*
OKS2   EQU *
       L     10,ENDBLK
       L     11,NXB                 LOAD BLOCK NUMBER JUST SENT
       LA    11,1(11)               UP BLOCK COUNT
       CR    11,10                  LAST RECORD
       BH    FIN                    YES  GO TELL USER
       ST    11,NXB                 NO SAVE THIS BLK NUMB
       ST    11,MGSS                AND STORE IT FOR PRMMSG
       B      SND
*
ERRSET LINEDIT TEXT='=BAD SET'
*
FIN    WRTERM  'SEND ALL DONE ENTER GO TO EXIT'
       NI     ECB1,X'00'           WAIT FOR SEND TO COMPLETE
       WAIT   ECB=ECB1
*
EXIT   HNDIUCV CLR,NAME=SMPPGM
       L      14,SAVE
       BR     14
       DROP   5
*
OUT         EQU    *               OUT IS IMMCMD EXIT TO TURN OFF WAIT
            DROP 12
            LR    9,15
            LR    8,14
            USING  OUT,9
            OI     ECB1,X'C0'
            BR     8
*
BCON   DS     0D
       STM  0,15,0(13)
       LR   12,15
       USING  BCON,12
```

```
       USING  IPARML,2
       CLI    IPTYPE,X'03'              CHECK FOR SEVER
       BE     SEV                       GO SHOW WHY
       CLI    IPTYPE,X'02'        CONNECTION COMPLETE
       BNE    NOTC                    BRANCH IF NOT OTHERWISE
       MVC    ENDBLK(4),IPUSER+4     STOR LAST BLOCK
       LH    7,IPPATHID              GET PATHID
       STH   7,PATH                  STOR IT
       B      OT                   GO SEND FIRST RECORD
*
NOTC   CLI    IPTYPE,X'07'         MSG COMPLETE
       BNE    OT                   IF NOT RETURN OTHERWISE
       LH    7,IPPATHID            GET PATHID
       STH   7,PATH                SAVE IT
       L     5,IPRMMSG1            GET RETURN CODE
       LTR   5,5                   CHECK IT
       BZ    OT                    IF ZERO BRANCH OTHERWISE PRINT BUF
       DMSKEY NUCLEUS
       LR   7,2
       LINEDIT TEXT='BAD SEND PLIST FOLLOWS',                          *
              DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = ......................................*
              ............................',SUB=(HEX4A,(7)),        *
              DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       B     OT                    AND RETURN
*
*     IF SEVER IPTYPE SHOW BUFFER
SEV    DMSKEY NUCLEUS
       LR   7,2
       LINEDIT TEXT='SEVER PLIST FOLLOWS',                             *
              DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = ......................................*
              ............................',SUB=(HEX4A,(7)),        *
              DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
*
OT     LM   0,15,0(13)
       BR   14
*
ALIGN  DS   0D
MSK    DC   X'00'
PLIST  DS   5D
WRT    DC   F'01'
ERRBLK DC   F'400'
ECB1   DC   F'0'
ENDBLK DC   F'0'
SAVE   DC   F'0'
SMPPGM DC  CL8'SMPPGM'
PACK01 DC  F'0'
OFFSET DC  F'0'
VDEVA  DC  H'0'
       DC  3H'0'
ID     DC  CL8'*BLOCKIO'
BDISK  DC  CL8'DISKID'
       DC  CL8'BLKPGM'
DSKAD  DS  XL2
DBLK   DS  H
DSKOFF DS  F
R15    EQU  15
MGSS   DC   F'0'
       DC   AL4(BUF)
TEN    DC   H'255'
BUF    DC   1024CL1'F'
NXB    DC    F'0'
PATH   DC    H'0'
       COPY IPARML
       NUCON
       END
```

Read Program

```
*************************************************************************
*                                                                       *
*    SAMPLE PROGRAM ID - BLKREAD                                        *
*                                                                       *
*    TYPE SYSTEM - CPIUCV/BLOCKIO                                       *
*                                                                       *
*    PURPOSE OF SAMPLE PROGRAM -                                        *
*                  VERIFY READING A DASD DEVICE USING THE               *
*                  CP BLOCK I/O INTERFACE                               *
*                                                                       *
*    METHOD -                                                           *
*              CONNECTION IS MADE TO A DASD DEVICE THRU BLOCK           *
*              I/O.THE ENDING BLOCK NUMBER IS OBTAINED AND A LOOP       *
*              IS SET UP TO START READING AT BLOCK 1 THRU THE ENDING    *
*              BLOCK.THE FINAL BLOCK IS WRITTEN OUT.                    *
*                                                                       *
*    EXPECTED RESULTS (NORMAL) -                                        *
*                  MSGS - CONNECTION COMPLETE                           *
*                       - SEND DONE                                     *
*                       - THE FINAL BLOCK OF DATA(F'S INS THIS CASE)     *
*                                                                       *
*    EXPECTED RESULTS (ABNORMAL) - ANYTHING OTHER THAN ABOVE.           *
*                                                                       *
*    EXECUTION INSTRUCTIONS:  BEFORE LOADING  AND STARTING THE          *
*             FOLLOWING STEPS MUST BE PREFORMED:                        *
*                                                                       *
*                     - FORMAT THE DASD(512,1K,2K,OR 4K)                *
*                     - ISSUE THE RESERVE COMMAND FOR THE DASD          *
*                     - ISSUE 'FILEDEF BLKPGM DISK XXX' WHERE XXX       *
*                       IS THE VIRTUAL ADDRESS OF THE DASD              *
*                                                                       *
*************************************************************************
*
IUCV01 CSECT
       BALR 12,0
       USING *,12
       USING NUCON,0
       ST    14,SAVE
       LA    1,BDISK          ISSUE DISK ID
       SVC   202
       DC    AL4(1)
       LA    6,SMPPGM
       IMMCMD SET,NAME='GO',EXIT=OUT       IMMCMD FOR WAIT
*
       HNDIUCV SET,EXIT=BCON,NAME=(6)       ENABLE IUCV
       LTR    15,15                            TEST
       BNZ    ERRSET                        BRANCH ON ERROR
       LA    5,PLIST                        GET PLIST
       USING IPARML,5                       ADDRESSABILITY
       XC   0(IPSIZE*8,5),0(5)              CLEAR PLIST
       MVC   VDEVA(2),DSKAD                 VIRT.DEV ADDRESS
       MVC   PACK01+2(2),DBLK               BLOCKSIZE
       MVC   OFFSET(4),DSKOFF               DISK OFFSET
*
       IUCV  CONNECT,PRMLIST=(5),USERID=ID,USERDTA=PACK01,PRMDATA=YES, *
               MSGLIM=TEN,MF=L               ISSUE CONNECT
       CMSIUCV CONNECT,PRMLIST=(5),NAME=SMPPGM,EXIT=BCON
       LTR 15,15                             CONNECT OK
       BZ     OKS                            YES BRANCH
       LR      7,5                           NO SHOW WHY
       LINEDIT  TEXT='BAD CONNECT PLIST FOLLOWS'
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = ........................................*
              ..............................',SUB=(HEX4A,(7)),        *
              DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       B        EXIT
*
OKS    EQU   *
       WRTERM  'CONNECT OK ENTER GO TO CONTINUE'
*
WT     NI     ECB1,X'00'               WAIT FOR CONNECT
       WAIT   ECB=ECB1
       LA     11,1                     SET UP TO WRITE RECORDS
       ST     11,MGSS                  STORE BLK NUMBER FOR PRMMSG
```

```
       ST     11,NXB                  STORE IT FOR COMPARE
*
SND    LH      7,PATH
       XC     IPARML(40),IPARML        CLEAR PARMLIST
       IUCV SEND,PRMLIST=(5),DATA=PRMMSG,PRMMSG=MGSS,                 *
             TYPE=2WAY,PATHID=(7),TRGCLS=WRT     SEND IT
       BZ      OKS2           SEND OK BRANCH
       LR      7,5            OTHER WISE PRINT OUT THE PLIST
       DMSKEY NUCLEUS
       LINEDIT TEXT='BAD SEND PLIST     ',                            *
             DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = .....................................*
             ............................',SUB=(HEX4A,(7)),        *
             DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
*
OKS2   EQU *
       L     10,ENDBLK
       L     11,NXB                LOAD BLOCK NUMBER JUST SENT
       LA    11,1(11)              UP BLOCK COUNT
       CR    11,10                 LAST RECORD
       BH    FIN                   YES  GO TELL USER
       ST    11,NXB                NO SAVE THIS BLK NUMB
       ST    11,MGSS               AND STORE IT FOR PRMMSG
       B      SND
*
ERRSET LINEDIT TEXT='=BAD SET'
*
FIN    WRTERM  'SEND ALL DONE ENTER GO TO EXIT'
       NI     ECB1,X'00'           WAIT FOR SEND TO COMPLETE
       WAIT   ECB=ECB1
       WRTERM  BUF,1024,EDIT=LONG,COLOR=B
*
EXIT   HNDIUCV CLR,NAME=SMPPGM
       L      14,SAVE
       BR     14
       DROP   5
*
OUT         EQU    *               OUT IS IMMCMD EXIT TO TURN OFF WAIT
            DROP 12
            LR    9,15
            LR    8,14
            USING  OUT,9
            OI     ECB1,X'C0'
            BR     8
*
BCON   DS     0D
       STM  0,15,0(13)
       LR   12,15
       USING  BCON,12
       USING  IPARML,2
       CLI    IPTYPE,X'03'             CHECK FOR SEVER
       BE     SEV                      GO SHOW WHY
       CLI    IPTYPE,X'02'        CONNECTION COMPLETE
       BNE    NOTC                    BRANCH IF NOT OTHERWISE
       MVC    ENDBLK(4),IPUSER+4     STOR LAST BLOCK
       LH    7,IPPATHID              GET PATHID
       STH   7,PATH                  STOR IT
       B      OT                   GO SEND FIRST RECORD
*
NOTC   CLI    IPTYPE,X'07'         MSG COMPLETE
       BNE    OT                   IF NOT RETURN OTHERWISE
       LH    7,IPPATHID            GET PATHID
       STH   7,PATH                SAVE IT
       L     5,IPRMMSG1            GET RETURN CODE
       LTR   5,5                   CHECK IT
       BZ    OT                    IF ZERO BRANCH OTHERWISE PRINT BUF
       DMSKEY NUCLEUS
       LR   7,2
       LINEDIT TEXT='BAD SEND PLIST FOLLOWS',                         *
             DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = .....................................*
```

```
               ..............................',SUB=(HEX4A,(7)),        *
               DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       B     OT                    AND RETURN
*
*     IF SEVER IPTYPE SHOW BUFFER
SEV    DMSKEY NUCLEUS
       LR   7,2
       LINEDIT TEXT='SEVER PLIST FOLLOWS',                            *
               DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
       DMSKEY NUCLEUS
       LINEDIT TEXT='BUFFER = ........................................*
               ..............................',SUB=(HEX4A,(7)),        *
               DISP=SIO,TYPCALL=NONE
       L    15,ADMSERR
       BALR 14,15
       DMSKEY RESET
*
OT     LM   0,15,0(13)
       BR   14
*
ALIGN  DS   0D
MSK    DC   X'00'
PLIST  DS   5D
WRT    DC   F'02'
ERRBLK DC   F'400'
ECB1   DC   F'0'
ENDBLK DC   F'0'
SAVE   DC   F'0'
SMPPGM DC  CL8'SMPPGM'
PACK01 DC  F'0'
OFFSET DC  F'0'
VDEVA  DC  H'0'
       DC  3H'0'
ID     DC  CL8'*BLOCKIO'
BDISK  DC  CL8'DISKID'
       DC  CL8'BLKPGM'
DSKAD  DS  XL2
DBLK   DS  H
DSKOFF DS  F
R15    EQU  15
MGSS   DC   F'0'
       DC   AL4(BUF)
TEN    DC   H'255'
BUF    DC   1024CL1' '
NXB    DC    F'0'
PATH   DC    H'0'
       COPY IPARML
       NUCON
       END
```

Appendix C. DIAGNOSE Code X'68' and VMCF

DIAGNOSE Code X'68'

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'68' to initiate a function of the Virtual Machine Communication Facility (VMCF).

Note: VMCF is maintained in z/VM only for compatibility; all new programs should use APPC/VM for communication. See the appendixes in the VM/ESA V2.4: Conversion Guide and Notebook (publibfp.dhe.ibm.com/epubs/pdf/hcse9a10.pdf), GC24-5839 for information on how to migrate your programs to APPC/VM or IUCV.

VMCF provides virtual machines with the ability to send data to, and receive data from, any other virtual machine.

VMCF is made up of control functions, data transfer functions, the VMCPARM parameter list, a special external interrupt (code X'4001') to asynchronously alert virtual machines to pending messages, and an external interrupt message header (VMCMHDR) to pass control information to another user.

See “The Virtual Machine Communication Facility” on page 989 for more information on VMCF protocol and details of VMCF functions.

Notes:

1. Before you can use any other VMCF function, you must use the AUTHORIZE function for communications. Before you can communicate with another user, that user must also have used the AUTHORIZE function.
2. In an XC virtual machine, DIAGNOSE code X'68' may not be issued in access-register mode.

Entry Values:

Rx
Contains the address of a parameter list (VMCPARM). The address of VMCPARM must be in guest real storage in the host-primary address space and must be doubleword-aligned. (See “The VMCPARM Parameter List” on page 1000 for the format of VMCPARM and a description of the contents of each of its fields.) One of the entries in this parameter list is a subcode, specifying the particular request being initiated. The functions and their subcodes are listed under the description of the VMCPFUNC field.

Ry
Not used.

Exit Values:

Ry
Contains the return code upon successful or unsuccessful completion of DIAGNOSE code X'68' function invocation. For a list of VMCF return codes and their meanings, see Table 207 on page 988 or Table 212 on page 1007. Note that return codes also are found in Final Response Interrupts in the VMCMEFLG field. See Table 213 on page 1009.

Usage Notes

1. Rx and Ry can be any general register, R0 through R15. They may also be the same register.
2. You may not be authorized to issue this DIAGNOSE code if an external security manager is installed on your system. For additional information, contact your security administrator.

© Copyright IBM Corp. 1991, 2020

Responses

Return Codes: The virtual machine initiating a VMCF request receives a return code that may be returned in the general register specified as Ry in the DIAGNOSE instruction or in VMCMEFLG upon receiving an interrupt associated with a data transfer operation. The return code indicates successful completion of the request or error conditions associated with the request.

Upon completion of DIAGNOSE code X'68', the following return codes may be received:

Table 207. VMCF Return Codes from DIAGNOSE code X'68'

Return Code in Ry	Meaning
0 (X'00')	Successful completion of a request
1 (X'01')	Invalid buffer address or length
2 (X'02')	Invalid subcode
3 (X'03')	Protocol violation
4 (X'04')	Source virtual machine not authorized
5 (X'05')	User not available
6 (X'06')	Store or fetch protection violation
7 (X'07')	SENDX data too large
8 (X'08')	Duplicate message
9 (X'09')	Target virtual machine in quiesce status
10 (X'0A')	Message limit for outgoing messages is exceeded by source or message limit for incoming messages at target exceeded. The outgoing message limit may be changed using the SETLIMIT function. A VMCF user, using the directory option MAXVMCFI, may specify an incoming message limit from 1 to 2,147,483,647. The default is 2,147,483,647.
11 (X'0B')	REPLY cancelled
12 (X'0C')	Message not found
13 (X'0D')	Synchronization error
14 (X'0E')	CANCEL too late
15 (X'0F')	Paging I/O error
16 (X'10')	Incorrect length
17 (X'11')	Destructive overlap. A virtual machine executed a RECEIVE or REPLY function and specified a receive buffer address that overlapped the source virtual machine send data address or specified a reply data address that overlapped the source virtual machine reply buffer address.
18 (X'12')	User not authorized for priority messages
19 (X'13')	Data transfer error
20 (X'14')	Cancel–busy. A virtual machine attempted to cancel a message being processed. If this is a SEND/RECV request and the RECEIVE function is in process, repeated retries may cancel the REPLY function.

Program Exceptions: These program exceptions can occur if the DIAGNOSE X'68' is given incorrect input data:

Problem Encountered	Cause
Special-operation exception	DIAGNOSE code X'68' cannot run in an XC virtual machine that is in access register mode.
Privilege-operation exception	The virtual machine is in the problem state.
Access exception (See “Access Exceptions” on page 8.)	An error occurred trying to fetch the parameter list.

The Virtual Machine Communication Facility

Note: VMCF is maintained in z/VM only for compatibility; all new programs should use APPC/VM for communication. See VM/ESA V2.4: Conversion Guide and Notebook (publibfp.dhe.ibm.com/epubs/pdf/hcse9a10.pdf), GC24-5839 for information on how to migrate your programs to APPC/VM or IUCV.

The Virtual Machine Communication Facility (VMCF) is part of the CP component of VM. VMCF provides virtual machines with the ability to send data to and receive data from any other virtual machine.

VMCF is made up of five data transfer functions, seven control functions, a special external interrupt (code X'4001') to asynchronously alert virtual machines to pending messages, and an external interrupt message header to pass control information (and data, at times) to another user.

VMCF is implemented by means of functions invoked using the DIAGNOSE instruction code X'68' and a special 40-byte parameter list called VMCPARM. (VMCPARM DSECT is in HCPGPI MACLIB.) A VMCF function is indicated by a particular function subcode in the VMCPFUNC field in the parameter list.

Note: Before you can use any other VMCF function, you must use the AUTHORIZE function for communications. Before you can communicate with another user, that user must also have used the AUTHORIZE function.

A special external interrupt (code X'4001') notifies one virtual machine of a pending transfer of data. This interrupt is also used to synchronize sending and receiving of data.

Along with this interrupt, the virtual machine receives a message header that is logged into a preassigned virtual storage area called VMCMHDR. (VMCMHDR DSECT is in HCPGPI MACLIB.) This message header is used to define the type of request and to provide data transfer information, such as length of data. The message header is also used to notify the originator of a transaction of the success or failure of the transaction. In this case, the message header includes such information as residual counts and data transfer return codes.

See Table 208 on page 989 for a list of the VMCF functions and a brief description of each. The functions are described in detail starting on page “Descriptions of VMCF Functions” on page 995.

Messages and data are directed to other virtual machines logically through the user ID. The amount of data that can be moved in a single transfer is limited only by the sizes of virtual machine storage of the respective virtual machines. Use of real storage is minimal. Only 1 real storage page per virtual machine (a total of 2 pages, 1 for the sender and 1 for the receiver) needs to be locked during the data transfer.

The special message facility (SMSG) uses VMCF to send messages from 1 virtual machine storage area to another virtual machine storage area. For a description of the special message facility and how it uses VMCF, see Appendix D, “The Special Message Facility,” on page 1011.

Table 208 on page 989 describes the purpose of each function.

Table 208. Virtual Machine Communication Facility (VMCF) Functions

Function	Type	Description
AUTHORIZE *	Control	Initializes VMCF for a given virtual processor. Once AUTHORIZE is executed, the virtual processor can execute other VMCF functions and receive messages or requests from other users.

Table 208. Virtual Machine Communication Facility (VMCF) Functions (continued)

Function	Type	Description
UNAUTHORIZE *	Control	Terminates VMCF activity on a virtual processor
SEND	Data Transfer	Directs a message or block of data to another virtual machine
SEND/RECV	Data Transfer	Directs a message or block of data to another virtual machine, and requests notification of a reply
SENDX	Data Transfer	Directs data to another virtual machine on a faster but more restrictive protocol than the SEND function
RECEIVE	Data Transfer	Allows you to accept selective messages or data sent through a SEND or SEND/RECV function
SETLIMIT	Control	Allows you to set the maximum number of outgoing messages that you can have pending.
CANCEL	Control	Cancels a message or data transfer directed to another user but not yet accepted by that user
REPLY	Data Transfer	Allows you to direct data back to the originator of a SEND/RECV function, simulating full duplex communication
QUIESCE *	Control	Temporarily rejects further SEND, SENDX, SEND/RECV, or IDENTIFY requests from other users to the virtual machine
RESUME *	Control	Resets the status set by the QUIESCE function and allows execution of subsequent requests from other users to the virtual machine
IDENTIFY	Control	Notifies another user that your virtual machine is available for VMCF communication
REJECT	Control	Allows you to reject specific SEND or SEND/RECV requests pending for your virtual machine

*These functions have different meanings in a virtual MP environment. For more information about a virtual MP environment, see “VMCF in an MP Environment” on page 1006.

Using the Virtual Machine Communication Facility

The following discussion presents ideas and suggestions for using the Virtual Machine Communication Facility (VMCF).

VMCF Applications

The VM system with VMCF provides the user with the potential to apply new and different techniques to current applications.

Resource Sharing

VMCF provides a clear and concise method for sharing and serializing resources between virtual machines. The resources can range from multi-write minidisks to entire processors. The control functions for resource sharing (such as resource management, serialization) can be contained in a virtual machine.

Virtual Extensions to VM

It is conceivable that functions could be added to VM without altering the control program (CP). A special privilege class virtual machine could be used to provide additional functions to nonprivilege class users

using the VMCF interface. Similarly, CMS capabilities could be expanded (or at least appear to be expanded) by linking CMS with other virtual machines.

Program Testing

The program testing capabilities offered by VMCF can range from device simulation to teleprocessing network simulation. In particular, VMCF can be used to provide external interactions from one virtual machine to another. A simulated teleprocessing network could be constructed with virtual machines. Each virtual machine would effectively become a node within the network. The network structure could range from a simple tree type structure to a complicated multi-path mesh type structure. The program logic within each node virtual machine would be the same logic as required for a real teleprocessing node. In theory, a reasonably complicated structure could be simulated without requiring the physical hardware.

The significant testing capability provided by VMCF is the ability to link the test system with test/simulation routines in another virtual machine.

Intra-Virtual Machine Communication

Although the VMCF interface is intended for communication from one virtual machine to another, it can also be used to communicate within a single virtual machine (wrap connection). The VMCF interface could conceivably be used to link one or more operating system tasks that are logically separated by the software. This would allow task-to-task communication rather than virtual machine-to-virtual machine communication.

Security and Data Integrity

The VMCF interface provides the following security aids:

- The user doubleword in the external interrupt message header can be used to contain a security code to prevent unwarranted users from accessing a shared database or other confidential information.
- The AUTHORIZE SPECIFIC option allows a user to restrict messages sent to his virtual machine. This option is useful when slave machines are to communicate only with a host machine. The slave machines can AUTHORIZE SPECIFIC with the host and prevent unwarranted users from clogging their message queues.
- The design of VMCF prevents malicious users from intercepting transactions in process for other users (for example, user D cannot execute a RECEIVE, REPLY, REJECT or CANCEL to a message sent to user B from user A).

The VMCF support module is designed so a user is always informed of conditions that could threaten the integrity of his own data. The user is notified either with a DIAGNOSE code X'68' return code or data transfer error code. There is no internal buffering of user data within the control program (CP). A message is always retained by either the SOURCE or TARGET virtual machine. If a SEND type request fails, the SOURCE still has a copy of the original message. If a TARGET REPLY fails, the TARGET user still has a copy of the REPLY data. The DIAGNOSE return code or data transfer error code can indicate to a user that a transaction failed. It is up to the user to preserve the associated transaction data. A VMCF user should consider the following notes:

1. The buffer used for SOURCE data in a SEND, SENDX or SEND/RECV request should not be freed or reused until the final response external interrupt is received by the SOURCE.
2. The buffer used for TARGET data in a REPLY function can be reused by the TARGET after the DIAGNOSE instruction (REPLY) has successfully completed.
3. The user parameter list, VMCPARM, may be re-used upon completion of the DIAGNOSE instruction. At that point the VMCPARM data has been copied to a VMCF control block by the control program. A user should, however, maintain queues of VMCPARM data to associate an external interrupt message header, VMCMHDR, with a particular request.
4. A user should always interrogate the DIAGNOSE return code or data transfer error code for possible error conditions. It is the user's responsibility to determine the types and extent of error recovery. The DIAGNOSE return code 19 (X'13') for a SOURCE SEND, SEND/RECV or SENDX request indicates that an error was associated with the TARGET user and for a TARGET RECEIVE or REPLY request indicates

that an error was associated with the SOURCE user. The user who receives this return code does not have to invoke error recovery for himself but only be aware that the transaction did not complete successfully because of an error associated with the other user.

Performance Considerations

There are several factors that can affect the performance of VMCF:

- It is to a user's benefit to have the user parameter list, VMCPARM, in the same 4K page as the DIAGNOSE code X'68' instruction. This may eliminate a paging operation.
- User support modules using the VMCF interface should be written as reentrant modules and be contained within a CP shared segment whenever possible. This helps reduce CP paging overhead.
- For applications that involve serial message processing, the SENDX function is the most efficient. The SENDX function eliminates the need for the TARGET to do a RECEIVE operation.

Note: Overall system VM performance is not affected when VMCF is not being used by an installation.

General Considerations

The SENDX function is a fast way to transfer messages or data and can be used in place of the CP MSG command where the message length exceeds the capacity of the terminal input line. Its use is somewhat restricted in that the maximum data length must be agreed upon by all VMCF users and then remains fixed unless renegotiated.

The SEND and SEND/RECV functions are better suited to transfer high volume database type information. This type of data transfer requires the flexibility of a wide range of data lengths along with rigorous management and control techniques.

The QUIESCE function allows a virtual machine to stop receiving messages. The virtual machine can process those messages already stacked and then use the RESUME function to continue reception. The QUIESCE function also allows a virtual machine to process all queued messages prior to terminating VMCF operation.

The user parameter list, VMCPARM, is designed so it can be used for any function by simply varying the contents of its fields.

Users should keep copies of VMCPARMs for all requests made through the SEND, SEND/RECV, or SENDX functions. When a final response interrupt is received and the interrupt message header indicates no data transfer errors, the corresponding VMCPARM copy can be released. If a data transfer error is indicated, the copy can be used to reinitiate the transaction.

VMCF Protocol

VMCF provides four types of protocol:

- SEND
- SEND/RECV
- SENDX
- IDENTIFY.

The protocol used to communicate between two virtual machines depends on the application of VMCF and conventions established by virtual machine users authorized to use VMCF. A virtual machine must invoke the AUTHORIZE function before it is allowed to use any of the other functions.

The types of transactions that virtual machines can be involved in are described by a series of VMCF protocols. In these protocols the originating virtual machine is called the *source* virtual machine. The destination virtual machine is called the *target* virtual machine.

The protocol for a transaction remains in effect for the duration of the transaction.

The SEND Protocol

The SEND protocol defines a one-way transfer of data from source virtual machine storage to target virtual machine storage. The SEND protocol uses the SEND and RECEIVE functions, as described in Figure 107 on page 993. The source virtual machine first transfers data to the target virtual machine. This is done by executing the SEND function which specifies the user ID of the target virtual machine, a message ID, and the address and length of the data being sent. The target virtual machine receives an external interrupt from CP notifying it of the data transfer request. The target virtual machine can then respond through the RECEIVE function. The RECEIVE request specifies the address and the length of the TARGET buffer that is to receive the data and causes the data to be transferred from source virtual machine storage to target virtual machine storage. When the data transfer is complete, the source virtual machine receives an external interrupt from CP, indicating that the transaction is complete and that the target virtual machine has received the data.

All virtual machines authorized to use VMCF can send data using this protocol.

The amount of data transferred is limited only by virtual machine storage size. Data is transferred in blocks of up to 2K (when necessary) and only one real page frame is locked during the data transfer operation.

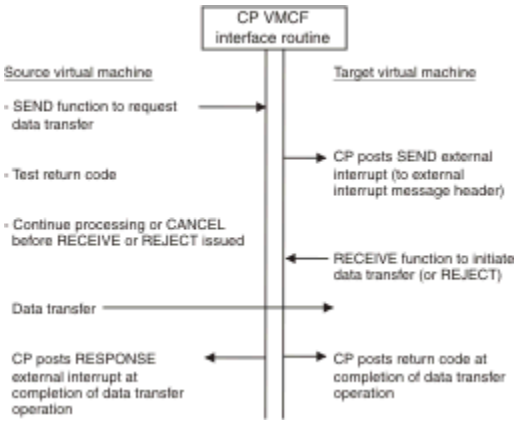


Figure 107. The SEND Protocol

The SEND/RECV Protocol

The SEND/RECV protocol defines a transaction calling for two-way transfer of data, as described in Figure 108 on page 994. The SEND/RECV protocol uses the SEND/RECV, RECEIVE, and REPLY functions.

The source virtual machine initiates the transaction using the SEND/RECV function. Using an external interrupt, CP notifies the target virtual machine that there is a message waiting. The target virtual machine uses the RECEIVE function to cause the data to be transferred from the source virtual machine's storage to the target virtual machine storage. The target virtual machine now uses the REPLY function to cause data to be transferred from its storage to the source virtual machine's storage. When the REPLY function completes processing, CP causes an external interrupt in the source virtual machine, notifying it that the transaction is complete.

The SEND/RECV request requires that the source virtual machine specify the address and length of the data to be transferred and the address where data is expected from the REPLY function. (Both addresses are in source virtual machine storage.) These addresses, along with the length of the data to be transferred, are specified through the VMCPARM parameter list. See “The VMCPARM Parameter List” on page 1000.

When RECEIVE is issued by the target virtual machine in response to the SEND/RECV request, VMCPARM contains the address in target virtual machine storage where data is to be received. Finally, when the REPLY request is issued, VMCPARM contains the address in the target virtual machine storage from which data is to be transferred.

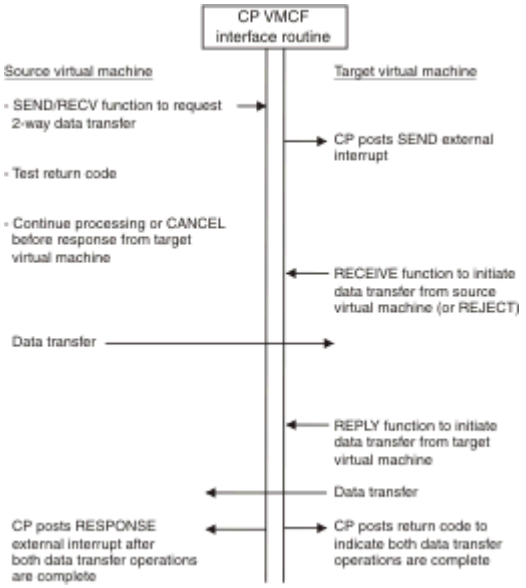


Figure 108. The SEND/RECV Protocol

The SENDX Protocol

The SENDX protocol defines a transaction calling for an expedited one-way transfer of data. Figure 109 on page 995 shows the SENDX protocol visually. SENDX differs from the SEND protocol in that the target virtual machine need not issue the RECEIVE function; data is transferred from source virtual machine storage to target virtual machine storage at the same time the external interrupt from CP notifies the target virtual machine of the transaction. Data sent by the source virtual machine is placed in the external interrupt buffer of the target virtual machine.

Virtual machines using the SENDX protocol are responsible for specifying the user ID for the target virtual machine, a message ID, the address and length of the data being sent, and the external interrupt buffer address and data length for the target virtual machine. A virtual machine to be used as a target virtual machine with the SENDX protocol must specify this information through VMCPARM when that virtual machine issues the AUTHORIZE function. The data length specified must be at least as long as the maximum amount of data to be transferred during a transaction; it need not be limited to the usual 40-byte external interrupt buffer. Effective use of the SENDX protocol requires that VMCF users agree on a maximum size for SENDX data and then issue the AUTHORIZE function with the appropriate external interrupt buffer size.

If the target virtual machine has not provided enough SENDX buffer area in the external interrupt buffer, CP notifies the source virtual machine that the transaction was not completed.

When a SENDX data transfer is complete, CP directs a response external interrupt to the source virtual machine, notifying it that the transaction is complete.

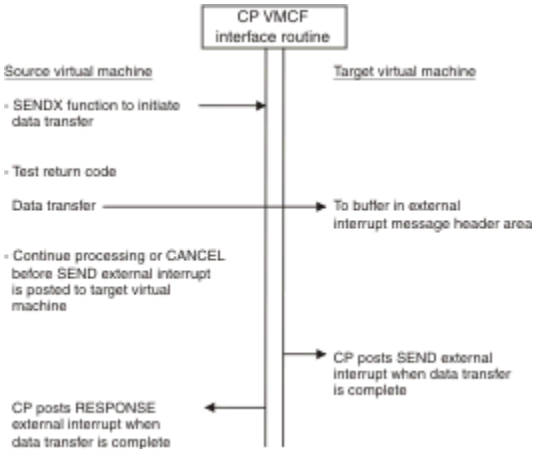


Figure 109. The SENDX Protocol

The IDENTIFY Protocol

The IDENTIFY protocol defines a means for virtual machines to identify themselves to other virtual machines by passing user-defined control information through a standard VMCF message header. Figure 110 on page 995 shows the IDENTIFY protocol visually.

When the IDENTIFY function is issued, CP directs an external interrupt to the target virtual machine. Along with the external interrupt, the target virtual machine receives a standard VMCF message header that contains user-defined information. The IDENTIFY protocol does not cause a response external interrupt to be directed to the source virtual machine.

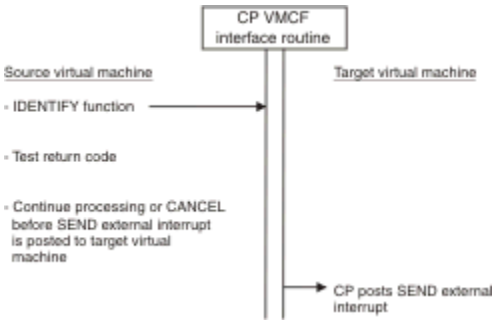


Figure 110. The IDENTIFY Protocol

Descriptions of VMCF Functions

There are two types of VMCF functions:

- Control functions
- Data transfer functions.

The Control Functions

The VMCF control functions allow efficient management of data transfer operations from your virtual machine console. The control functions are:

- AUTHORIZE
- UNAUTHORIZE
- CANCEL
- QUIESCE
- RESUME
- IDENTIFY
- REJECT.

AUTHORIZE: DIAGNOSE Code X'68' Subcode X'0000'

AUTHORIZE enables VMCF for a virtual machine; once an AUTHORIZE has been executed, the virtual machine can execute other VMCF functions and receive messages and data from other authorized VMCF virtual machines. It is possible to specify three options with the AUTHORIZE function: SPECIFIC, PRIORITY, and VMCPSMSG.

The SPECIFIC option authorizes communication with a specific virtual machine. Any messages sent to the virtual machine from other than the specified virtual machine will be rejected. The SPECIFIC option can be used in an application where virtual machines desire to communicate with a master controller but not among themselves. Under the special message facility, CP is authorized with every virtual machine that is to receive messages sent with the SMSG command. Virtual machines that are to receive messages must authorize themselves. The VMCPSMSG option allows a virtual machine to receive special messages.

The PRIORITY option allows a virtual machine to authorize the receipt of priority messages. A virtual machine is allowed to send priority messages to another virtual machine only if the other virtual machine is authorized to receive priority messages. A priority message is queued ahead of nonpriority messages and therefore accepted first.

When you execute the AUTHORIZE function, you must specify the address and length of the external interrupt buffer for your virtual machine. The buffer must be large enough to contain a fixed message header (40 bytes). The message header identifies messages sent by other virtual machines or responses to messages you might send to your own virtual machine.

If you are going to accept SENDX-type communications, you must specify the size of the external interrupt buffer as 40 plus the maximum size of SENDX data that you plan to accept. This has the effect of authorizing SENDX protocol. That is, a virtual machine may receive data along with the external interrupt in its external interrupt buffer. When a virtual machine sends data to another virtual machine through the SENDX function the data must fit in that virtual machine's external interrupt buffer or the function is rejected. It is recommended that you specify a buffer length of 280 bytes.

Any AUTHORIZE options in effect can be reset or changed by executing the AUTHORIZE function again. If there are errors during execution of the AUTHORIZE function, a virtual machine's authorization status is not changed.

UNAUTHORIZE: DIAGNOSE Code X'68' Subcode X'0001'

UNAUTHORIZE terminates VMCF activity for a virtual machine. The UNAUTHORIZE function causes any stacked or queued messages associated with the virtual machine to be purged. A virtual machine should execute the QUIESCE function before executing UNAUTHORIZE if messages that are already queued are to be handled. When a virtual machine executing UNAUTHORIZE has pending final response external interrupts, the interrupts are purged. If a virtual machine has pending SEND external interrupts from another source virtual machine, a RESPONSE interrupt is reflected to the source indicating that the virtual machine is no longer available.

CANCEL: DIAGNOSE Code X'68' Subcode X'0006'

CANCEL cancels a message or data transfer pending for but not accepted by another VMCF virtual machine. A virtual machine can CANCEL messages it originates with SEND, SENDX, or SEND/RECV functions. A message cannot be cancelled if any of the following conditions exist and the request was:

- SENDX or IDENTIFY and the target had already received the SEND external interrupt.
- SEND and the target had already executed the RECEIVE or REJECT functions.
- SEND/RECV and the target had already executed the REPLY or REJECT functions.

If the original request was SEND/RECV and the target virtual machine had executed the RECEIVE function but not the REPLY, the REPLY can be cancelled. A virtual machine is notified of this condition with a DIAGNOSE return code. (For a description of the return codes, see Table 210 on page 1003.)

QUIESCE: DIAGNOSE Code X'68' Subcode X'0008'

QUIESCE temporarily rejects SEND, SENDX, SEND/RECV, or IDENTIFY requests from other virtual machines. QUIESCE allows a virtual machine to receive any stacked or queued messages but reject

further SEND, SENDX, IDENTIFY, or SEND/RECV requests from other virtual machines. QUIESCE can be used to indicate to other virtual machines that the virtual machine is in QUIESCE status, authorized for communication but not able to accept messages at this time (for example, entering slowdown, my buffers are full, try again later). The IDENTIFY function could be used to inform other virtual machines that a particular user is no longer in QUIESCE status. You should execute the QUIESCE function before executing the UNAUTHORIZE function to avoid losing messages (see "UNAUTHORIZE: DIAGNOSE Code X'68' Subcode X'0001'" on page 996). A virtual machine can reset the QUIESCE status (exit slowdown) by executing the RESUME function. (See "RESUME: DIAGNOSE Code X'68' Subcode X'0009'" on page 997.) A virtual machine in QUIESCE status may continue to send messages to other virtual machines. QUIESCE status for a virtual machine only affects messages sent from other virtual machines.

RESUME: DIAGNOSE Code X'68' Subcode X'0009'

RESUME cancels the QUIESCE status, allowing your virtual machine to resume reception of VMCF requests from other virtual machines. You can use the IDENTIFY function to inform other virtual machines that your virtual machine is no longer in QUIESCE status. (See "IDENTIFY: DIAGNOSE Code X'68' Subcode X'000A'" on page 997.)

IDENTIFY: DIAGNOSE Code X'68' Subcode X'000A'

IDENTIFY notifies another virtual machine that your virtual machine is available for VMCF communication. Use the IDENTIFY function after issuing the AUTHORIZE function or after your virtual machine has been in the VMCF QUIESCE state and you have issued the RESUME function. IDENTIFY causes an external interrupt to be stacked for a specified virtual machine. The virtual machine executing the IDENTIFY function specifies the user ID of the user to receive the external interrupt. The external interrupt identifies the virtual machine executing the IDENTIFY function. The IDENTIFY function is provided to inform a host or controller virtual machine that a virtual machine is activated (logged on) and ready for VMCF communication. The IDENTIFY function can also be used to inform other virtual machines that your virtual machine has exited QUIESCE state. There is no response external interrupt associated with the IDENTIFY function.

The IDENTIFY function can also be used to pass virtual machine defined control information. The fields in the VMCF parameter list, VMCPARM, not used by the IDENTIFY function may be used to contain additional virtual machine data.

REJECT: DIAGNOSE Code X'68' Subcode X'000B'

REJECT selectively rejects pending SEND or SEND/RECV requests from other VMCF virtual machines. REJECT causes a response external interrupt to be reflected to the originator of a message. The external interrupt indicates to the originator that the message was rejected. (See VMCMRJCT on page "The External Interrupt Message Header (VMCMHDR)" on page 1004.) The user doubleword within the external interrupt header may tell a user why the message was rejected. (See VMCPUSE on page "The VMCPARM Parameter List" on page 1000 and VMCMUSE on page "The External Interrupt Message Header (VMCMHDR)" on page 1004.) When the user of a virtual machine executes the REJECT function, he specifies within the VMCF parameter list, VMCPARM, the message ID of the message to be rejected. A virtual machine cannot reject a message sent with the SENDX function since the message is received at the same time the external interrupt is received. The REJECT function can be executed as response to either SEND or SEND/RECV requests.

The Data Transfer Functions

The data transfer functions are:

- SEND
- SEND/RECV
- SENDX
- RECEIVE
- REPLY
- SETLIMIT

These operations involve the movement of data from one virtual machine storage to another virtual machine storage.

SEND: DIAGNOSE Code X'68' Subcode X'0002'

SEND directs a message or block of data to another virtual machine. Specify the virtual address and length of data to be sent within the user parameter list, VMCPARM. Also, specify in the parameter list a message ID to be associated with the message and the user ID of the user to receive the message data. You can also send a doubleword of data to be transmitted within the external interrupt message header (refer to the section "VMCF User Doubleword" on page 1006). If the SEND function is executed with a data length of 0, only the user doubleword is transmitted to the target virtual machine. The target virtual machine can then respond with a RECEIVE function (0 length) and pass back a doubleword of data to the source virtual machine. The external interrupt message header identifies the SEND request. When the target virtual machine executes a RECEIVE function, the message is transmitted from the source virtual machine storage to the target virtual storage. There is no internal buffering of data within the control program (CP). When the data transfer function is complete, the source virtual machine receives a response external interrupt indicating that the SEND request is complete. The target virtual machine receives a DIAGNOSE code X'68' return code indicating that the RECEIVE function is complete. The return code can indicate error conditions associated with the RECEIVE function or normal completion.

The target virtual machine has the option to reject a message rather than execute the RECEIVE function. (See "REJECT: DIAGNOSE Code X'68' Subcode X'000B'" on page 997.) The source virtual machine may cancel a SEND request before the target virtual machine has executed a RECEIVE function or REJECT function. (See "CANCEL: DIAGNOSE Code X'68' Subcode X'0006'" on page 996.)

If you are executing the SEND function, you may specify the PRIORITY option. The PRIORITY option causes the external interrupt for the target virtual machine to be queued ahead of all other nonpriority external interrupts. If there are other PRIORITY external interrupts pending for the target virtual machine, the queuing is done in a first-in-first-out manner. That is, PRIORITY interrupts are queued FIFO among themselves but ahead of all nonpriority interrupts.

SEND/RECV: DIAGNOSE Code X'68' Subcode X'0003'

SEND/RECV provides the capability to both send and receive data in a single VMCF transaction. The SEND/RECV function causes an external interrupt to be queued for the target virtual machine. When the target virtual machine receives the external interrupt, it can respond with the RECEIVE function. The RECEIVE function causes data to be transferred from the source virtual storage to target virtual storage. The target virtual machine can then respond with a REPLY function. The REPLY function causes data to be transferred from specified target virtual storage to a reply buffer in the source virtual storage. The source virtual machine then receives a response external interrupt indicating that the SEND/RECV request is complete.

When the source virtual machine executes the SEND/RECV function it specifies the address and length of both the SEND buffer and reply buffer. The address and length specifications are contained within the user parameter list, VMCPARM. The user parameter list also contains a message ID and user ID of the user to receive the data. (See the "The VMCPARM Parameter List" on page 1000.)

The source virtual machine can cancel a previously executed SEND/RECV request provided the target virtual machine has not yet executed the REPLY or REJECT function. If the target virtual machine has already executed the RECEIVE function, only the REPLY can be cancelled. (See "CANCEL: DIAGNOSE Code X'68' Subcode X'0006'" on page 996.)

The target virtual machine can execute the REJECT function in response to the SEND/RECV request and cause the entire operation to be terminated. (See "REJECT: DIAGNOSE Code X'68' Subcode X'000B'" on page 997.)

The target virtual machine can respond to a SEND/RECV request with the REPLY function without executing the RECEIVE function. This has the effect of informing the source virtual machine that the target virtual machine cannot accept data but that it can send data. The source virtual machine could have executed the SEND/RECV function only as a means to solicit data from the target virtual machine. The application of this protocol is up to VMCF users. The user doubleword can be used as a means to control such an application. (See "VMCF User Doubleword" on page 1006.)

You can execute a SEND/RECV request using the PRIORITY option. The PRIORITY option causes the target external interrupt for the SEND/RECV request to be queued ahead of any other nonpriority external interrupts. Response external interrupts directed to the source of a PRIORITY message are also queued in priority order.

SENDX: DIAGNOSE Code X'68' Subcode X'0004'

SENDX directs data to another virtual machine through a faster but more restrictive protocol than the SEND function. SENDX function data reaches the target virtual machine at the same time the SEND external interrupt reaches the target. To use the SENDX function, the target virtual machine must have an external interrupt buffer large enough to contain both the standard message header and the data. The size of the external interrupt buffer is specified when you execute the AUTHORIZE function. Attempts to execute SENDX are rejected when the target virtual machine's external interrupt buffer is not large enough to contain the data. After the target virtual machine receives the SEND external interrupt and data, a response external interrupt is directed to the source virtual machine. The SENDX function eliminates the need for a target virtual machine to execute a RECEIVE function.

A SENDX request can be cancelled by the source virtual machine provided the SENDX external interrupt has not yet been reflected to the target virtual machine. (See “CANCEL: DIAGNOSE Code X'68' Subcode X'0006'” on page 996.)

Specify the SENDX buffer address and length in the user parameter list, VMCPARM. The message ID and user ID of the target virtual machine are also specified in VMCPARM.

The SENDX function can be executed with the PRIORITY option allowing the SEND external interrupt to be queued ahead of all nonpriority external interrupts for the target virtual machine.

A SENDX request cannot be rejected by the target virtual machine since the message is received at the same time the external interrupt is received.

You can execute the SENDX function with a 0 data length causing only the message header and user doubleword to be transmitted.

RECEIVE: DIAGNOSE Code X'68' Subcode X'0005'

RECEIVE allows you to selectively accept messages or data sent through the SEND or SEND/RECV functions. You must specify in the user parameter list, VMCPARM, the virtual address and length of the RECEIVE buffer. The parameter list also contains the message ID of the message to be received and user ID of the virtual machine that originated the SEND or SEND/RECV request. When a virtual machine has more than one message pending, the RECEIVE function can be executed to select messages in any order by message ID.

You can execute the REJECT function to reject messages sent by other virtual machines. The REJECT function terminates the SEND or SEND/RECV request. (See “REJECT: DIAGNOSE Code X'68' Subcode X'000B'” on page 997.)

You can execute the RECEIVE function in response to a SEND/RECV request and then execute a REJECT function rather than a REPLY. The user doubleword passed back with the REJECT function could indicate RESEND, for example, if the original data was not received correctly (depending on how you want to use the protocol).

REPLY: DIAGNOSE Code X'68' Subcode X'0007'

REPLY allows you to direct data back to the sender of a SEND/RECV function. This simulates full duplex communication. The REPLY function is used with the SEND/RECV function. A user who receives a SEND/RECV external interrupt normally responds by executing the RECEIVE function. The RECEIVE function causes data to be transferred from the source virtual storage to the target virtual storage. The target virtual machine can then respond with the REPLY function causing data to be transferred from specified target virtual storage to the source virtual storage. The REPLY function causes a response external interrupt to be reflected to the source virtual machine.

The user parameter list, VMCPARM identifies the virtual buffer address and length of reply data. When the REPLY function is executed, the user parameter list, VMCPARM, also contains the message ID and the user ID of the virtual machine to receive the reply.

The REPLY function can be executed with a 0 data length indicating no response. You can transmit a reply, 0 length or otherwise, using the user doubleword.

A reply can be executed in response to a SEND/RECV request without executing the RECEIVE function. This indicates that you do not want to receive the message but may want to send a reply. A reply of 0 length could be executed simply to terminate the SEND/RECV request. The application of the REPLY function is a user decision. It must be used to terminate a SEND/RECV request, however, unless the REJECT function is executed. (See "REJECT: DIAGNOSE Code X'68' Subcode X'000B'" on page 997.) The reply is complete when the source virtual machine receives the external interrupt response.

A REPLY function cannot be executed in response to a SEND request, this is a protocol violation.

SETLIMIT: DIAGNOSE Code X'68' Subcode X'000C'

SETLIMIT determines the maximum number of VMCF messages that a virtual machine can have outstanding (that is, sent but not received, cancelled, or rejected) at any one time. When a virtual machine uses the AUTHORIZE function, the message limit is set to the default value of 50. Thereafter, the SETLIMIT function can be used to change the limit at any time. If the limit value is specified as zero, the limit is set to the default value of 50. Only the low-order halfword of the specified limit is used, so the maximum value that can be set is 65535.

If a virtual machine sets its maximum number of messages to a value less than its current number of outstanding outgoing VMCF messages, the user is not allowed to send additional messages until the number outstanding is reduced to less than the limit.

Invoking VMCF Functions

VMCF functions are invoked by means of:

- DIAGNOSE code X'68' subcodes
- The VMCPARM parameter list
- External interrupt code X'4001'
- The external interrupt message header (VMCMHDR).

DIAGNOSE Code X'68'

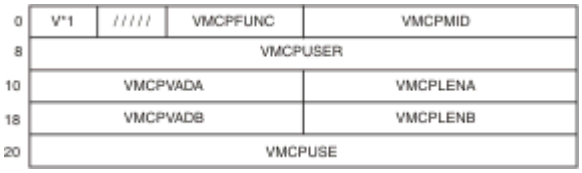
All VMCF functions are invoked from within assembler language programs by means of DIAGNOSE code X'68'. For more information, see "DIAGNOSE Code X'68'" on page 987.

The VMCPARM Parameter List

The Rx register of DIAGNOSE code X'68' contains the address of a parameter list, VMCPARM. This parameter list specifies the VMCF function to be executed, along with other information required by VMCF to execute that function. The address of VMCPARM must be doubleword-aligned and must be in second level storage (the storage that appears real to the virtual machine).

The following is the format of the VMCPARM parameter list, provided as VMCPARM COPY in the HCPGPI macro library:

VMCPARM DSECT



V*1 (VMCPFLG1)
is a flag byte used to specify options associated with a particular function. This flag byte can be set to the following values:

VMCPAUTS (X'80')
indicates, for the AUTHORIZE function, an AUTHORIZE SPECIFIC request. When this bit is set, the VMCPUSER field must contain the user ID of the target virtual machine. The target virtual machine is the receiving virtual machine. The status of the specified target virtual machine is not checked by the control program (CP) at this time.

VMCPPRTY (X'40')
indicates, for SEND, SEND/RECV, SENDX, and IDENTIFY requests, a priority message request. For an AUTHORIZE request, it indicates an AUTHORIZE PRIORITY request. You cannot send priority messages to another virtual machine unless that virtual machine has been authorized for priority messages. The send and response external interrupts for a priority message are queued ahead of pending nonpriority external interrupts.

VMCPSMSG (X'20')
indicates that the virtual machine is authorized to receive special messages. This bit can be turned either on or off by the SET SMSG ON command.

Bits 3 through 7 are reserved for IBM use.

VMCPFUNC
Contains the halfword DIAGNOSE code X'68' subcode that defines the VMCF function being requested as shown in Table 209 on page 1001.

Table 209. VMCF Function Codes for DIAGNOSE Code X'68'

Field Attributes in the DSECT	Hexadecimal Code	Function
VMCPAUTH	X'0000'	AUTHORIZE
VMCPUAUT	X'0001'	UNAUTHORIZE
VMCPSEND	X'0002'	SEND
VMCPSENR	X'0003'	SEND/RECV
VMCPSENX	X'0004'	SENDX
VMCPRECV	X'0005'	RECEIVE
VMCPCANC	X'0006'	CANCEL
VMCPREPL	X'0007'	REPLY
VMCPQUIE	X'0008'	QUIESCE
VMCPRESM	X'0009'	RESUME
VMCPIDEN	X'000A'	IDENTIFY
VMCPRJCT	X'000B'	REJECT
VMCPSETL	X'000C'	SETLIMIT

VMCPMID
contains a unique message identifier associated with a transaction. The source virtual machine must originate the message ID for SEND, SEND/RECV, and SENDX requests. The message ID is used by the target virtual machine (along with VMCPUSER) to respond to the source request with a RECEIVE, REPLY, or REJECT request. The message ID allows the target virtual machine to selectively Receive, REPLY, or REJECT messages when more than one message is enqueued. The message ID is used by both the source and target as a unique identification for all messages. You may send messages with the same message ID to multiple users; you cannot send multiple messages with the same message ID to one user. Once a transaction is completed, however, the message ID may be reused.

This field is also used to specify the message limit for the SETLIMIT function.

VMCPUSER
specifies the user ID of the target virtual machine for SEND, SEND/RECV, SENDX, IDENTIFY, and CANCEL requests, and the user ID of the source virtual machine for RECEIVE, REPLY, and REJECT

requests. The target virtual machine uses this field in combination with the message ID (VMCPMID) to respond to source requests. When the original source parameter list VMCPARM is passed to the target as the message header VMCMHDR, the user ID is changed from target to source.

This field is also used to specify the specific user ID for an AUTHORIZE SPECIFIC request.

VMCPVADA
contains one of the following addresses, depending upon which VMCF function is requested for:

- SEND, SEND/RECV, and SENDX requests, the address of the source virtual machine data.
- RECEIVE requests, the address of a target virtual machine receive buffer.
- REPLY requests, the address in target virtual machine storage where reply data is located.
- An AUTHORIZE request, the address of the virtual machine external interrupt buffer.

All of these addresses are in guest absolute addresses in the host-primary address space.

For the AUTHORIZE function, the interrupt buffer is checked during execution of DIAGNOSE code X'68' for storage-protection violations. All storage-protection mechanisms applicable to synchronous stores are enforced. Subsequently, when each interrupt is presented, the buffer is again checked for key-controlled protection according to the PSW key at the time of the AUTHORIZE. If this check fails, the interruption is discarded and, if it was not a final response or IDENTIFY interrupt, then a final response interrupt indicating a data transfer error (VMCMEFLG=19) is presented to the sender.

For functions other than AUTHORIZE, the PSW key at the time of the DIAGNOSE code X'68' instruction is used to enforce key-controlled protection when data is transferred. Low-address protection, fetch-protection override, and storage-protection override do not apply.

The length of the associated data or buffer is specified in the VMCPLENA field.

VMCPLENA
contains the length of the data sent by a user, the length of a receive buffer, or the length of an external interrupt buffer, whichever is specified in the field VMCPVADA. The size of the value specified in VMCPLENA is restricted only by virtual machine storage size.

The target virtual machine can use the value in this field as the data length for Receive operations.

VMCPVADB
contains the address of a source virtual machine's reply buffer for a SEND/RECV request. The address a guest absolute address in the host-primary address space. When the target virtual machine issues a REPLY in response to a SEND/RECV from the source virtual machine, the reply data is moved in this buffer. The length of the reply buffer is contained in the field VMCPLENB.

The PSW key at the time of the DIAGNOSE code X'68' instruction is used to enforce key-controlled protection when data is transferred. Low-address protection, fetch-protection override, and storage-protection override do not apply.

VMCPLENB
specifies the length of the source virtual machine's reply buffer. The target virtual machine uses this field to determine the maximum length of the reply. A corresponding field within the response message header contains a residual data count. The source virtual machine uses this residual count to determine the length of the target reply. The original reply buffer length (less the residual count) is the length of the reply from the target virtual machine.

VMCPUSE
contains the VMCF user doubleword. The user doubleword is transmitted to the target virtual machine in the send message header for SEND, SEND/RECV, SENDX, and IDENTIFY requests. For RECEIVE, REPLY, and REJECT requests, the user doubleword is transmitted to the source virtual machine within the response message header. The target virtual machine can transmit the user doubleword to the source virtual machine with REJECT or REPLY requests only if the original request was a SEND/RECV. The user doubleword is transmitted only with requests that result in send or response external interrupts.

See Table 210 on page 1003 for a summary of the VMCPARM fields required for execution of each of the VMCF functions. Possible return codes (in decimal) associated with each function are also listed. For a list of the return codes, hexadecimal values, and their meanings, see "Responses" on page 988.

Table 210. Required VMCPARM Fields for VMCF Functions

VMCF Function	Applicable VMCPARM Parameters	Return Codes
AUTHORIZE	VMCPFLG1 – Specific/Priority option VMCPFUNC – X'0000'–subcode VMCPUSER – Specific user ID VMCPVADA – External interrupt buffer address VMCPLENA – External interrupt buffer length	0, 1, 6, 15
UNAUTHORIZE	VMCPFUNC – X'0001'–subcode	0, 4
SEND	VMCPFLG1 – Priority option VMCPFUNC – X'0002'–subcode VMCPMID – Message identifier VMCPUSER – Target user ID VMCPVADA – Send data address VMCPLENA – Send data length VMCPUSE – User doubleword (See Note)	0, 1, 4, 5, 8, 9, 10, 18
SEND/RECV	VMCPFLG1 – Priority option VMCPFUNC – X'0003'–subcode VMCPMID – Message identifier VMCPUSER – Target user ID VMCPVADA – Send data address VMCPLENA – Send data length VMCPVADB – Reply buffer address VMCPLENB – Reply buffer length VMCPUSE – User doubleword	0, 1, 4, 5, 8, 9, 10, 18
SENDX	VMCPFLG1 – Priority option VMCPFUNC – X'0004'–subcode VMCPMID – Message identifier VMCPUSER – Target user ID VMCPVADA – Send data address VMCPLENA – Send data length VMCPUSE – User doubleword (See Note)	0, 1, 4, 5, 7, 8, 9, 10, 18
RECEIVE	VMCPFUNC – X'0005'–subcode VMCPMID – Message identifier VMCPUSER – Source user ID VMCPVADA – Receive buffer address VMCPLENA – Receive buffer length VMCPUSE – User doubleword	0, 1, 3, 4, 5, 6, 12 13, 15, 16, 17, 19
CANCEL	VMCPFUNC – X'0006'–subcode VMCPMID – Message identifier VMCPUSER – Target user ID	0, 3, 4, 5, 11, 12, 14, 20

Table 210. Required VMCPARM Fields for VMCF Functions (continued)		
VMCF Function	**Applicable VMCPARM Parameters**	**Return Codes**
REPLY	VMCPFUNC – X'0007'–subcode VMCPMID – Message identifier VMCPUSER – Source user ID VMCPVADA – Reply data address VMCPLENA – Reply data length VMCPUSE – User doubleword	0, 1, 3, 4, 5, 6, 12, 13, 15, 16, 17, 19
QUIESCE	VMCPFUNC – X'0008'–subcode	0, 4
RESUME	VMCPFUNC – X'0009'–subcode	0, 4
IDENTIFY	VMCPFLG1 – Priority option VMCPFUNC – X'000A'–subcode VMCPUSER – Target user ID VMCPUSE – User doubleword (See Note)	0, 4, 5, 9, 10, 18
REJECT	VMCPFUNC – X'000B'–subcode VMCPMID – Message identifier VMCPUSER – Source user ID VMCPUSE – User doubleword	0, 3, 4, 5, 12, 13
SETLIMIT	VMCPFUNC – X'000C'–subcode VMCPMID – Message limit	4, 10
Note: Fields within the user parameter list that are not used by a particular function may be used to contain additional user data. The data, however, can only be passed to the target virtual machine by the source virtual machine. The reply buffer address and length fields (VMCPVADB + VMCPLENB) may be used to transmit additional user data for SEND and SENDX requests. All fields except VMCPFLG1, VMCPFUNC, and VMCPUSER may be used to pass control information with an IDENTIFY request.		

External Interrupt Code X'4001'

External interruption code X'4001' is a special interrupt code recognized by CP as part of a VMCF transaction. Just as virtual machines use the DIAGNOSE instruction to communicate with CP, so too CP uses this interrupt code to communicate with virtual machines. External interrupt code X'4001' and DIAGNOSE code X'68' provide the mechanism VMCF uses to synchronize message processing.

The External Interrupt Message Header (VMCMHDR)

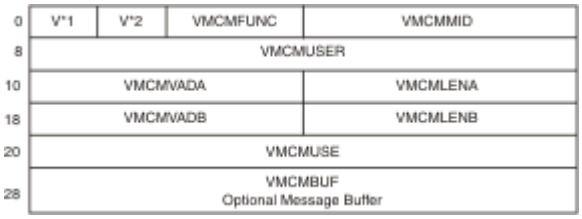
Associated with external interruption code X'4001' is a storage area referred to as the external interruption message header. The external interrupt message header (VMCMHDR) contains the control information required to send and receive messages. The fields within the message header are, for the most part, a copy of VMCPARM parameter list fields.

CP passes the external interruption buffer (containing the external interruption message header) to the user's interruption handler for processing. The user must specify the address and length of this buffer when he executes the AUTHORIZE function. This address must be in second-level storage (the storage that appears real to the virtual machine). Then, when the user sends or receives messages, CP knows the address of the buffer and passes it to the appropriate interruption handler routine.

Fields VMCMFUNC through VMCMUSE correspond to the fields VMCPFUNC through VMCPUSE in the VMCMHDR DSECT parameter list transmitted by the source virtual machine. The VMCMHDR COPY file is provided in the HCPGPI macro library.

The format of the message header and optional SENDX data buffer is:

VMCMHDR DSECT



V*1 (VMCMSTAT)
is a status byte associated with the message header. The bits within the status byte are defined as follows:

VMCMRESP (X'80')
indicates final external interrupt (transaction complete).

VMCMRJCT (X'40')
is set in a response external interrupt to indicate that the target virtual machine rejected the message by means of the REJECT function.

VMCMPRTY (X'20')
is set in both send and response external interrupts to indicate a priority message.

V*2 (VMCMEFLG)
contains a data transfer error code indicating success or errors associated with a data transfer operation. This is only valid for Final Response Interrupts.

VMCMFUNC
contains the subcode of the original request. The target virtual machine uses this field to determine the type of request. The possible subcodes are shown in Table 211 on page 1005.

Table 211. VMCMFUNC Subcodes - DIAGNOSE Code X'68'

Field Attributes in the DSECT	Hexadecimal Code	Function
VMCPSEND	X'0002'	SEND
VMCPSENR	X'0003'	SEND/RECV
VMCPSENX	X'0004'	SENDX
VMCPIDEN	X'000A'	IDENTIFY

VMCMMID
contains the message ID associated with the original source request.

VMCMUSER
contains the user ID of the source virtual machine for send external interrupts and the user ID of the target virtual machine for response external interrupts.

VMCMVADA
contains the address of the original send data for SEND requests.

VMCMLENA
indicates the length of send data for send external interrupts. It indicates a data transfer residual count for response external interruptions.

VMCMVADB
contains the address of the reply buffer for SEND/RECV requests.

VMCMLENB
contains the length of the source virtual machine reply buffer for SEND/RECV external interrupts; contains the residual reply count for response external interrupts. The target virtual machine uses this field to determine the maximum length of the reply; the source virtual machine uses this field to determine the length of the target virtual machine reply data.

VMCMUSE
contains the user doubleword, which is transmitted to the target virtual machine with send external interrupts and to the source virtual machine with response external interrupts.

VMCMBUF
This is the optional data buffer used by the SENDX function. The data sent with the SENDX function is moved into this buffer. The buffer size is specified when a virtual machine executes the VMCF AUTHORIZE function.

VMCF User Doubleword

VMCF provides a doubleword for user data that can be transmitted within the external interrupt message header. A user supplies the doubleword of data within the parameter list (VMCPARM) for certain VMCF requests (SEND, SENDX, SEND/RECV, RECEIVE, REPLY, IDENTIFY, and REJECT). You can use the user doubleword in any manner you desire. The doubleword is transmitted within the external interrupt message header for both send and response external interrupts.

The user doubleword can be used for control information in a user-defined higher level protocol. That is, you could have your own message headers defined within the data transmitted from one virtual machine to another. The user doubleword could be used to control such a protocol.

The user doubleword can also be used as a security code or provide additional information for functions such as IDENTIFY and REJECT. You can specify a 0 data length for any VMCF transaction. The effect of this is that only the external interrupt message header with user doubleword is transmitted or received.

VMCF in an MP Environment

VMCF includes support for a virtual multiprocessor (MP) environment. If you use VMCF, but do not use virtual multiprocessing, you are not impacted by this support. The following list is intended to provide some guidance on using VMCF in a virtual MP environment:

- As long as one processor in a virtual configuration has authorized for VMCF, any virtual processor within the virtual configuration can invoke VMCF functions.
- If you run with multiple virtual processors defined and only one virtual processor has issued a VMCF AUTHORIZE, note the following:
  - The virtual processor that issued the AUTHORIZE will be presented with all VMCF external interrupts if enabled. The buffer length may be changed on subsequent AUTHORIZE commands issued by the one authorized virtual processor. Any virtual processor in the complex can invoke VMCF functions.
  - All other VMCF functions will remain the same.
- If you run with multiple virtual processors defined and two or more virtual processors have issued a VMCF AUTHORIZE, note the following:
  - The VMCF functions of QUIESCE and RESUME apply to the entire virtual processor complex.
  - Any processor in a virtual MP complex can issue a VMCF function as long as at least one processor in the virtual MP complex has issued a VMCF AUTHORIZE. Multiple virtual processors may AUTHORIZE.
  - The external interrupt buffer length (VMCPLENA) on the first VMCF AUTHORIZE in a virtual MP complex is the required length for all other interrupt buffers defined in that virtual MP complex by subsequent AUTHORIZEs. If a subsequent AUTHORIZE on another virtual processor is issued with an interrupt buffer length that does not match the length defined in the initial AUTHORIZE, a return code of 1 is received from the AUTHORIZE command. The buffer length for the virtual MP complex can be changed after the entire MP complex unauthorizes, or when one processor remains authorized and that processor issues another authorize with a new length (this corresponds to a non-MP environment).
  - Each processor that has issued a VMCF AUTHORIZE in the virtual MP complex has its own interrupt buffer.
  - In the virtual MP environment, VMCF interrupts are treated as "floating" external interrupts. Any virtual processor that issued an AUTHORIZE and has enabled for VMCF external interrupts may receive a VMCF interrupt.

  - The type of authorization is complex-wide for all processors that issue an AUTHORIZE, and is of the type of the last AUTHORIZE issued. For example, if the last AUTHORIZE issued is an AUTHORIZE SPECIFIC, then all processors in the virtual MP complex are authorized specific. AUTHORIZE PRIORITY and authorization for SMSG are treated the same way.
  - UNAUTHORIZE is a processor-specific function. The entire complex is not unauthorized until the last authorized processor issues an UNAUTHORIZE.
  - CMS can only handle VMCF interrupts on its base processor, that is, the IPLed processor. Therefore, in order to run a CMS VMCF application, the application must first issue VCPUSELECT VM_CPU_BASE_ONLY.
- Regardless of the number of virtual processors defined, the message limit established by the SETLIMIT function (see "SETLIMIT: DIAGNOSE Code X'68' Subcode X'000C'" on page 1000) applies to all processors collectively.

DIAGNOSE Code X'68' Return Codes

The virtual machine initiating a VMCF request receives a return code in the general purpose register specified as Ry in the DIAGNOSE instruction. The return code indicates successful completion of the request or error conditions associated with the request. Table 212 on page 1007 is a description of all possible return codes returned to a virtual machine executing DIAGNOSE code X'68'.

Table 212. DIAGNOSE Code X'68' Return Codes

Return Code	Meaning
0 (X'00')	The normal response. Indicates successful completion of a request or successful initiation of a request. For example, for an AUTHORIZE request, 0 indicates that the AUTHORIZE function is complete; for a SEND request, 0 indicates that the SEND was successfully initiated. The SEND request, of course, would not be complete until the final RESPONSE external interrupt was received by the source virtual machine.
1 (X'01')	Invalid virtual buffer address or length. A virtual machine tried to execute a VMCF function but specified an invalid address or length: • External interrupt buffer not within virtual storage. • External interrupt buffer address not doubleword aligned. • Message data or buffer not within virtual storage. • External interrupt buffer less than the standard message header length. • In a virtual MP complex, if this is not the first AUTHORIZE, this return code may indicate that your interrupt buffer length does not match the initial length on the first AUTHORIZE.
2 (X'02')	Invalid function code. A virtual machine tried to execute a VMCF function but specified an unsupported subcode.
3 (X'03')	Protocol violation. A virtual machine tried to execute a function which would violate the defined protocol: • Cancel a message it did not originate. • Reply to a message not sent via SEND/RECV. • Executed more than one RECEIVE to a SEND or SEND/RECV request.
4 (X'04')	Source virtual machine not authorized. A virtual machine tried to execute a function (other than AUTHORIZE) but was not authorized to use VMCF (had not successfully executed the AUTHORIZE function).

Table 212. DIAGNOSE Code X'68' Return Codes (continued)

Return Code	Meaning
5 (X'05')	User not available. A virtual machine tried to execute a function and specified a virtual machine currently not available for VMCF communication: • Not logged on. • Not authorized for VMCF communication. • Virtual machine authorized SPECIFIC for some other virtual machine.
6 (X'06')	Protection violation. A virtual machine tried to execute a VMCF function that would result in a STORE or FETCH protection violation. The virtual machine specified a data or buffer address that contained a storage key other than its current PSW key (assume the key was nonzero). This return code is also set if a virtual machine tries to receive data in a CP-owned shared segment.
7 (X'07')	SENDX data too large. A virtual machine tried to execute a SENDX request but specified a SENDX data length larger than the target virtual machine external interrupt buffer.
8 (X'08')	Duplicate message. A virtual machine tried to execute a SEND-type function and specified a message ID and virtual machine user ID for which there was already an active message.
9 (X'09')	Target virtual machine in QUIESCE status. A virtual machine tried to execute a SEND-type function and specified a target virtual machine user ID of a virtual machine in QUIESCE status.
10 (X'0A')	Message limit exceeded. A virtual machine tried to execute a SEND function but already had the maximum number of messages active as specified by the SETLIMIT function, or the partner virtual machine has as many active incoming messages as are allowed by the MAXVMCFI limit in his directory. The virtual machine should clear any pending RESPONSE external interrupts or CANCEL previously sent messages to continue processing.
11 (X'0B')	REPLY cancelled. The source virtual machine executed a CANCEL to a previous SEND/RECV request. The target virtual machine had already RECEIVED the message but had not yet executed a REPLY. The target virtual machine REPLY in this case is cancelled. The target virtual machine receives return code 12 - X'0C' - (message not found) when it executes the REPLY function.
12 (X'0C')	Message not found. A virtual machine tried to execute a function and specified a message ID and virtual machine user ID for a message that does not exist. The message may have existed at one time but could have been cancelled by the originator.
13 (X'0D')	Synchronization error. The target virtual machine tried to respond to a message for which it had not yet received the SEND external interrupt. This condition can occur if the target virtual machine is anticipating certain messages but does not wait for the SEND external interrupt.
14 (X'0E')	CANCEL too late. A virtual machine tried to CANCEL a message that had already been processed. The target virtual machine had already responded with RECEIVE or REJECT (SEND request) or REPLY or REJECT (SEND/RECV request). This return code is also set if a virtual machine tries to CANCEL a SENDX request for which the target virtual machine had already received the SEND external interrupt.
15 (X'0F')	Paging I/O error. A virtual machine tried to execute a function which resulted in an uncorrectable paging I/O error. This is a hardware failure.

Table 212. DIAGNOSE Code X'68' Return Codes (continued)

Return Code	Meaning
16 (X'10')	Incorrect length. A virtual machine executed a RECEIVE or REPLY function and specified a RECEIVE buffer length less than the source virtual machine SEND data length or a REPLY data length larger than the source virtual machine reply buffer length. The source virtual machine receives a data transfer return code identifying the condition.
17 (X'11')	Destructive overlap. A virtual machine executed a RECEIVE or REPLY function and specified a RECEIVE buffer address which overlapped the source virtual machine SEND data address or a REPLY data address that overlapped the source virtual machine reply buffer address. This condition can occur only when a virtual machine is sending messages to itself (a wrap connection).
18 (X'12')	User not authorized for PRIORITY messages. A virtual machine tried to send a PRIORITY message to a virtual machine that was not authorized to accept PRIORITY messages (that is, had not executed the AUTHORIZE function with the PRIORITY option).
19 (X'13')	Data transfer error. A virtual machine executed a request that resulted in a data transfer error condition associated with the other virtual machine. The return code is returned to the target virtual machine to indicate that the transaction did not complete successfully.
20 (X'14')	CANCEL - busy. A virtual machine tried to cancel a message being processed. If this is a SEND/RECV request and the RECEIVE function is in process, repeated retries may cancel the REPLY function.

Data Transfer Error Codes

When a virtual machine executes a SEND, SENDX, or SEND/RECV function, the normal DIAGNOSE return code is 0, indicating that the request was successfully initiated. However, when the actual data transfer takes place, errors can occur. All errors occurring at data transfer time are communicated to the source virtual machine in the Final Response external interrupt message header, VMCMHDR. Table 213 on page 1009 shows error codes indicating conditions that are possible after the SENDX, SEND, or SEND/RECV request is initiated. The error codes correspond to DIAGNOSE return code numbers.

Table 213. DIAGNOSE Code X'68' Data Transfer Error Codes

Error Code	Meaning
0 (X'00')	The normal response (no errors).
1 (X'01')	Invalid buffer address or length. The SEND and/or RECEIVE buffers used for a data transfer operation are not within the virtual machine's virtual storage. The beginning and ending addresses were valid when a request was initiated but all addresses are not valid.
5 (X'05')	User not available. The target virtual machine executed the UNAUTHORIZE function, executed the AUTHORIZE SPECIFIC function again, or implicitly reset his virtual machine after the source virtual machine request was initiated.
6 (X'06')	Protection violation. The storage key for a virtual machine's SEND or RECEIVE buffer did not match its PSW key at the time the transfer was initiated (assume the key was nonzero). This error code is also set if a virtual machine tries to RECEIVE data into a CP-owned shared segment.

Table 213. DIAGNOSE Code X'68' Data Transfer Error Codes (continued)

Error Code	Meaning
7 (X'07')	SENDX data is too large. The target virtual machine executed AUTHORIZE again and specified an external interrupt buffer size less than the buffer size at the time a SENDX function was executed. The SENDX data no longer fits in the target virtual machine buffer.
15 (X'0F')	Paging I/O error. An uncorrectable paging I/O error occurred during the data transfer operation trying to fetch a virtual machine SEND or RECEIVE buffer. This is a hardware failure.
16 (X'10')	Incorrect length. The target virtual machine executed a RECEIVE function with a data length (VMCPLENA) smaller than the original SEND data length or a REPLY function with a REPLY data length larger than the source virtual machine reply buffer length. The data is transferred for the smaller length.
17 (X'11')	Destructive overlap. A virtual machine was communicating with itself in a wrap connection and his SEND or RECEIVE buffers overlapped one another (intra-virtual machine communication).
19 (X'13')	Data transfer error. A data transfer error occurred which was associated with the other virtual machine. The transaction did not complete successfully.

Appendix D. The Special Message Facility

The Special Message Facility enables a virtual machine to send messages to another virtual machine by issuing the CP SMSG command. The Special Message Facility may be used with the Virtual Machine Communication Facility (VMCF) or with the Inter-User Communications Vehicle (IUCV). However, the sending virtual machine does not need to perform the initialization required by VMCF or IUCV. Initialization is handled by CP and is described later in this topic.

To send a message, a virtual machine need only prepare the message text and issue the class G SMSG command. Parameters on the SMSG command identify the user ID of the receiving virtual machine and specify the message text. The format of the message text must be acceptable to the receiving virtual machine. The SMSG command is described in the *z/VM: CP Commands and Utilities Reference*.

For VMCF: Before the receiving virtual machine can receive special messages through VMCF, it must:

- Enable itself to receive external interrupts.
- Set bit 31 of control register 0 to a value of 1.
- Authorize itself by issuing DIAGNOSE code X'68', AUTHORIZE. The parameter list, VMCPARM, specified with DIAGNOSE code X'68' must contain a pointer to an external-interrupt buffer, should specify a buffer length of 280 bytes, and must have the special message flag (VMCPSMSG) turned on.

  Note that you may receive a message, *Message too large*, if you issue the SMSG command from a 3279 or 3287 Model 5 terminal to send a message longer than what the receiving virtual machine has specified.
- Turn on this special message flag (VMCPSMSG) by setting VMCPSMSG to a value of B‘1’ or by issuing the class G command, SET SMSG ON. For information on using DIAGNOSE code X'68', see “DIAGNOSE Code X'68'” on page 987, and “The Virtual Machine Communication Facility” on page 989.

To understand how a special message is presented to the receiving virtual machine through VMCF, see “The SENDX Protocol” on page 994.

For IUCV: Before the receiving virtual machine can receive special messages through IUCV, it must do the following:

- Enable itself to receive external interrupts
- Set bit 30 of control register 0 to a value of 1
- Issue the IUCV DECLARE BUFFER function
- Issue the IUCV CONNECT function to the CP Message System Service
- Turn on the special message flag by issuing the class G command SET SMSG IUCV.

When VMCF or IUCV message are no longer required, reset the corresponding bit in the contents of control register 0. The IUCV mask is set and cleared by CMS support for IUCV and APPC/VM, and should be used to receive special messages through IUCV.

When a virtual machine no longer wishes to accept special messages, it may turn off the special message flag by issuing the command SET SMSG OFF. To resume receiving messages, the virtual machine may issue the command SET SMSG ON or SET SMSG IUCV. CP sends an error message to any virtual machine that tries to send a special message to another virtual machine that is not accepting special messages.

Special messages are queued only as long as the virtual machine is logged on. If the virtual machine sets SMSG off or logs off, this queue of SMSGs is lost. A system IPL also loses this queue of messages for the virtual machine.

CP handles VMCF/IUCV initialization and special message processing as follows. When the SMSG command is issued, CP verifies that no invalid options were specified and that a valid user ID was specified. CP also verifies that the receiving virtual machine is accepting special messages. CP then obtains storage for the message, builds the appropriate parameter list, and sends the message to the receiving virtual machine.

© Copyright IBM Corp. 1991, 2020

Appendix E. Logical Device Support Facility

The Logical Device Support Facility allows an application running in a virtual machine to create within CP one or more logical devices. 3270 extended data streams are supported to enable logical devices to utilize full color, programmed symbol sets, and extended highlighting capabilities. 3284, 6, 7, 8, 9 logical printer devices are supported to allow the presentation of status from a logical device printer. Applications are allowed to create logical 328x printers in addition to logical 327x display devices. Except for the logical device support facility, CP is unaware of the fact that this device has no real existence and is driven by the application program. In particular, CP sees it as a local 3270 device. Any output directed to a logical device is redirected to the virtual machine for which the device was created. The virtual machine can also transfer data to CP to be entered as input from a specific logical device, as if it were interactively produced on a real terminal.

The logical device support facility is made up of two data transfer functions, four control functions, a special external interrupt (code X'2402'), and an external control word for passing control information with the external interrupt.

To implement this facility, functions are invoked using DIAGNOSE code X'7C'. Registers Rx, Rx+1, Ry, and Ry+1 indicate the function, logical device identification, and other function-dependent information.

A special interrupt code (X'2402') notifies a virtual machine of pending logical device status for a logical device created for that virtual machine. Along with this interrupt, the virtual machine receives a control word at a virtual storage location indicating the ID of the associated logical device and the reason for the interrupt.

See Table 214 on page 1013 for a summary of logical device support facility functions. More complete information about each of these functions is included under “Logical Device Support Facility Functions” on page 89.

Data is directed to a logical device using the logical device ID. This ID is assigned by CP during execution of the INITIATE function. Data transfer takes place within CP at a channel command level. I/O directed to a logical device proceeds within CP through the normal path for a local device up to the point where the I/O is to be started. At that point, control passes to the logical device support modules to process the CCW string. Channel commands requiring interaction cause external interrupts to the virtual machine for which the associated logical device was created.

The format of data from the virtual machine must conform to 3270 architecture for local devices.

The default maximum number of logical devices for the system is 4096. The maximum number of logical devices allowed may be changed with the CP command SET MAXLDEV. There is no limit to the number of hosts as long as the number of logical devices does not exceed the system limit.

Table 214. Summary of Logical Device Support Facility Functions

Function	Description
INITIATE	Initiate logical device communications
ACCEPT	Transfer data written to logical device to virtual machine storage.
PRESENT	Transfer data from virtual machine to CP as input from logical device.
TERMINATE	Drop a specific logical device.
TERMINATE ALL	Drop all logical devices created for this virtual machine.
STATUS	Allows status to be returned to CP after an ACCEPT function is performed.

The VM/Pass-Through Facility licensed program is an example of an application using the logical device support facility. Through the combined support of these two facilities, a z/VM user attached to system A

© Copyright IBM Corp. 1991, 2020

through a 3270 Display Station can access z/VM system B as though the display station were locally attached to system B.

Appendix F. Reserved DIAGNOSE Codes

Attention:

This appendix contains information that is NOT Programming Interface information. These DIAGNOSE codes are reserved for IBM use.

DIAGNOSE Code X'40' – Clean-Up After Virtual IPL by Device

Privilege class: Any

Addressing Mode: 24-bit or 31-bit

This code is valid only during virtual IPL, and is for system use only. Other use may cause unpredictable results.

Entry Values: The Rx and Ry fields are used together to specify flag fields, and do not specify general registers. A value of X'00' in the RxRy field requests the IPL clean-up function. A value of X'FF' in the RxRy field requests relocation of the IPL simulator to another page in storage.

RxRy field=X'00'
When the simulation of the IPL function is complete, DIAGNOSE code X'40' with RxRy code X'00' is issued to cause CP to clean up the page in the virtual machine's storage that was used to contain the IPL simulator.

RxRy field=X'FF'
HCPVMI resides in a page in virtual machine storage and simulates the virtual IPL function. If a CCW is going to write into the page occupied by HCPVMI, DIAGNOSE X'40' with RxRy code X'FF' is issued to move the IPL simulator to another page in storage. CP relocates the virtual machine's registers and the PSW to the new page.

Usage Note

If DIAGNOSE X'40' is issued with an RxRy field having a value other than X'00' or X'FF', a specification exception is reflected to the virtual machine.

Responses

None.

DIAGNOSE Code X'E0' – System Trace File Interface

Privilege Class: Any

DIAGNOSE X'E0' provides a virtual machine access to system trace files. The WRITE function of DIAGNOSE X'E0' is a programming interface for customers. For more information see “DIAGNOSE Code X'E0' – System Trace File Interface” on page 171. The remaining functions allow the owner of a system trace file to read trace blocks sequentially from the file. These functions are not programming interfaces for customers. The TRACERED command of the Dump Viewing Facility is the preferred interface for reading system trace files.

To use the open, read, and close subcodes of this DIAGNOSE code, the virtual machine must be authorized using the TRSAVE command. The virtual machine can process (open, read, or close) system trace files if a TRSAVE command was issued to specify its user ID as the receiver. It can also process these trace files if it is the default receiver for a file that is completed. In this case, it is the owner of the file.

© Copyright IBM Corp. 1991, 2020

The virtual machine can use this facility to read system trace files created by the following in various combinations:

- The Monitor Call Class 10 interface
- The DIAGNOSE Code X'E0' write interface
- The TRSOURCE command
- The TRSAVE command.

The TRSOURCE command supports the definitions and control of I/O, data, and guest tracing, while TRSAVE specifies where the TRSOURCE and CPTRACE data is to be stored. The Monitor Call interface is only one of the ways data can be collected. Refer to the description in this manual for further details on the Monitor Call Class 10 interface.

Event trace records are formatted by CP into blocks that have the header format described below.

Entry values:

Rx
Contains the guest real address of a 4-character spool ID (SPID) of a system trace file that it owns.

Note: For the READ function, Rx cannot be specified as register 15.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the spool ID. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the spool ID is in the host-primary address space.

Ry
Is the DIAGNOSE subcode that specifies the function requested.

Value
Operation performed

X'00000008'
Open (read only)

X'0000000C'
Read

X'00000010'
Close (read only)

Note: Do not specify Ry as register 15 for the close operation.

Rx+1
For the READ function, contains the address of the buffer into which to read the next block of data. The first two bytes of this buffer must contain the length of the buffer. The length must be a multiple of 4KB.

Ax+1
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the buffer to receive the data. If Rx+1 designates general register 0 (when Rx is general register 15), if Ax+1 contains X'00000000', or if the virtual machine is not in XC mode, the buffer is in the host-primary address space.

Exit values:

On return from the DIAGNOSE processor, a return code is set in the Ry+1 register. The return codes are right-justified in the register and padded with zeros. Only the rightmost byte value is given below.

Unsupported subcode:

Return Code	Meaning
48 (X'30')	Invalid subcode

Program Exceptions:

Problem Encountered	Cause
Specification exception	Ry is register 15

Subcode X'00000008' — Open (read-only)

The system trace file indicated by Rx is opened if the virtual machine is the owner of this file. The virtual machine is the owner of a system trace file if its ID was designated as the trace receiver ('TO userid') on the TRSAVE command used to start the trace.

Entry values:

Ry+1
On return, contains the return code from this DIAGNOSE open operation.

Exit values:

On return from the DIAGNOSE processor, a return code is set in the Ry+1 register. The return codes are right-justified in the register and padded with zeros. Only the rightmost byte value is given below.

Return Code	Meaning
0 (X'00')	Successful open of trace data defined by TRSOURCE
4 (X'04')	Successful open of CP system trace data
8 (X'08')	Buffer too small for block
12 (X'0C')	I/O error
16 (X'10')	Invalid spool ID
20 (X'14')	File not found
24 (X'18')	File in use
36 (X'24')	Protection exception condition
44 (X'2C')	Invalid spool ID address
52 (X'34')	Rx, Ry, and Ry+1 registers overlap
56 (X'38')	Severe error
60 (X'3C')	ALET-specification exception condition: For an XC virtual machine in access-register mode, Ax or Ax+1 contains an ALET that has an unexpected bit setting. See "Access Exceptions" on page 8 for more information.
64 (X'40')	ALEN-translation exception condition: For an XC virtual machine in access-register mode, Ax or Ax+1 contains an ALET that cannot be translated. See "Access Exceptions" on page 8 for more information.
68 (X'44')	Addressing-capability exception condition: For an XC virtual machine in access-register mode, Ax or Ax+1 contains an ALET that designates an address space for which your virtual machine's access has been revoked. (See usage note "5" on page 1019and "Access Exceptions" on page 8 for more information.)

Subcode X'0000000C' — Read

The next block of trace data is read from the file into the buffer provided by the virtual machine. When an end-of-file is reached, the files are not closed. To close them, the virtual machine must issue a DIAGNOSE code X'E0' CLOSE. If the buffer size specified is insufficient to read the next block of data, an error code is returned to the virtual machine and the file is positioned to reread the same block. The Blength field in the truncated record can be used to determine the required buffer size except for blocks containing CP trace table entries which are always 4096 bytes long.

Exit values:

On return from the DIAGNOSE processor, a return code is set in the Ry+1 register. The return codes are right-justified in the register and padded with zeros. Only the rightmost byte value is given below.

Return Code	Meaning
0 (X'00')	Successful read
4 (X'04')	End-of-file
8 (X'08')	Buffer too small for block
12 (X'0C')	I/O error
16 (X'10')	Invalid spool ID
20 (X'14')	No file open
28 (X'1C')	Invalid buffer address
32 (X'20')	Invalid buffer length
36 (X'24')	Protection exception condition
44 (X'2C')	Invalid spool ID address
52 (X'34')	Rx, Ry, and Ry+1 registers overlap or Rx is register 15
56 (X'38')	Severe error
60 (X'3C')	ALET-specification exception condition: For an XC virtual machine in access-register mode, Ax or Ax+1 contains an ALET that has an unexpected bit setting. See “Access Exceptions” on page 8 for more information.
64 (X'40')	ALEN-translation exception condition: For an XC virtual machine in access-register mode, Ax or Ax+1 contains an ALET that cannot be translated. See “Access Exceptions” on page 8 for more information.
68 (X'44')	Addressing-capability exception condition: For an XC virtual machine in access-register mode, Ax or Ax+1 contains an ALET that designates an address space for which your virtual machine's access has been revoked. (See usage note “5” on page 1019 and “Access Exceptions” on page 8 for more information.)

Subcode X'00000010' — Close (read-only)

When reading a file, close is used to close an open file. Note that when an end-of-file is reached on a read operation, the file is not closed. To close the file, issue DIAGNOSE code X'E0' CLOSE.

Exit values:

On return from the DIAGNOSE processor, a return code is set in the Ry+1 register. The return codes are right-justified in the register and padded with zeros. Only the rightmost byte value is given below.

Return Code	Meaning
0 (X'00')	Successful close
12 (X'0C')	I/O error
16 (X'10')	Invalid spool ID
20 (X'14')	File not opened
36 (X'24')	Protection exception condition
44 (X'2C')	Invalid spool ID address

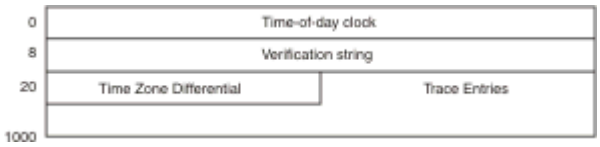
Return Code	Meaning
52 (X'34')	Rx, Ry, and Ry+1 registers overlap
60 (X'3C')	ALET-specification exception condition: For an XC virtual machine in access-register mode, Ax contains an ALET that has an unexpected bit setting. See “Access Exceptions” on page 8 for more information.
64 (X'40')	ALEN-translation exception condition: For an XC virtual machine in access-register mode, Ax contains an ALET that cannot be translated. See “Access Exceptions” on page 8 for more information.
68 (X'44')	Addressing-capability exception condition: For an XC virtual machine in access-register mode, Ax contains an ALET that designates an address space for which your virtual machine's access has been revoked. (See usage note “5” on page 1019 and “Access Exceptions” on page 8 for more information.)

Usage Notes

1. In order for CP trace entry data to be independent of the TRSOURCE trace entry data, the DIAGNOSE OPEN differentiates between the two types and sets a return code to identify which type of data the file contains. If RC=X'00', a TRSOURCE trace file was opened; if RC= X'04', a CP system trace file was opened.
2. Multiple files may be opened simultaneously for reading by a user. Only one file is open for write for an IO, DATA, or individual trace ID. There is one system trace file open for each guest virtual machine in the enabled trace ID.
3. If an I/O error occurs on a DIAGNOSE read request, CP does not attempt to close the file, and RC=X'0C' is returned to the caller.
4. An RC=X'38' (severe error) probably is caused by a programming error. Contact your IBM service representative.
5. An addressing-capability exception condition (RC=X'44') can occur after reading has begun.

Trace Block Containing CP Trace Table Entries

The following diagram describes the format of a block of trace data collected from CP trace tables as a result of a TRSAVE FOR CP ON command.



Time-of-day clock
Is the 8-byte time-of-day clock.

Verification string
Is the 24-character string *CP TRACE SERVICE TOOLS__*.

Time Zone Differential
Is the fullword time-of-day clock time zone differential.

Trace Entries
Are trace table entries.

DIAGNOSE Code X'214' – Pending Page Release

Privilege class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE X'214' is used by CMS storage manager to establish and cancel pending page releases. Prior to VM/ESA Release 1.1, DIAGNOSE X'10' was used for this purpose. Unlike DIAGNOSE X'10', DIAGNOSE X'214' allows CP to delay or omit the reclamation of host resources. DIAGNOSE X'10' is intended for general use while DIAGNOSE X'214' is not.

Attention: DIAGNOSE X'214' is not intended for use in access-register mode in an XC virtual machine. To do so yields unpredictable results.

DIAGNOSE X'214' will fail if executed when VMRELOCATION is in progress.

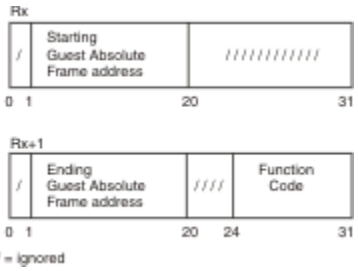
Entry Values:

Rx, Rx+1
Represents an even-odd register pair. The third byte of the odd register always contains a function code. For certain function codes the Rx and Rx+1 registers identify a range of pages. In these cases, bits 1-19 of the even register (Rx) designates the first page in the range and bits 1-19 of the odd register (Rx+1) designates the last page in the range. These page addresses are guest absolute addresses in the host-primary space.

Ry
For function code X'01' and X'03' (see below), Ry may designate a register that contains a guest key value in bits 24-28. When Ry designates register 0 the operand is ignored as are the contents of GPR 0.

Function Code	Meaning
X'00'	Establish Pending Release (EPR). EPR is used to establish pending releases for the range of pages designated by bits 1-19 of Rx and bits 1-19 of Rx+1. It permits CP to reclaim any resources backing the pages whenever CP may see fit as long as the page release remains pending.
X'01'	Cancel Pending Release (CPR). CPR is used to cancel any pending releases that may be in effect for the pages within the range specified in bits bits 1-19 of Rx and bits 1-19 of Rx+1. Optionally, if Ry designates a GPR other than GPR0, CP will set the guest keys for the specified range. It will use bits 24-27 of the contents of Ry to set the access key and bit 28 to set the fetch protect bit. The state of the reference and change bits is unpredictable.
X'02'	Cancel All Pending Releases (CAPR). CAPR is used to cancel all pending releases for this guest. For a CAPR, only the function code is examined. The contents of Rx, Ry, and bits 0-23 of Rx+1 are ignored.
X'03'	Cancel Pending Release and Validate (CPRV). CPRV is identical to CPR except that CP is notified of the CMS storage manager's intention to immediately reference the final page of the specified range. CP therefore will respond to CPRV by validating that page (if not currently valid) before returning to the guest.
Other	Reserved for future IBM use. The guest receives a specification exception if the function code is *other than* X'00', X'01', X'02', or X'03'.

Rx, Rx+1 contents for CPR(V) and EPR:



Ry contents (if Ry does not designate GPR 0) for CPR(V):

The condition code is set to 1 if key action was requested by a CPR(V) (i.e. Ry was not specified as 0) and the keys were not all successfully set. Otherwise, the condition code is set to zero.

Responses

Program Exceptions: You may receive one of the following program checks if the DIAGNOSE X'214' input data is invalid:

Problem Encountered	Cause
Specification exception	Any of the following: • Rx is not even register • In an MP virtual machine, the issuer was not running on the base processor • Function code is invalid • Function code is not CAPR (X'02') and address in Rx (starting address) is greater than address in Rx+1 (ending address) • Function code is not CAPR (X'02') and address in Rx or Rx+1 is not in range of guest's real storage.

DIAGNOSE Code X'23C' – Address Space Services

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE X'23C' is invoked from the guest ADRSPACE macro to perform address space management functions. For more information on the functions of the ADRSPACE macro, including a description of the possible return codes and program exceptions, refer to “ADRSPACE — Address Space Services” on page 797.

Notes:

1. This DIAGNOSE code is reserved for IBM use; it is not a supported programming interface. It is included here to be used only in the diagnosis task. The supported method of invocation for the address-space services is the ADRSPACE macro.
2. You may not be authorized to issue function X'03' of this DIAGNOSE code if an external security manager is installed on your system. For additional information, contact your security administrator.

The following address-space service functions can be invoked using this DIAGNOSE code:

- Create space
- Destroy space
- Query space
- Permit access
- Isolate space.

Entry Values:

Rx
The real address of a function parameter list, the format of which is determined by the function code in the parameter list. The parameter list is built by the ADRSPACE macro expansion in the macro's work area.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

Exit Values:

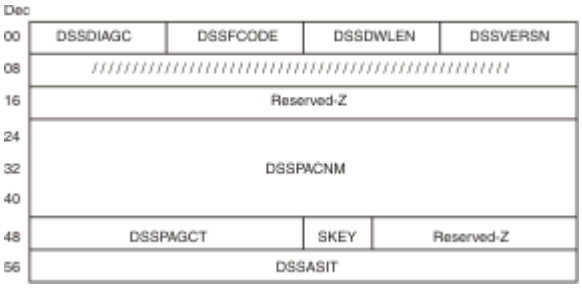
Ry
On return, contains a return code, as stated in under the address-space service functions of the ADRSPACE macro starting at “ADRSPACE — Address Space Services” on page 797.

Create-Space Function

This function creates a new address space and returns the ASIT associated with the new address space.

Your virtual machine must be an ESA/XC virtual machine to use this function.

For the create-space function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



DSSDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal DIAGNOSE code, X'023C'.

DSSFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the DIAGNOSE. A function code of X'0000' indicates the create-space function.

DSSDWLEN
Bytes 4 and 5 of the parameter list contain the length of the parameter list in doublewords.

DSSVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

Reserved-Z
Bytes 16 through 23, and bytes 53 through 55 of the parameter list are reserved and contain binary zeros.

DSSPACNM
Bytes 24 through 47 of the parameter list contain the address-space name to be assigned to the new address space.

DSSPAGCT
Bytes 48 through 51 of the parameter list contain a fullword that is the size of the new address space in pages; each page is 4096 bytes.

DSSSKEY
Bits 0 through 4 of byte 52 of the parameter list contain the access-control and fetch-protection bits of the storage keys for the new address space. Bits 0 through 3 designate the access-control bits, and bit 4 is the fetch-protection bit.

Bits 5 through 7 of byte 52 of the parameter list are reserved and contain binary zeros.

DSSASIT
Bytes 56 through 63 of the parameter list are set by CP to be the ASIT associated with the newly-created address space.

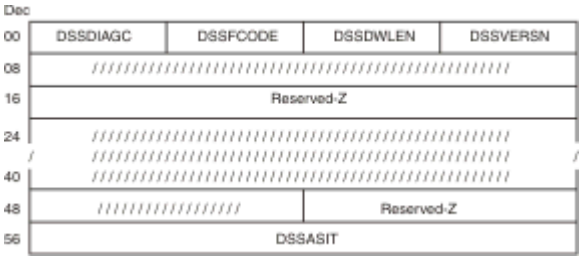
Bytes 8 through 15 of the parameter list are ignored.

Destroy-Space Function

This function destroys an address space previously created by your virtual machine.

Your virtual machine must be an ESA/XC virtual machine to use this function.

For the destroy-space function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



DSSDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal DIAGNOSE code, X'023C'.

DSSFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the DIAGNOSE. A function code of X'0001' indicates the destroy-space function.

DSSDWLEN
Bytes 4 and 5 of the parameter list contain the length of the parameter list in doublewords.

DSSVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

Reserved-Z
Bytes 16 through 23 and bytes 52 through 55 of the parameter list are reserved and contain binary zeros.

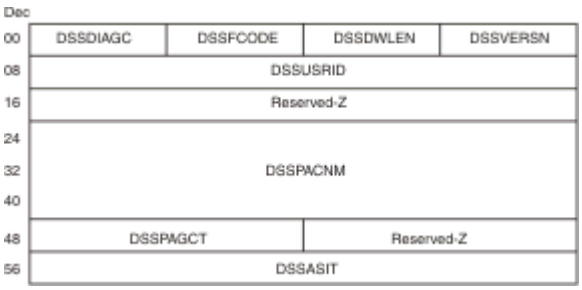
DSSASIT
Bytes 56 through 63 of the parameter list contain the ASIT associated with the address space to be deleted.

Bytes 8 through 15 and 24 through 51 of the parameter list are ignored.

Query-Space Function

This function returns the ASIT and size associated with an address space your virtual machine owns or is authorized to access.

For the query-space function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



DSSDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal DIAGNOSE code, X'023C'.

DSSFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the DIAGNOSE. A function code of X'0002' indicates the query-space function.

DSSDWLEN
Bytes 4 and 5 of the parameter list contain the length of the parameter list in doublewords.

DSSVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

DSSUSRID
Bytes 8 through 15 of the parameter list contain the user ID of the virtual machine owning the address space for which data is requested. The user ID can be from one to eight characters long; if it is less than eight characters long, it must be left-justified in the field, and the remainder of the field padded with spaces. If this field contains binary zeros, your virtual machine is the owner of the address space.

Reserved-Z
Bytes 16 through 23 and bytes 52 through 55 of the parameter list are reserved and contain binary zeros.

DSSPACNM
Bytes 24 through 47 of the parameter list contain the name of the address space for which information is to be returned.

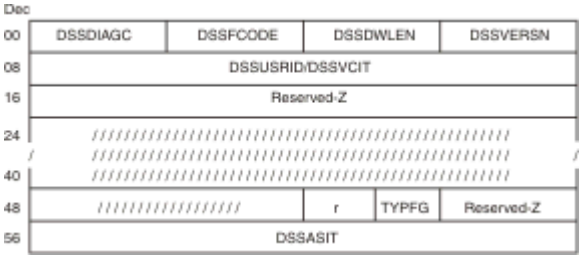
DSSPAGCT
Bytes 48 through 51 of the parameter list are set by CP to be a fullword that is the size of the named address space in pages; each page is 4096 bytes.

DSSASIT
Bytes 56 through 63 of the parameter list are set by CP to be the ASIT associated with the named address space.

Permit-Access Function

This function authorizes a virtual machine to use the add-ALE function of DIAGNOSE X'240' to add to its host access list an ALE designating an address space owned by your virtual machine.

For the permit-access function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



DSSDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal DIAGNOSE code, X'023C'.

DSSFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the DIAGNOSE. A function code of X'0003' indicates the permit-access function.

DSSDWLEN
Bytes 4 and 5 of the parameter list contain the length of the parameter list in doublewords.

DSSVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

DSSUSRID/DSSVCIT
Bytes 8 through 15 of the parameter list contain either the user ID or the VCIT of the virtual machine that is to be given access authorization. When this field contains a user ID, the user ID can be from one to eight characters long; if it is less than eight characters long, it must be left-justified in the field, and the remainder of the field padded with spaces. Whether this field contains a user ID or a VCIT is indicated by bit 4 of byte 53 (DSSTYPFG) as described below.

Reserved-Z
Bytes 16 through 23, byte 52, bits 1 through 7 of byte 53, and bytes 54 and 55 of the parameter list are reserved and contain binary zeros.

DSSTYPFG
Bit 0 of byte 53 of the parameter list controls whether the virtual machine specified in the DSSUSRID field is to be given read-only or read-write access authority. If this bit is zero, the virtual machine is given authority for read-only access to the address space. If this bit is 1, the virtual machine is given authority for read-write access to the address space.

Bit 4 of byte 53 of the parameter list controls whether the field at bytes 8 through 15 of the parameter list is interpreted as containing the user ID or the VCIT of the virtual machine to be permitted. If this bit is zero, bytes 8 through 15 of the parameter list are treated as the user ID of the virtual machine to be permitted. If this bit is one, bytes 8 through 15 of the parameter list are treated as the VCIT of the virtual machine to be permitted.

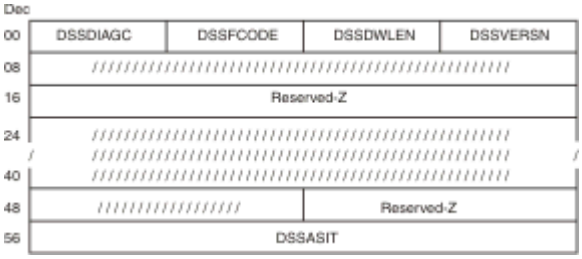
DSSASIT
Bytes 56 through 63 of the parameter list contain the ASIT associated with the address space for which access authorization is to be given.

Bytes 24 through 51 of the parameter list are ignored.

Isolate-Space Function

This function restores to a private state an address space owned by your virtual machine.

For the isolate-space function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



DSSDIAGC
Bytes 0 and 1 of the parameter list contain the hexadecimal DIAGNOSE code, X'023C'.

DSSFCODE
Bytes 2 and 3 of the parameter list contain the hexadecimal function code for the DIAGNOSE. A function code of X'0005' indicates the isolate-space function.

DSSDWLEN
Bytes 4 and 5 of the parameter list contain the length of the parameter list in doublewords.

DSSVERSN
Bytes 6 and 7 of the parameter list contain a version code of X'0001'.

Reserved-Z
Bytes 16 through 23 and bytes 52 through 55 of the parameter list are reserved and contain binary zeros.

DSSASIT
Bytes 56 through 63 of the parameter list contain the ASIT associated with the address space to be isolated.

Bytes 8 through 5 and bytes 24 through 51 of the parameter list are ignored.

Responses

Program Exceptions: The program exceptions for DIAGNOSE X'23C' are the same as those documented for the “ADRSPACE — Address Space Services” on page 797.

DIAGNOSE Code X'240' – Access List Services

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE X'240' is invoked from the guest ALSERV macro to perform access-list management functions.

Notes:

1. This DIAGNOSE code is reserved for IBM use; it is not a supported programming interface. It is included here to be used only in the diagnosis task. The supported method of invocation of the access-list services is the guest ALSERV macro. For more information on the ALSERV macro, refer to “ALSERV — Access List Services” on page 815.
2. This DIAGNOSE code does not support HyperPAV alias devices.

The following access-list service functions can be invoked using this DIAGNOSE code:

- Add-ALE
- Remove-ALE.

Entry Values:

Rx
Real address of a function parameter list, the format of which is determined by the function code in the parameter list. The parameter list is built by the ALSERV macro expansion in the macro's work area.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

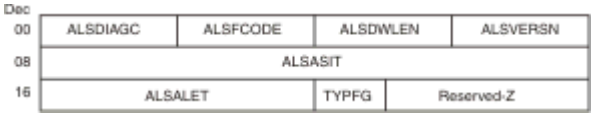
Exit Values:

Ry
On return, the Ry register contains a return code, as stated in the access-list services described under the “ALSERV — Access List Services” on page 815.

Add-ALE Function

This function establishes a valid ALE in your virtual machine's host access list.

For the add-ALE function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



Where:

ALSDIAGC
Bytes 0-1 of the parameter list contain the hexadecimal diagnose code, X'0240'.

ALSFCODE
Bytes 2-3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0000' indicates the add-ale function.

ALSDWLEN
Bytes 4-5 of the parameter list contain the length of this parameter list in doublewords.

ALSVERSN
Bytes 6-7 of the parameter list contain a version number ofX'0001'.

ALSASIT
Bytes 8-15 of the parameter list contain the ASIT associated with the address space to be designated by the new ALE.

ALSALET
Bytes 16-19 of the parameter list are set by CP to be the ALET that your virtual machine can use to reference the address space specified by the ALSASIT field.

ALSTYPFG
Bit 0 of byte 20 of the parameter list controls whether the new ALE allows read-only or read-write access to the designated address space. If this bit is zero, then the ALE is set to provide read-only access to the address space. If this bit is one, then the ALE is set to provide read-write access to the address space.

If the ALSASIT field specifies an address space owned by another virtual machine, then that virtual machine must have authorized your virtual machine for the requested type of access.

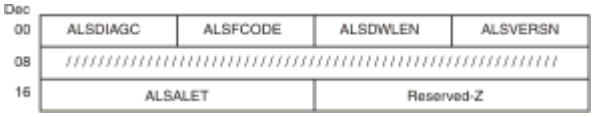
Bit 1 of byte 20 of the parameter list controls whether page-fault handshaking is to be activated. If this bit is zero, then page-fault handshaking should not be activated. If this bit is one then, whenever the address of a page-fault handshaking token is established for the virtual CPU, page-fault handshaking should be activated to notify both initiation of a page fault and completion of a page fault.

The address space is eligible for page-fault handshaking while a page-fault handshaking token is established for the virtual CPU until the address space is either explicitly or implicitly removed.

Reserved-Z
Bits 2-7 of byte 20, and bytes 21-23 of the parameter list are reserved and contain binary zeros.

Remove-ALE Function

This function sets a specified ALE in your virtual machine's host access list to the unused state. For the remove-ALE function, the Rx register contains the real address of a doubleword-aligned parameter list in the following format:



Where:

ALSDIAGC
Bytes 0-1 of the parameter list contain the hexadecimal diagnose code. A specification exception is recognized if this field is not X'0240'.

ALSFCODE
Bytes 2-3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0001' indicates the remove-ALE function.

ALSDWLEN
Bytes 4-5 of the parameter list contain the length of the parameter list in doublewords.

ALSVERSN
Bytes 6-7 of the parameter list contain a version number of X'0001'.

ALSALET
Bytes 16-19 of the parameter list contain an ALET which designates the ALE to be removed.

Reserved-Z
Bytes 20-23 of the parameter list are reserved and contain binary zeros.

Bytes 8-15 are ignored.

Responses

Program Exceptions: The program exceptions for DIAGNOSE X'240' are the same as those documented for the ALSERV macro; refer to "ALSERV — Access List Services" on page 815 for more information.

DIAGNOSE Code X'244' – Mapping Services

Privilege Class: Any (XC virtual machines only)

Addressing Mode: 24-bit or 31-bit

DIAGNOSE X'244' is invoked from the guest MAPMDISK macro to perform mapping-service functions.

Note: This diagnose is reserved for IBM use; it is not a supported programming interface. The information is included here to be used only in the diagnosis task. The supported method of invocation for the mapping services is the MAPMDISK macro. For more information on the MAPDISK macro, refer to "MAPMDISK — Mapping Services" on page 824.

The following mapping-service functions can be invoked using this DIAGNOSE:

- Identify-pool
- Define-mapping
- Remove-mapping
- Save-list.

Entry Values:

Rx
Address of a function parameter list, the format of which is determined by the function code in the parameter list. The parameter list is built by the MAPMDISK macro expansion in the macro's work area.

Ax
Is used only in access-register mode. Ax contains the ALET for the address space containing the parameter list.

When Rx is general register 0, Ax is not examined. The ALET is assumed to be X'00000000', which indicates the host-primary address space.

Exit Values:

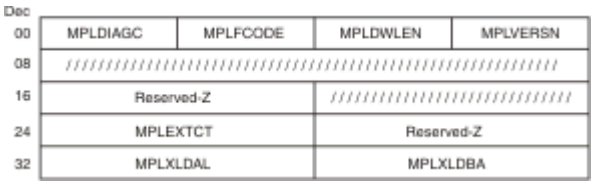
Ry
On return, register Ry contains a return code, as stated in the mapping-service functions described under the "MAPMDISK — Mapping Services" on page 824.

Ry+1
On an error return, register Ry+1 contains the address of the parameter, definition block, or entry where the error was encountered.

Ay+1
Is used only in access-register mode. Ay+1 contains the ALET for the address space containing the address returned in Ry+1.

Identify-pool Function

This function identifies the minidisk pool that will participate in mappings, and defines the assignment of pool-relative block numbers to the blocks contained in the minidisk pool. For the identify-pool function, register Rx contains the real address of a doubleword-aligned parameter list in the following format:



Where:

MPLDIAGC
Bytes 0-1 of the parameter list contain the hexadecimal diagnose code, X'0244'.

MPLFCODE
Bytes 2-3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0000' indicates the identify-pool function.

MPLDWLEN
Bytes 4-5 of the parameter list contain the length of this parameter list in doublewords.

MPLVERSN
Bytes 6-7 of the parameter list contain a version code of X'0001'.

Reserved-Z
Bytes 16-19 and 28-31 of the parameter list are reserved and contain binary zeros.

MPLEXTCT
Bytes 24-27 of the parameter list contains the total number of extents in the extent list.

MPLXLDAL
Bytes 32-35 of the parameter list contain the ALET for the address space containing the first extent-list definition block in the chain of extent-list definition blocks. The contents of this field are ignored in primary space mode.

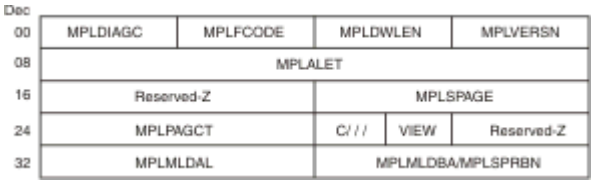
MPLXLDBA
Bytes 36-39 of the parameter list contain the address of the first extent-list definition block in the chain of extent-list definition blocks for this request.

Bytes 8-15 and bytes 20-23 of the parameter list are ignored.

Define-mapping Function

This function establishes a mapping between a range of pages in an address space, and, indirectly via pool-relative block numbers, a set of blocks residing in the minidisk pool.

For the define-mapping function, register Rx contains the address of a doubleword-aligned parameter list in the following format:



Where:

MPLDIAGC
Bytes 0-1 of the parameter list contain the hexadecimal diagnose code, X'0244'.

MPLFCODE
Bytes 2-3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0001' indicates the define-mapping function.

MPLDWLEN
Bytes 4-5 of the parameter list contain the length of this parameter list in doublewords.

MPLVERSN
Bytes 6-7 of the parameter list contain a version code of X'0001'.

MPLASIT
Bytes 8-15 of the parameter list contain the ASIT associated with the address space that is to be the target of the mapping.

Reserved-Z
Bytes 16-19, bits 1-7 of byte 28, and bytes 30-31 are reserved and contain binary zeros.

MPLSPAGE
Bytes 20-23 of the parameter list contain the absolute address of the first page in the consecutive range of pages to be mapped.

MPLPAGCT
Bytes 24-27 of the parameter list contains the number of consecutive pages to be mapped by this request.

C
Bit 0 of byte 28 of the parameter list controls whether the pool-relative block numbers to be associated with the mapped pages are specified as a consecutive range, or by a list.

If this bit is zero, then the mapping is to be established using a specified list of pool-relative block numbers. If this bit is one, then the mapping is to be established using a consecutive range of pool-relative block numbers.

VIEW
Byte 29 of the parameter list contains the page view codes.

A value of binary 0 indicates that the current contents of the pages are discarded and the contents of the associated minidisk blocks are made available in the mapped pages. A value of binary 1 indicates the current contents of the pages are retained.

A value of binary 2 indicates the current contents of the pages are discarded and the mapped pages are considered to contain binary zeros.

MPLMLDAL
When bit 0 of byte 28 is zero, bytes 32-35 contain the ALET of the address space containing the first mapping list definition block. This field is ignored if the virtual machine is in primary space mode or if the pages are to be mapped to consecutive block numbers.

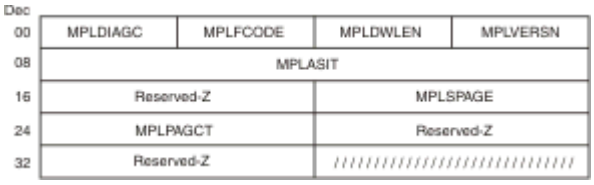
MPLMLDBA
When the C field (bit 0 of byte 28) is zero, bytes 36-39 of the parameter list contain the address of the first mapping list definition block in the chain of mapping definition blocks for this request.

MPLSPRBN
When the C field (bit 0 of byte 28) is one, bytes 36-39 of the parameter list contain the pool-relative block number to be associated with the page identified by the MPLSPAGE field. Each subsequent page in the range of pages to be mapped is associated with the next larger pool-relative block number.

Remove-mapping function

This function removes a mapping between a range of pages in an address space, and a set of blocks residing in the minidisk pool.

For the remove-mapping function, register Rx contains the address of a doubleword-aligned parameter list in the following format:



Where:

MPLDIAGC
Bytes 0-1 of the parameter list contain the hexadecimal diagnose code, X'0244'.

MPLFCODE
Bytes 2-3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0002' indicates the remove-mapping function.

MPLDWLEN
Bytes 4-5 of the parameter list contain the length of this parameter list in doublewords.

MPLVERSN
Bytes 6-7 of the parameter list contain a version code of X'0001'.

MPLASIT
Bytes 8-15 of the parameter list contain the ASIT associated with the address space that is the target of the mapping.

Reserved-Z
Bytes 16-19 and bytes 28-35 of the parameter list are reserved and must be zeros.

MPLSPAGE
Bytes 20-23 of the parameter list contain the address of the first page in the range of pages for which the mapping is to be removed.

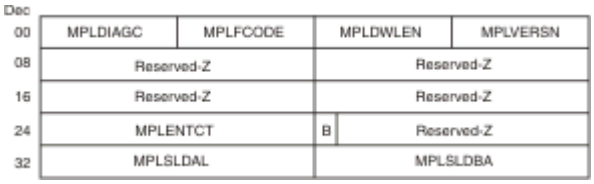
MPLPAGCT
Bytes 24-27 of the parameter list contains the count of pages for which the mapping is to be removed.

Bytes 36-39 of the parameter list are ignored.

Save-list Function

This function saves a specified range of pages in an address space.

For the save-list function, register Rx contains the address of a doubleword-aligned parameter list in the following format:



Where:

MPLDIAGC
Bytes 0-1 of the parameter list contain the hexadecimal diagnose code, X'0244'.

MPLFCODE
Bytes 2-3 of the parameter list contain the hexadecimal function code for the diagnose. A function code of X'0003' indicates the save-list function.

MPLDWLEN
Bytes 4-5 of the parameter list contain the length of this parameter list in doublewords.

MPLVERSN
Bytes 6-7 of the parameter list contain a version code of X'0001'.

Reserved-Z
Bytes 8-23, 29-31, and bits 1-7 of byte 28 of the parameter list are reserved and contain binary zeros.

MPLENTCT
Bytes 24-27 of the parameter list contain the total count of entries in the SLDBK.

B
when bit 0 of byte 28 is one it indicates that the SLDBK is in ***block form***.

If bit 0 of byte 28 is zero then the SLDBK is in the ***list form***.

MPLSLDAL
Bytes 32-35 contain the ALET of the first save-list definition block. This field is ignored if the virtual machine is in primary space mode.

MPLSLDBA
bytes 36-39 of the parameter list contain the address of the first save-list definition block in the chain of save-list definition blocks for this request.

The format of the save-list definition block chain is described in "MAPMDISK SAVE" on page 846.

Responses

Program Exceptions: The program exceptions for DIAGNOSE X'244' are the same as those documented for the MAPMDISK macro at "MAPMDISK — Mapping Services" on page 824.

DIAGNOSE Code X'254' – Access Real Subsystem

Privilege Class: Any, with the LIBRARY CTL option of the STDEVOPT directory control statement.

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'254' to issue a limited set of channel programs to select tape control unit subsystems. DIAGNOSE X'254' provides a virtual machine with asynchronous access to a subsystem even if all devices in that subsystem are dedicated to other virtual machines or CP. The initiating virtual machine is not required to have a device from the target subsystem in its virtual configuration.

Function Codes:

X'00'
Open CP Connection

X'01'
Close CP Connection

X'02'
Perform I/O

Entry Values:

Rx
is a general register that contains the guest real address of the Access Real Subsystem Parameter List (HCPARSPL). The ARSPL must be on a doubleword boundary.

Ry
is a general register. It is not examined as input. Rx and Ry can be the same register.

Exit Values:

Ry
contains a return code indicating the result of the request. Refer to "Responses" on page 1038 for a description of the possible values.

The entire ARSPL may be replaced in guest storage upon completion of a Diagnose X'254' function.

Hardware Specifications

Tape Subsystems

Select tape subsystems are supported by this DIAGNOSE. A small subset of CCWs are supported. The intent is to give a virtual machine the ability to issue library function CCWs to a tape library subsystem without requiring that the virtual machine acquire a dedicated tape drive from the library.

Supported TAPE Subsystems

- 3490 controller subsystems
- 3590 controller subsystems
- 3494 library subsystems
- 3495 library subsystems

Subsystem Identifiers

Controller
Subsystem ID can be found in bytes 112 - 125 of the response to the Read Configuration Data CCW. Bytes 112 - 113 are the Token NED's Plant of Manufacture, and bytes 114 - 125 are the Token NED's Sequence Number.

Library
Subsystem ID can be found in bytes 80 - 93 of the response to the Read Configuration Data CCW. Bytes 80 - 81 are the Library NED's Plant of Manufacture, and bytes 82 - 93 are the Library NED's Sequence Number.

The Subsystem ID will be needed as input to execute all functions of this DIAGNOSE.

Supported CCWs

- Channel Path Nop
- Perform Subsystem Function
- Read Subsystem Data
- Read Configuration Data
- Read Device Characteristics
- Sense Id

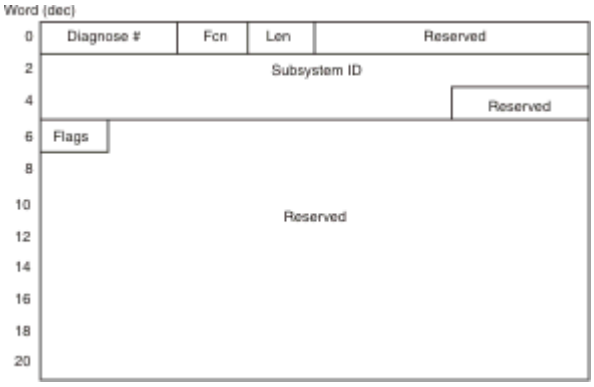
Further Restrictions

- The MAINTCCW option of the OPTION directory control statement is required to issue the Pin and Unpin orders of the Perform Subsystem function CCW.
- Because this DIAGNOSE allows I/O to be issued to a drive that may be dedicated to another VM guest, or may be shared by another real host, I/O issued by this DIAGNOSE is subject to Dynamic Partitioning (DP) Unit Checks. If a DP Unit Check occurs and the DIAGNOSE user has not indicated a specific drive to issue the I/O on, VM will attempt to find another drive in the specified subsystem. If VM receives DP Unit Checks from all available drives, VM will reflect the last Unit Check to the DIAGNOSE user. If VM starts I/O to another drive because of a previous DP Unit Check and receives a non-DP error, the non-DP error will be reflected to the DIAGNOSE user. Use the Dynamic Partitioning Override bit to avoid DP Unit Checks on applicable CCW orders.
- Use of a CCW order in a way that produces an unsolicited interrupt accompanied by an attention message is not advised. This diagnose will have no knowledge of the interrupt message pair when presented by the subsystem. The interrupt message pair will be presented to the virtual machine that has dedicated to it the device that reflected the interrupt message pair, or VM will throw away the interrupt message pair if the associated device isn't attached to a guest. Leave off the Message Required bit associated with any applicable orders to avoid unsolicited interrupts and attention messages.
- When using this DIAGNOSE to issue I/O to a tape library subsystem, the Library Manager must be online when the first device on a controller in the library subsystem is defined and varied on (for example, VM initialization). VM will not be able to access devices on a controller if the Library Manager was offline because VM will have no knowledge of the Library Subsystem ID. You can correct this situation by deleting all RDEVs (for example, DELETE RDEVICE command) for devices on a controller and then recreating and varying on the RDEVS (for example, SET RDEVICE and VARY commands) when the Library Manager is online. This restriction does not apply to the Controller Subsystem ID.
- The Open CP Connection function will not be allowed for a tape library subsystem that has an active DIAGNOSE X'254' environment on a controller subsystem that is part of the library subsystem. Use the Open function consistently on either a library subsystem or a controller subsystem. An active environment on a library subsystem will give access to all controller subsystems in the library. See “Open CP Connection” on page 1033 for details on the DIAGNOSE X'254' environment.

Open CP Connection

This function establishes the necessary environment to invoke subsequent functions of this DIAGNOSE.

The function code for this request is X'00'. The ARSPL is defined as follows:



Diagnose Number
(Input) Byte 0 of word 0 starts a halfword field that defines the DIAGNOSE number. The field must contain the halfword X'0254'.

Function Code
(Input) Byte 2 of word 0 is a 1-byte field that defines the function code of this request. The byte must contain X'00'.

Parameter List Length
(Input) Byte 3 of word 0 is a 1-byte field that defines the length, in bytes, of the ARSPL. Because the ARSPL is a fixed size, the byte must contain X'58'.

Subsystem ID
(Input) Byte 0 of word 2 starts a 14-byte field that defines the Subsystem ID of the subsystem that will be the target of subsequent DIAGNOSE X'254' requests. The contents of the field must be right justified and padded on the left with binary zeros, if necessary, to fill all 14 bytes. Refer to “Hardware Specifications” on page 1032 for more information on subsystem identifiers.

Hardware Flags
(Input) Byte 0 of word 6 is a 1-byte field containing a set of flags related to the hardware in the subsystem. The following table describes all defined flag bits, all other bits are reserved and must contain binary zeros.

Table 215. Hardware Flags

Bit	Description
0	Indicates the type of tape subsystem specified by the Subsystem ID field. If 1, the Subsystem ID represents a tape library subsystem. If 0, a tape controller subsystem is assumed.
1	Indicates whether VM should ignore the Plant of Manufacture bytes in the Subsystem ID field if applicable for the hardware. If 1, the Plant bytes, if applicable, will be ignored. If 0, the Plant bytes, if applicable, will be included in VM's search for the specified subsystem.

Reserved
The remaining fields in the block are reserved and must contain binary zeros.

Usage Notes:

1. The subsystem must be supported by this DIAGNOSE and must not be defined to CP as 'unsupported'. Refer to “Hardware Specifications” on page 1032 for a list of supported subsystems.
2. A given Subsystem ID can have only one active DIAGNOSE X'254' environment at a time among all users on a VM system.

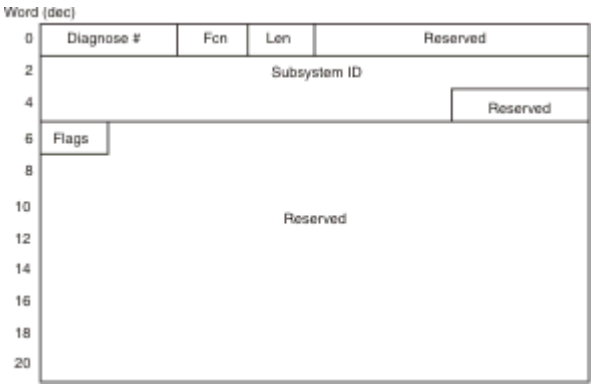
Close CP Connection

This function removes the environment established by the Open CP Connection function of this DIAGNOSE.

The environment established by the Open CP Connection function remains in effect until you explicitly terminate the environment using this function or until an I/O reset is performed for the virtual machine, for example by the SYSTEM RESET, IPL, or LOGOFF commands.

This function, as well as an I/O reset, will terminate any pending DIAGNOSE X'254' I/O on the subsystem.

The function code for this request is X'01'. The ARSPL is defined as follows:



Diagnose Number
(Input) Byte 0 of word 0 starts a halfword field that defines the DIAGNOSE number. The field must contain the halfword X'0254'.

Function Code
(Input) Byte 2 of word 0 is a 1-byte field that defines the function code of this request. The byte must contain X'01'.

Parameter List Length
(Input) Byte 3 of word 0 is a 1-byte field that defines the length, in bytes, of the ARSPL. Because the ARSPL is a fixed size, the byte must contain X'58'.

Subsystem ID
(Input) Byte 0 of word 2 starts a 14-byte field that defines the Subsystem ID of the subsystem that will have its DIAGNOSE X'254' environment removed. The contents of the field must be right justified and padded on the left with binary zeros, if necessary, to fill all 14 bytes.

Hardware Flags
(Input) Byte 0 of word 6 is a 1-byte field containing a set of flags related to the hardware in the subsystem. The following table describes all defined flag bits, all other bits are reserved and must contain binary zeros.

Table 216. Hardware Flags

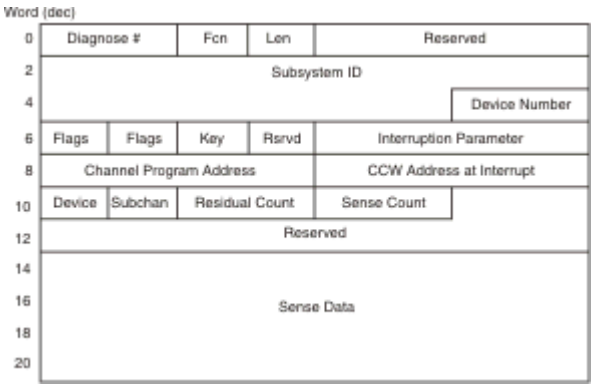
Bit	Description
0	Indicates the type of tape subsystem specified by the Subsystem ID field. If 1, the Subsystem ID represents a tape library subsystem. If 0, a tape controller subsystem is assumed.
1	Indicates whether VM should ignore the Plant of Manufacture bytes in the Subsystem ID field if applicable for the hardware. If 1, the Plant bytes, if applicable, will be ignored. If 0, the Plant bytes, if applicable, will be included in VM's search for the specified subsystem.

Reserved
The remaining fields in the block are reserved and must contain binary zeros.

Perform I/O

This function allows the issuer to issue select channel programs to a subsystem. Refer to “Hardware Specifications” on page 1032 for a list of supported CCWs.

The function code for this request is X'02'. The ARSPL is defined as follows:



Diagnose Number
(Input) Byte 0 of word 0 starts a halfword field that defines the DIAGNOSE number. The field must contain the halfword X'0254'.

Function Code
(Input) Byte 2 of word 0 is a 1-byte field that defines the function code of this request. The byte must contain X'02'.

Parameter List Length
(Input) Byte 3 of word 0 is a 1-byte field that defines the length, in bytes, of the ARSPL. Because the ARSPL is a fixed size, the byte must contain X'58'.

Subsystem ID
(Input) Byte 0 of word 2 starts a 14-byte field that defines the Subsystem ID of the subsystem that will be the target of the DIAGNOSE X'254' I/O request. The contents of the field must be right justified and padded on the left with binary zeros, if necessary, to fill all 14 bytes.

Device Number
(Input) Byte 2 of word 5 starts a halfword field that defines the Real Device Number of the device that will be the target of the DIAGNOSE X'254' I/O request. The device must be a member of the subsystem specified by the Subsystem ID and its number must be in the range X'0000' - X'FFFF'. This field is optional, and VM will choose the device from the specified subsystem if omitted as indicated in the I/O Request Flags field.

Hardware Flags
(Input) Byte 0 of word 6 is a 1-byte field containing a set of flags related to the hardware in the subsystem. The following table describes all defined flag bits, all other bits are reserved and must contain binary zeros.

Table 217. Hardware Flags

Bit	Description
0	Indicates the type of tape subsystem specified by the Subsystem ID field. If 1, the Subsystem ID represents a tape library subsystem. If 0, a tape controller subsystem is assumed.
1	Indicates whether VM should ignore the Plant of Manufacture bytes in the Subsystem ID field if applicable for the hardware. If 1, the Plant bytes, if applicable, will be ignored. If 0, the Plant bytes, if applicable, will be included in VM's search for the specified subsystem.

I/O Request Flags
(Input) Byte 1 of word 6 is a 1-byte field containing a set of flags for the I/O request. The following table describes all defined flag bits, all other bits are reserved and must contain binary zeros.

Table 218. I/O Request Flags

Bit	Description
0	Indicates the format of the CCWs which make up the channel program. If 1, format-1 CCWs are specified. If 0, format-0 CCWs are specified.
1	Indicates whether the device number is indicated in the ARSPL Device Number field. If 1, the device is specified. If 0, the Device Number field is ignored and the device will be chosen by VM.

Storage Key
(Input) Bits 0-3 of byte 2, word 6 contain the Storage Protection Key for fetching all CCWs and CCW related output data, and for storing all CCW related input data. The key is matched against the appropriate guest real storage key during all CCW related storage references. Bits 4-7 of byte 2, word 6 are reserved and must contain binary zeros.

Interruption Parameter
(Input) Byte 0 of word 7 starts a fullword field that defines user data to be stored at guest real storage locations 128-131 upon presentation of the Access Real Subsystem External Interruption at the completion of the I/O request. Refer to "Access Real Subsystem External Interruption" on page 1040 for a complete description.

Channel Program Address
(Input) Byte 0 of word 8 starts a fullword field that designates the location of the channel program's first CCW in guest absolute storage. If format-0 CCWs have been specified in the I/O Request Flags, then bits 0 thru 7 of the Channel Program Address must be binary zeros. If format-1 CCWs have been specified, then bit 0 must be zero. Also, the three rightmost bits of the Channel Program Address must be binary zeros. This indicates that the first CCW is on a doubleword boundary.

CCW Address at Interrupt
(Output) Byte 0 of word 9 starts a fullword field that designates the guest absolute address of the last executed CCW + 8 in the channel program.

Device Status
(Output) Byte 0 of word 10 is a 1-byte field identifying the conditions in the device, that is, the device in the subsystem to which the I/O was issued, when the channel program ended. The conditions that this byte can represent are defined in the *IBM ESA/390 Common I/O-Device Commands*, SA22-7204.

Subchannel Status
(Output) Byte 1 of word 10 is a 1-byte field identifying the conditions in the subchannel when the channel program ended. The conditions that this byte can represent are defined in the *IBM ESA/390 Principles of Operation*, SA22-7201.

Residual Count
(Output) Byte 2 of word 10 starts a halfword field that identifies the Residual Count from the channel program's ending CCW as defined in the *IBM ESA/390 Principles of Operation*, SA22-7201.

Sense Count
(Output) Byte 0 of word 11 starts a halfword field that defines the number of bytes of sense data present if Unit Check is indicated in the Device Status field.

Sense Data
(Output) Byte 0 of word 14 starts a 32-byte field that contains the sense data, as limited by the Sense Count field, if Unit Check is indicated in the Device Status field.

Reserved
The remaining fields in the block are reserved and must contain binary zeros.

Usage Notes:

1. The guest will receive control back after the I/O request has been accepted. If an error prevents acceptance of the I/O, the error will be indicated by the DIAGNOSE condition code and return code. The completion of any I/O performed will be reflected to the guest by an external interruption. The results of the I/O can be found by first examining the external interruption status and then, if appropriate, the output fields of the ARSPL. Refer to "Access Real Subsystem External Interruption" on page 1040 for a complete description of the external interruption.
2. It is possible that queued or started I/O could be ended by the Close CP Connection function of this DIAGNOSE. This could happen if the Close CP Connection function is processed before the I/O is complete. The Access Real Subsystem External Interruption will indicate this condition if it occurs. It is also possible that a virtual machine I/O reset (for example, caused by SYSTEM RESET, IPL, or LOGOFF commands) could end queued or started I/O before it is complete. In this case, there will be no indication that the I/O was ended because the virtual machine ends up being reset.
3. Unsupported CCWs will receive an indication of Unit Check in the Device Status field and Command Reject in the Sense Data field.
4. The Interruption Parameter field can be used to associate an Access Real Subsystem External Interruption with a particular invocation of the Perform I/O function. Refer to "Access Real Subsystem External Interruption" on page 1040 for more details.

Responses

Condition Codes and Return Codes: Upon completion of DIAGNOSE X'254', control is returned to the issuer with a condition code and return code set to indicate the status of the requested function. The return code is set in Ry.

Table 219. General condition code descriptions for all functions

Condition Code	Description
0	Function was successful.
2	Subsystem is busy.
3	Subsystem is not operational.

Table 220. Condition codes and return codes for the Open CP Connection function

Condition Code	Return Code in Ry	Description
0	0 (X'00')	Initialization of the DIAGNOSE X'254' environment for the specified subsystem has completed successfully.
2	32 (X'20')	The issuer already has a DIAGNOSE X'254' environment active for the specified subsystem.
2	36 (X'24')	Another user currently has a DIAGNOSE X'254' environment active for the specified subsystem.
2	40 (X'28')	A CP Close Connection Function is currently pending for the specified subsystem.
3	52 (X'34')	The specified subsystem is not known in VM's real I/O configuration or is not supported by VM for this DIAGNOSE.
3	60 (X'3C')	Reserved.
3	64 (X'40')	Reserved.
3	255 (X'FF')	An unexpected error occurred. The error is unrecoverable. It will be accompanied by a soft abend.

Table 221. Condition codes and return codes for the Close CP Connection function

Condition Code	Return Code in Ry	Description
0	0 (X'00')	The active DIAGNOSE X'254' environment has been successfully deleted. Any pending DIAGNOSE X'254' I/O to the subsystem has been terminated.
2	40 (X'28')	A CP Close Connection Function is currently pending for the specified subsystem.
3	120 (X'78')	A DIAGNOSE X'254' environment was not previously established for the specified subsystem using the Open CP Connection function.
3	255 (X'FF')	An unexpected error occurred. The error is unrecoverable. It will be accompanied by a soft abend.

Table 222. Condition codes and return codes for the Perform I/O function

Condition Code	Return Code in Ry	Description
0	0 (X'00')	The I/O request has accepted.
2	40 (X'28')	A CP Close Connection Function is currently pending for the specified subsystem.
3	120 (X'78')	A DIAGNOSE X'254' environment was not previously established for the specified subsystem using the Open CP Connection function.
3	148 (X'94')	VM is unable to find an available device. Either the specified device or all devices associated with the specified subsystem are offline, or they do not exist.
3	255 (X'FF')	An unexpected error occurred. The error is unrecoverable. It will be accompanied by a soft abend.

Program Exceptions: DIAGNOSE X'254' may result in one of the following program exceptions:

Problem Encountered	Cause
Addressing exception	The ARSPL is not within addressable guest real storage.
Operand exception	Any of the following: • The ARSPL Channel Program Address does not indicate a CCW that is on a doubleword boundary. • Bits required to be binary zero are set in the high-order byte of the ARSPL Channel Program Address. • Bits required to be binary zero are set in the ARSPL byte containing the Storage Protection Key.
Privileged-operation exception	The guest is not authorized to issue this DIAGNOSE.
Protection exception	The ARSPL is store or fetch protected.

Problem Encountered	Cause
Specification exception	Any of the following: • The ARSPL is not on a doubleword boundary. • The ARSPL Diagnose Number field does not contain X'0254'. • The ARSPL Function Code field contains an invalid value. • The ARSPL Parameter List Length field does not contain X'58'. • Reserved bits are set in the ARSPL Hardware Flags. • Reserved bits are set in the ARSPL I/O Request Flags. • A reserved field in the ARSPL does not contain binary zeros.

Access Real Subsystem External Interruption

An Access Real Subsystem External Interruption is generated if a Perform I/O function request has completed at a device.

The external interruption is a floating interruption condition and is presented to the first virtual CPU in the virtual configuration that is enabled for the interruption. The condition is cleared once the interruption is presented or if a virtual machine reset occurs, for example, by the SYSTEM RESET, IPL, or LOGOFF commands.

The subclass mask to enable for the interruption is bit 22 of control register 0.

The Access Real Subsystem condition is indicated by an external interruption code of X'2603' stored at guest real locations 134-135, and a subinterruption code of X'04' stored at guest real location 132. The Interruption Parameter, as specified in the ARSPL on the Perform I/O request, is stored at guest real locations 128-131. In addition, one of the following status codes will be stored at guest real location 133.

Table 223. Perform I/O function completion status codes

Code	Meaning
X'00'	The I/O completed as indicated in the updated ARSPL.
X'01'	The updated ARSPL could not be stored into guest storage. The results of the I/O operation are indeterminate.
X'02'	The user-specified or VM chosen device became unavailable after the I/O request was queued to be issued to the device. Retry the request to see if the user specified device is now available or if VM can find a different device in which to issue the I/O on.
X'03'	The I/O, which was queued but not yet started, was terminated due to a Close CP Connection function request.
X'04'	The I/O, which was already started at the device but not yet complete, was terminated due to a Close CP Connection function request.
X'05'	A missing interrupt condition was detected. The I/O request may or may not have been started.
X'FF'	An unexpected error occurred. The error is unrecoverable. It will be accompanied by a soft abend.

DIAGNOSE Code X'25C' – Directory Query

Privilege Class: B

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'25C' is a privileged function provided to allow applications access to the data in the CP user directory. Applications should use the VMUDQ macro to invoke the functions contained herein. For more information on the VMUDQ macro, refer to "VMUDQ – VM User Directory Query" on page 875.

Entry Values:

Rx
contains the address of the parameter list.

Ax
is used only in access-register mode in an XC virtual machine. Ax contains the ALET of the address space containing the parameter list.

Ry
contains the address of the data buffer. Register 0 selects the virtual machine's primary address space.

Ay
contains the ALET of the address space.

Ry+1
contains the size of the data buffer in bytes.

Exit Values:

Rx
is modified to contain the length in bytes of the results of the query if successful or the number of bytes needed for the query if the buffer length is not sufficient. Otherwise, it remains unchanged.

Rx+1
might or might not be used. See the description of the function in the Table 224 on page 1042 for details.

Ry and ARy
remains unchanged, describing the location of the data buffer.

Ry+1
is modified to contain the length in bytes of the results of the query.

Table 224. DIAGNOSE X'25C' Function List

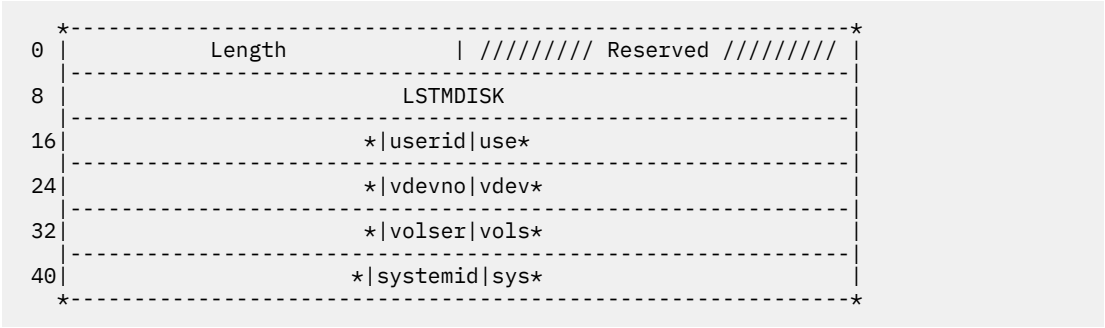
Function	Description and Parameters
LSTMDISK	Creates a list of MDISK definitions based on, or qualified by, the owners to which they belong, virtual device number, serial number of the volume, and the SSI member on which they reside. An asterisk (*) can be used as a trailing wild card to select a wider group of definitions. The list is built in the buffer provided by the caller. **Input Parameter List:** Consists of six doublewords as shown in the following example:  The parameter list is a maximum of six doublewords long. It must be on a doubleword boundary and must not span a page boundary. **length** The length is a fullword containing the length in bytes of the length field itself, the reserved field, and the parameter string that follows. The length field must be one of the following values: 16, 24, 32, 40, or 48. **LSTMDISK** is the name of the function to be performed by the DIAGNOSE. The function must be LSTMDISK. **** | *userid* | *use**** selects MDISK definitions that belong to the specified user IDs. The user ID from a USER or IDENTITY directory statement can be used. SUBCONFIG IDs cannot be specified. Definitions belonging to multiple owner ID's are selected by the use of an asterisk (*). An asterisk (*) by itself indicates that all definitions are to be analyzed regardless of *userid*. An asterisk (*) used as a trailing wild card selects definitions that belong to a group of *userid*s that begin with the specified character combination. The value is left-justified in the field and padded to eight characters with blanks. The default assumed if the field is blank is an asterisk (*). **** | *vdevno* | *vdev**** selects MDISK definitions that are defined to the specified virtual device numbers. Multiple *vdevnos* are specified by the use of the asterisk (*). An asterisk (*) by itself causes all MDISK definitions to be selected regardless of *vdevno*. An asterisk (*) used as a trailing wild card causes selection of MDISK definitions found on the selected virtual devices where the combined value indicates a range of addresses beginning with a common value. A four-digit address is assumed; therefore, 000* would mean 0000 through 000F, and 04* would mean 0400 through 04FF. The value is left-justified in the field and padded to eight characters with blanks. The default assumed if the field is blank is an asterisk (*).

Table 224. DIAGNOSE X'25C' Function List (continued)

Function	Description and Parameters

LSTMDISK (cont'd.)	*** | *volser* | *vols****
	selects MDISK definitions that are found on the volume containing the specified volume serial number. An asterisk (*) can be used as a trailing wild card to select a range of *volsers* that begin with a common set of characters. An asterisk (*) by itself indicates that the definitions on all the volumes are to be analyzed. The value is left-justified in the field and padded to eight characters with blanks. The default assumed if the field is blank is an asterisk (*).
	*** | *systemid* | *sys****
	selects MDISK definitions that are found on the specified SSI member system. An asterisk (*) can be used as a trailing wild card to select a range of systems that begin with a common set of characters. An asterisk (*) by itself indicates that the MDISK definitions on all member systems are to be analyzed. The value is left-justified in the field and padded to eight characters with blanks. If the field is blank, the default assumed is blank, indicating that only the MDISK definitions on the current system are to be analyzed. Specifying a system other than by an asterisk or a blank is allowed only when an SSI-enabled directory is in use.
	Output Buffer Format: Consists of seven blank-delimited fields containing the information indicated in the example below:
	`OWNERID- VDEV VOLSER DEVTYPE- START----- SIZE------ System`
	`12345678 1234 123456 12345678 1234567890 1234567890 12345678`
	`---------+---------+---------+---------+---------+---------+-------`
	`User123  0191 ABC123 3390     0000000010 0000000200`
	`User456  0191 CCC123 FB-512   0000000010 0000200000`
	`User789  0222 CCC222 3380     0000000010 END`
	`UserABC  0223 ABC223 3370     0000000000 0000200000 SYSTEMABC`
	Notes:
	1. In the above example the "START" field can contain either the starting cylinder or block number. The "SIZE" field contains the remaining number of cylinders or blocks. The non-device specific "END" in the size field denotes the end of the volume. Also, the header lines are not part of the returned data.
	2. If the MDISK was defined with a size of "END", then the size returned in the output buffer is simply "END".
	3. Temporary disks (T-disks) and virtual disks in storage are ignored by the LSTMDISK function and are not included in the output buffer.
	4. If the MDISK is defined within a SUBCONFIG stanza, the system ID to which the MDISK is restricted is included in the output buffer. If the MDISK is defined within a user or identity stanza, then no system ID is specified because the MDISK is not restricted to a particular system.

Usage Notes

1. The application programmer uses the VMUDQ macro to invoke this DIAGNOSE because this DIAGNOSE code is reserved for IBM use.
2. An address space is acquired in blocks of 256 pages; therefore the number of bytes in the buffer to be passed on the call would equal:

```
(number pages requested+255/256) * 1048576.
```

3. In an SSI-enabled directory, MDISK definitions can be global or local. A global MDISK definition is one that is included in a user or identity stanza and the minidisk being defined can be linked by virtual machines on any member system of the SSI cluster. A local MDISK definition is one that is included in a SUBCONFIG stanza and the minidisk being defined can be linked only by virtual machines on the SSI

member system to which the SUBCONFIG stanza applies. When specifying the system ID parameter (* | *systemid* | *sys**), you are requesting all MDISK definitions that are global and local to a particular system. If you specify a system ID that is not specified on any BUILD statements in the directory, then all global MDISK definitions are reported. If you specify a system ID that is specified on a BUILD statement in the directory, then all global MDISK definitions and local MDISK definitions for the specified system are reported.

Responses

Condition codes and return codes are described in Table 225 on page 1044 and Table 226 on page 1044.

Table 225. DIAGNOSE X'25C'—Condition codes

Condition Code	Meaning
0	A normal exit from DIAGNOSE X'25C'. The guest Ry+1 register contains the return code.
1	DIAGNOSE code X'25C' has detected an error.
3	DIAGNOSE code X'25C' has encountered a paging error while trying to access the target area in the data space or an object directory page.

Table 226. Diagnose X'25C' - Return codes

Function	Return Code in Ry +1	Description
General	X'00'	Successful. The guest Rx register contains the length of bytes moved as the result of the query.
	X'04'	No records were found to match criteria specified.
	X'08'	The buffer length supplied is insufficient. The guest Rx register contains the number of bytes needed for the query.
LSTMDISK	X'100'	An invalid user ID was specified.
	X'104'	An invalid virtual device number was specified.
	X'108'	An invalid volume serial number was specified.
	X'10C'	An invalid system ID was specified.

Program Exceptions: The program exceptions for DIAGNOSE X'25C' are the same as those documented for the VMUDQ macro; refer to "VMUDQ – VM User Directory Query" on page 875 for more information.

DIAGNOSE Code X'264' – CP Communication

Privilege Class: ANY

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'264' defines an area in which CP stores notification of certain events.

Entry Values:

Rx, Rx+1, Ry+1
Depends on the subcode specified in Ry (see description below).

Ry
Function subcode.

Program Exceptions: CP returns a specification exception if the subcode specified in the Ry register is invalid.

Subcode X'00000000'—Establish CP communication area

Table 227. CP Subfunctions

Name	Bits #	Definition
CPCRECON	0	Set to one (1) when a RECONNECT of the virtual machine occurs
CPCFSLM	1	Set to one (1) when a virtual line-mode write has completed successfully and set to zero (0) when a virtual full-screen write has completed successfully.
CPCSHORT	2	Set to one (1) when the virtual machine default date format is SHOrtdate.
CPCFULL	3	Set to one (1) when the virtual machine default date format is FULldate.
CPCISO	4	Set to one (1) when the virtual machine default date format is ISOdate.

Entry Values:

Rx
Contains the guest absolute address of CP communication area in the host-primary space. The communication area must not be in read-only storage and must not cross a page boundary. Key-controlled protection and low-address protection do not apply to references to the communication area.

Rx+1
Contains the size of CP communication area in fullwords

Usage Notes:

1. Once established, the CP communication area remains active until the virtual configuration is reset or until DIAGNOSE X'264' function subcode X'00000004' removes it.
2. The communication area may become inaccessible or read-only as, for example, through DIAGNOSE X'64'. As long as this prevails, the area is not updated.
3. The guest should use Compare and Swap when modifying the CP communication area.
4. Only bits defined within the established CP communication area are updated.

Program Exceptions:

Problem Encountered	Cause
Specification exception	Any of the following: • The CP communication area is not aligned on a fullword boundary. • The CP communication area crosses a page boundary. • The CP communication area is already active for the virtual configuration. • The size, in fullwords, in the Rx+1 register is not positive.
Addressing exception	The address of the CP communication area is not valid.
Protection exception	The specified communication area is in read-only storage.

Subcode X'00000004'—Remove CP communication area

Entry Values:

Rx
Not used.

Rx+1
Not used.

Ry+1
Not used.

Usage Note: If the CP communication area has not been established, no error is presented.

Responses: None.

DIAGNOSE Code X'278' – Extract XLINK Control Blocks

Privilege Class: G

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'278' extracts XLINK Control Block data from CP and returns it to a virtual machine.

Note: This DIAGNOSE is reserved for IBM use. It is not a supported programming interface. It is included here only for diagnostic purposes.

Entry Values:

Rx
Request flags in byte 0; bytes 1 through 3 must be zero.

X'80'
DXLSYINR — Request for System Include list

X'40'
DXLSYEXR — Request for System Exclude list

X'20'
DXLDTABR — Request for Device Table

X'10'
DXLVLINR — Request for Volume Include list

X'08'
DXLVLEXR — Request for Volume Exclude list

All undefined bits must be 0; otherwise, a specification exception occurs.

Ry
Address of doubleword-aligned parameter list, HCPDXLPL. Bit 0 of Ry must be zero.

Ay
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the override parameter list file name. If Ry designates general register 0, if Ay contains X'00000000', or if the virtual machine is not in XC mode, the override parameter list file name is in the host-primary address space.

Exit Values: The following are normal exit situations.

Condition Code	Meaning
0	Normal completion. The response area was large enough for the data, and DXLDLEN and DXLDATA have been updated.
3	The response area was not large enough for the data. DXLDLEN has been updated, but DXLDATA is unpredictable.

On entry, the parameter list must be filled in as follows:

DXLHDR

0	DXLPLID	//////////	DXLBLEN	////////

DXLHDR
indicates the beginning of the input fields.

DXLPLID
is a halfword identifier for the parameter list. This field must contain the value X'0278'.

DXLBLEN
is a halfword containing the number of doublewords available in the DXLPL area. The DXLPL area comprises both the header and the response area. This value must be at least 1.

Between the parameter list and the response area, there is a field set by CP:

DXLDLEN
is a halfword containing the actual number of doublewords of DXLPL into which results have been (or will be) placed. This number includes one doubleword for the header. If the guest condition code is 0, then DXLDATA contains data. If the guest condition code is 3, then the contents of DXLDATA are unpredictable.

DXLHDR

0	////////	//////////	////////	DXLDLEN

The response area, DXLDATA, immediately follows DXLDLEN:

DXLDATA

8	DXLSYINO	DXLSYINC	DXLSYINL	//////////////
10	DXLSYEXO	DXLSYEXC	DXLSYEXL	//////////////
18	DXLVLINO	DXLVLINC	DXLVLINL	//////////////
20	DXLVLEXO	DXLVLEXC	DXLVLEXL	//////////////
28	DXLDTABO	DXLDTABC	DXLDTABL	//////////////
30	Response information at offsets above.			

where:

DXLDATA
is the data from CP, consisting of the fields listed below. Only the fields pertaining to the data items requested in Rx are meaningful; the contents of the other fields are unpredictable.

DXLSYINR
XLINK included system names:

DXLSYINO
shows the offset, in doublewords, from the beginning of the parameter list to the system include list. DXLSYINO is two bytes long.

DXLSYINC
shows the number of entries in the system include list. DXLSYINC is two bytes long.

DXLSYINL
is the length of each entry in bytes. DXLSYINL is two bytes long.

The last two bytes of the field are reserved.

The response for each entry will be the following:

Eight-byte system name

DXLSYEXR
XLINK excluded system names:

DXLSYEXO
shows the offset, in doublewords, from the beginning of the parameter list to the system exclude list. DXLSYEXO is two bytes long.

DXLSYEXC
shows the number of entries in the system exclude list. DXLSYEXC is two bytes long.

DXLSYEXL
is the length of each entry in bytes. DXLSYEXL is two bytes long.

The last two bytes of the field are reserved.

The response for each entry will be the following:

Eight-byte system name

DXLVLINR
XLINK volume include list:

DXLVLINO
shows the offset, in doublewords, from the beginning of the parameter list to the volume include list. DXLVLINO is two bytes long.

DXLVLINC
shows the number of entries in the volume include list. DXLVLINC is two bytes long.

DXLVLINL
is the length of each entry in bytes. DXLVLINL is two bytes long.

The last two bytes of the field are reserved.

The response for each entry will be the following:

Six-byte volume serial pattern
Two-byte cylinder number
Two-byte track number
Two-byte record length
Two-byte records

DXLVLEXR
XLINK volume exclude list:

DXLVLEXO
shows the offset, in doublewords, from the beginning of the parameter list to the volume exclude list. DXLVLEXO is two bytes long.

DXLVLEXC
shows the number of entries in the volume exclude list. DXLVLEXC is two bytes long.

DXLVLEXL
is the length of each entry in bytes. DXLVLEXL is two bytes long.

The last two bytes of the field are reserved.

The response for each entry will be the following:

Six-byte volume serial pattern

DXLDTABR
XLINK device table:

DXLDTABO
shows the offset, in doublewords, from the beginning of the parameter list to the device table. DXLDTABO is two bytes long.

DXLDTABC
shows the number of entries in the device table. DXLDTABC is two bytes long.

DXLDTABL
is the length of each entry in bytes. DXLDTABL is two bytes long.

The last two bytes of the field are reserved.

The response for each entry will be the following:

One-byte device type (from HCPDVTYP)
One-byte model/etc. information

Two-byte cylinder number
Two-byte track number
Two-byte record length
Two-byte records

Responses

Program Exceptions:

Problem Encountered	Cause
Specification exception	One of the following occurred: • Undefined bits were set in Rx • Parameter list address was negative or not doubleword-aligned • Field DXLPLID was not X'0278' • Field DXLBLEN was less than or equal to zero.
Access exception	An error occurred either while fetching from the parameter list or while storing in the response area.

DIAGNOSE Code X'280' – Set POSIX IDs - security values

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'280' securely sets POSIX security values for one of the family of *exec()* functions. If the file to be processed is not a set_UID or set_GID file, it sets the saved set-IDs to the effective ID values. If the file is a set_UID or set_GID file, it communicates with the server and then sets the effective and saved set-IDs to the specified values.

Entry Values:

Rx
is the general register that contains the guest real address of the exec() parameter list (HCPEXCBK). The EXCBK must be on a doubleword boundary.

The entire EXCBK may be replaced in guest storage when this function completes.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the exec() parameter list (HCPEXCBK). If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the exec() parameter list is in the host-primary address space.

Active Process ID
The active process is specified in the active process ID (PID) field in the POSIX communication area. (Before DIAGNOSE X'280' is used, DIAGNOSE X'2A4' must have been issued in this virtual configuration to define the communication area and allocate one or more PIDs.)

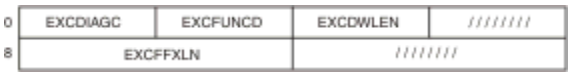
Exit Values:

Ry
Contains the return code indicating the result of this request. Refer to “Responses” on page 1052 for a description of the possible values of this field. Ry cannot be the same register as Rx.

Format of the DIAGNOSE X‘280’ parameter list - HCPEXCBK: The EXCBK consists of two major sections: the header, and the function-specific section. The function-specific section is mapped differently for each function, and may include both fixed-length and variable length portions. For upward compatibility, the field EXCFFXLN, the length of the fixed-length portion of the function-specific section,

must be filled in during the execution-time creation of the block. Any subsequent references to the variable-length portion must always be based on the address of the beginning of the block plus the lengths of the header and the fixed-length portion of the function-specific section.

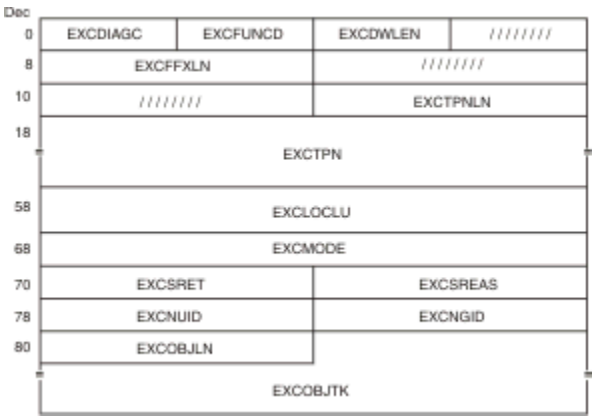
The format of the header is:



Function EXCSETID - Request changes in POSIX security values for an exec() function call

This function is called when the virtual configuration's POSIX security values must be changed on behalf of one of the *exec()* functions.

Format of the HCPEXCBK for function EXCSETID



Input:

EXCDIAGC
is a halfword field containing the hexadecimal DIAGNOSE code, X'0280'.

EXCFUNCD
is a halfword field containing the function code. For this function, EXCFUNCD must contain the value of EXCSETID (0).

EXCDWLEN
is an unsigned halfword field containing the length of this parameter list in doublewords. The length must include the length of the header, the length of the fixed-length section, and the length of any specified variable-length sections. It must be no more than one page (512 doublewords) in length.

EXCFFXLN
is an unsigned fullword field containing the length in bytes of the fixed-length portion of the function-specific area of the control block. This area starts immediately after the header, and includes the fields up to, but not including, the beginning of the object token length field (EXCOBJLN).

Reserved header fields
The rest of the fields in the header are reserved and must contain binary zeros.

EXCTPNLN
is an unsigned fullword field containing the length in bytes of the TPN specified in EXCTPN. The minimum TPN length is 1 byte, and the maximum length is 64 bytes.

EXCTPN
is a 64-byte field containing the resource name of the server in which the object to be executed resides. The TPN may be between 1 and 64 bytes in length and must be left-justified.

EXCLOCLU
is a 16-byte field containing the locally-known LUname of the target server. This field contains 2 8-byte parts: the luname qualifier (EXCLUQUL), and the target luname (EXCLUTGT), and must follow the rules for specifying a locally-known luname for a global resource.

EXCMODE
is an 8-byte field containing the SNA modename.

EXCSRET
is an unsigned fullword field set by this function to contain the return code set by the server.

EXCSREAS
is an unsigned fullword field set by this function to contain the reason code set by the server.

EXCNUID
is an unsigned fullword field set by this function. If the function completed successfully, the field contains the current effective UID.

EXCNGID
is an unsigned fullword field set by this function. If the function completed successfully, the field contains the current effective GID.

EXCOBJLN
is an unsigned fullword field containing the length of the object token data (in bytes). The value in this field must include the length of this field and the length of the object token specified in EXCOBJTK, and thus must be at least equal to the length of this field. The maximum value for this field is EXCOBJMX bytes.

EXCOBJLN must immediately follow the fixed-length portion of this function's information.

EXCOBJTK
is a variable-length character field containing the token representing the file to be executed. The token must be left-justified.

Reserved
The rest of the fields in this block are reserved and should contain binary zeros.

All other flag bits must be binary zeros.

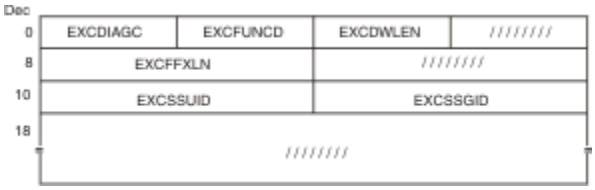
All other fields of the parameter list are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE X'280' are unpredictable.

Output: See return codes.

Function EXCSSID - Request changes in saved set-IDs for an exec() function call

This function is called when an application in a virtual machine performs an *exec()* function call, and only the saved set-ID values must be changed.

Format of the HCPEXCBK for function EXCSSID



Input:

EXCDIAGC
is a halfword field containing the hexadecimal DIAGNOSE code, X'0280'.

EXCFUNCD
is a halfword field containing the function code. For this function, EXCFUNCD must contain the value of EXCSSID (1).

EXCDWLEN
is an unsigned halfword field containing the length of this parameter list in doublewords. The length must include the length of the header and the length of the fixed-length section. It must be no more than one page (512 doublewords) in length.

EXCFFXLN
is an unsigned fullword field containing the length in bytes of the fixed-length portion of the function-specific area of the control block. This area starts immediately after the header, and includes the fields up to and including field EXCSSGID (the end of the parameter list for this function).

Reserved header fields
The rest of the fields in the header are reserved and must contain binary zeros.

EXCSSUID
is an unsigned fullword field set by this function. If the function completed successfully, the field contains the current saved set-UID.

EXCSSGID
is an unsigned fullword field set by this function. If the function completed successfully, the field contains the current saved set-GID.

Reserved
The rest of the fields in this block are reserved and should contain binary zeros.

All other fields of the parameter list are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE X'029C' are unpredictable.

Output: See return codes.

Responses

Upon completion of DIAGNOSE code X'280', control is returned to the invoker with a condition code set to indicate the status of both input parameter list processing and the function requested. A return code in Ry further defines that status.

The condition code and the return code in Ry supersede all other status information, including that in the return and reason code fields in the EXCBK. Those fields are set by the server, but are only valid if the return code in Ry indicates that they should be checked for further information.

Table 228 on page 1052 contains a general description of each of the condition codes.

Table 228. DIAGNOSE X'280' condition codes

Condition Code	Meaning
0	Function completed successfully.
1	Function failed. HCPEXCBK indicates the return and reason codes from the server.
2	Function failed. The return code in Ry indicates the reason for the failure.

Return codes and their corresponding condition codes for the function to set POSIX security values for an *exec()* function call are listed in Table 229 on page 1052.

Table 229. Condition codes and return codes for changing effective and saved set-IDs.

Condition Code	Return Code in Ry			Description
	Decimal value	Hex value	Symbol	
0	0	X'00'	EXCOK	The function completed successfully.
2	4	X'04'	EXCIDIAG	The DIAGNOSE code specified in EXCDIAGC is incorrect.
2	8	X'08'	EXCBDFUN	The function code specified in EXCFUNCD is incorrect.
2	12	X'0C'	EXCNZERO	Reserved fields do not contain binary zeros.

Table 229. Condition codes and return codes for changing effective and saved set-IDs. (continued)

Condition Code	Return Code in Ry			Description
	Decimal value	Hex value	Symbol	
2	16	X'10'	EXCBDDWL	The length specified in EXCDWLEN is not valid. It must be • greater than or equal to the header length (EXCHDRLN) • less than or equal to a page (512 doublewords) in length • greater than or equal to the minimum specified for the function • equal to the sum of the header, the function-specific fixed-length section, and the variable section of the block.
2	20	X'14'	EXCBDFXL	The length specified in EXCFFXLN is not valid.
2	24	X'18'	EXCBDTPL	The length specified in EXCTPNLN is not valid. It must be greater than or equal to EXCTPNMN and less than or equal to EXCTPNMX.
2	28	X'1C'	EXCBDOBL	The length specified for the object token is not valid. It must be greater than or equal to EXCOBJMN and less than or equal to EXCOBJMX.
2	32	X'20'	EXCINVLU	The locally-known LU name is not valid.
2	36	X'24'	EXCNOGAT	The gateway was not found.
2	40	X'28'	EXCNORES	The resource was not found.
2	44	X'2C'	EXCNORPY	The request was terminated because the server did not reply within the allotted time.
2	48	X'30'	EXCNOSRV	The server is unable to handle a request for POSIX security values for one or more of the following reasons: • the server has not identified itself as prepared to handle a request for POSIX security values • the server is not permitted to set POSIX security values.
2	52	X'34'	EXCSVREJ	The server rejected the request.

Table 229. Condition codes and return codes for changing effective and saved set-IDs. (continued)

Condition Code	Return Code in Ry			Description
	Decimal value	Hex value	Symbol	
1	56	X'38'	EXCSVERR	The server has returned an error condition. Refer to the return and reason codes in the EXCBK for details.
2	60	X'3C'	EXCSVINV	The information the server returned was invalid.
2	64	X'40'	EXCUAUTH	The user is not permitted to have his POSIX security values set.
2	68	X'44'	EXCESMRJ	The External Security Manager rejected the specified POSIX security values.
2	72	X'48'	EXCNOCOM	There is no POSIX communication area defined.
2	76	X'4C'	EXCNPROC	There is no valid active POSIX process for which to change the POSIX security values.
2	80	X'50'	EXCTERM	The request was terminated at the user's request (for example, by an exigent command).
2	255	X'FF'	EXCFATAL	An unrecoverable error occurred while processing the DIAGNOSE and a soft abend dump may have been taken.

Return codes and their corresponding condition codes for the function to set saved set-IDs for an *exec()* function call are listed in Table 230 on page 1054.

Table 230. Condition codes and return codes for changing saved set-IDs only.

Condition Code	Return Code in Ry			Description
	Decimal value	Hex value	Symbol	
0	0	X'00'	EXCOK	The function completed successfully.
2	4	X'04'	EXCIDIAG	The DIAGNOSE code specified in EXCDIAGC is incorrect.
2	8	X'08'	EXCBDFUN	The function code specified in EXCFUNCD is incorrect.
2	12	X'0C'	EXCNZERO	Reserved fields do not contain binary zeros.

Table 230. Condition codes and return codes for changing saved set-IDs only. (continued)

Condition Code	Return Code in Ry			Description
	Decimal value	Hex value	Symbol	
2	16	X'10'	EXCBDDWL	The length specified in EXCDWLEN is not valid. It must be • greater than or equal to the header length (EXCHDRLN) • less than or equal to a page (512 doublewords) in length • greater than or equal to the minimum specified for the function • equal to the sum of the header, the function-specific fixed-length section, and the variable section of the block.
2	20	X'14'	EXCBDFXL	The length specified in EXCFFXLN is not valid.
2	72	X'48'	EXCNOCOM	There is no POSIX communication area defined.
2	76	X'4C'	EXCNPROC	There is no valid active POSIX process for which to change the POSIX security values.
2	255	X'FF'	EXCFATAL	An unrecoverable error occurred while processing the DIAGNOSE and a soft abend dump may have been taken.

Program Exceptions: One of the following program exceptions may be reflected to the issuing virtual machine indicating guest or host error conditions. In all cases, no meaningful return code is given, and the guest instruction (Diagnose) is nullified, suppressed, or terminated according to the architecture.

DIAGNOSE code X'280' may result in one of the following program exceptions:

Problem Encountered	Cause
Specification exception	Any of the following: • Rx is the same register as Ry. • The address of the parameter list (EXCBK) specified in Rx is not on a doubleword boundary.
Access exception (See “Access Exceptions” on page 8.)	Attempt to fetch from or store into the HCPEXCBK failed. Regardless of the value of EXCDWLEN, an access exception may be recognized for a minimum of one doubleword at the EXCBK address. Conversely, an access exception may not be recognized for the portion of the operand beyond the length supported by CP for the function requested.

DIAGNOSE Code X'29C' – Set-POSIX-IDs Services

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE X'29C' specifies one of several function codes that designate Set-POSIX-IDs services. The services alter the user IDs (UIDs) and group IDs (GIDs) of the active POSIX process. DIAGNOSE code X'29C' uses the parameter list (SPXBK) to receive input data and, depending on the function requested, to return results. The SPXBK and the codes named below are defined in member HCPSPXBK and HCPOM1 MACLIB.

Any virtual machine can use this diagnose. However certain operations are privileged and require either permission from an External Security Manager (ESM) or authorization in the CP directory.

The following Set-POSIX-IDs services can be invoked using this DIAGNOSE code:

- Set user IDs (UIDs) for the active process — function code SPXFUSER (0)
- Set group IDs (GIDs) for the active process — function code SPXFGRP (1)
- Change the active process' GIDs to designate another group to which the logged-on user belongs — function code SPXFNGRP (2)
- Change the active process' supplementary group id list — function code SPXFSGID (3)

Entry Values:

Rx
The real address of a function parameter list (SPXBK) in guest storage. The SPXBK must be on a doubleword boundary. The format of the parameter list is determined by the function code which is also in the parameter list. The SPXBK contains all the input parameters.

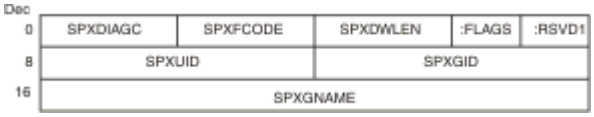
Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

Active Process ID
The active process is specified in the active process ID (PID) field in the POSIX communication area. (Before DIAGNOSE X'29C' is used, DIAGNOSE X'2A4' must have been issued in this virtual configuration to define this area and to allocate one or more PIDs.)

Exit Values:

Ry
On return, contains a return code. Ry can not be the same as Rx.

Format of the HCPSPXBK:



Function SPXFUSER - Set User IDs (UIDs) for the Active Process

This function sets the effective user ID (UID) for the active POSIX process to a given user ID. For a process with appropriate privileges, this function optionally sets the real and saved set-user-IDs for the process to that same user ID as well.

Input:

SPXDIAGC
Contains the hexadecimal DIAGNOSE code, SPXDIAGV (value X'029C').

SPXFCODE
Contains the function code, SPXFUSER (value 0).

SPXDWLEN
Contains the length of this parameter list in doublewords. this value must be at least 2.

SPXFLAGS
Contains flags as follows:

SPXALLID
(bit 1) If this bit is on and the active process has appropriate privileges, set all three UIDs (real, effective, and saved set). If this bit is off, only the effective UID is changed.

If the process is not authorized, this bit is ignored. Only the effective UID is changed.

All other flag bits must be binary zeros.

SPXRSVD1
Reserved for IBM used; must contain binary zeros.

SPXUID
Value to which this process' UID(s) are to be changed. If this process does not have appropriate privileges, then this must be the process' current real or saved set-UID.

All other fields of the parameter list are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE X'029C' are unpredictable.

Output: See return codes.

Return Codes for Set User IDs for the Active Process:

SPXEOK (0)
Effective UID has been changed. Real and saved set-UIDs have not been changed.

SPXEOKAL (10)
Real, effective, and saved set-UIDs have been changed (not presented unless SPXALLID is specified).

SPXECODE (1)
Incorrect value in SPXDIAGC.

SPXEFUNC (2)
Invalid function code in SPXFCODE.

SPXELEN (3)
Length in SPXDWLEN is below the minimum allowed.

SPXEFLAG (4)
Invalid option flags in SPXFLAGS or SPXRSVD1 field is nonzero.

SPXEPXCA (11)
POSIX communication area has not been defined.

SPXEAPID (12)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

SPXEID (8)
External Security Manager (ESM) has rejected the specified UID as invalid.

SPXEAUTH (5)
Active process not authorized to change to specified UID.

SPXENFND (6)
Specified user ID or user name not found in user database.

SPXEDBAS (9)
User database is invalid or inaccessible.

See Program Exceptions and Machine Checks for possible guest interruptions.

Note: When an External Security Manager (ESM) is not installed or defers the decision to CP, CP will consider a process to have appropriate privileges to change all IDs (SPXALLID) and to set IDs to a value other than the real or saved set-ID if and only if the process' effective UID is 0.

Function SPXFGRP - Set Group IDs (GIDs) for the Active Process

This function sets the effective group ID (GID) for the active POSIX process to a given group ID. For an authorized process, this function optionally sets the real and saved set-group-IDs for the process to that same group ID as well.

Input:

SPXDIAGC
Contains the hexadecimal DIAGNOSE code, SPXDIAGV (value X'029C').

SPXFCODE
Contains the function code, SPXFGRP (value 1).

SPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 3.

SPXFLAGS
Contains flags as follows:

SPXALLID
(bit 1) If this bit is on and the active process has appropriate privileges, set all three GIDs (real, effective, and saved set-ID). If this bit is off, only the effective GID is changed.

If the process does not have appropriate privileges, this bit is ignored. Only the effective GID is changed.

SPXNAMIN
(bit 2) If this bit is on, set the GID(s) to the GID associated with the group name specified in SPXGNAME.

SPXIDIN
(bit 3) If this bit is on, set the GID(s) to the value specified in SPXGID.

Exactly one of SPXNAMIN and SPXIDIN must be set. All other flag bits not listed must be binary zeros.

SPXRSVD1
Reserved for IBM use; must contain binary zeros.

SPXGID
Value to which this process' GID(s) are to be changed when SPXNAMIN is on. If this process does not have appropriate privileges, then this must be the process' current real or saved set-GID. This field is ignored when SPXNAMIN is off.

SPXGNAME
Group name to whose GID this process' GID(s) are to be changed, when SPXNAMIN is on. If this process does not have appropriate privileges, then this name must correspond to the process' current real or saved set-GID. This field is ignored when SPXNAMIN is off.

All other fields of the parameter list are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE X'029C' are unpredictable.

Output: See return codes.

Return Codes for Set Group IDs for the Active Process:

SPXEOK (0)
Effective GID has been changed. Real and saved set-GIDs have not been changed.

SPXEOKAL (10)
Real, effective, and saved set-GIDs have been changed (not presented unless SPXALLID is specified).

SPXECODE (1)
Incorrect value in SPXDIAGC.

SPXEFUNC (2)
Invalid function code in SPXFCODE.

SPXELEN (3)
Length in SPXDWLEN is below the minimum allowed.

SPXEFLAG (4)
Invalid option flags in SPXFLAGS or SPXRSVD1 field is nonzero.

SPXEPXCA (11)
POSIX communication area has not been defined.

SPXEAPID (12)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

SPXEID (8)
External Security Manager (ESM) has rejected the specified GID as invalid.

SPXEAUTH (5)
Active process not authorized to change to specified group.

SPXENFND(6)
Specified group ID or group name not found in group database.

SPXEDBAS (9)
Group database is invalid or inaccessible

See Program Exceptions and Machine Checks for possible guest interruptions.

Notes:

1. When an External Security Manager (ESM) is not installed or defers the decision to CP, CP will consider a process to have appropriate privileges to change all IDs (SPXALLID) and to set IDs to a value other than the real or saved set-ID if and only if the process' effective user ID (UID) is 0.
2. This function does not change any supplementary group IDs of the process.

Function SPXFNGRP — Change to a New Group

This function changes the real, effective and saved set-group-IDs (GIDs) for the active POSIX process to a specified GID or to the GID associated with a specified group name, or resets them to their database value. If a GID or group name is specified, the requesting virtual configuration's login name must be a member of the specified group.

Input:

SPXDIAGC
Contains the hexadecimal DIAGNOSE code, SPXDIAGV (value X'029C').

SPXFCODE
Contains the function code, SPXFNGRP (value 2).

SPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 3.

SPXFLAGS
Contains flags as follows:

SPXNAMIN
(bit 2) If this bit is on, set the real, effective and saved set-GIDs to the GID associated with the group name specified in SPXGNAME.

SPXIDIN
(bit 3) If this bit is on, set the real, effective and saved set-GIDs to the value specified in SPXGID.

All other flag bits must be binary zeros.

It is not valid to set both SPXIDIN and SPXNAMIN on. If both SPXIDIN and SPXNAMIN are off, then the real, effective and saved-set GIDs will be reset to the GID assigned to the user in the user's database entry.

SPXRSVD1
Reserved for IBM use; must contain binary zeros.

SPXGID
Value to which this process' GIDs are to be changed when SPXIDIN is on. The virtual configuration's user name must be a member of a group with which this GID is associated. If more than one group is assigned to this GID, it is unpredictable which of them is considered to be associated with the GID. This field is ignored when SPXIDIN is off.

SPXGNAME
Group name to whose GID this process' GIDs are to be changed, when SPXNAMIN is on. The virtual configuration's login name must be a member of this group. This field is ignored when SPXNAMIN is off.

All other fields of the parameter list are reserved and should contain binary zeros. If they do not, the results of a DIAGNOSE X'029C' are unpredictable.

Output:

When return code is SPXEOK:

SPXGID is set to the new value of the real, effective, and save-set GIDs. (This is the unchanged input value if SPXIDIN was on, or the GID corresponding to SPXGNAME if SPXNAMIN was on, or the database value if both flags were off.)

Return Codes for Change to a New Group:

SPXEOK (0)
Real, effective and saved set-GIDs have been set as requested. SPXGID contains the GID value to which they were set.

SPXECODE (1)
Incorrect value in SPXDIAGC.

SPXEFUNC (2)
Invalid function code in SPXFCODE.

SPXELEN (3)
Length in SPXDWLEN is below the minimum allowed.

SPXEFLAG (4)
Invalid option flags in SPXFLAGS or SPXRSVD1 field is nonzero.

SPXEPXCA (11)
POSIX communication area has not been defined.

SPXEAPID (12)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

SPXEID (8)
External Security Manager (ESM) has rejected the specified GID as invalid.

SPXENFND (6)
Specified group ID or group name not found in group database.

SPXEAUTH (5)
Requesting login name is not a member of the specified group.

SPXEDBAS
Group database is invalid or inaccessible.

See Program Exceptions and Machine Checks for possible guest interruptions.

Notes:

1. If a user belongs to more than {NGROUPS_MAX} groups, or the database has changed since the process was created, or an External Security Manager (ESM) authorizes the request, then function SPXFNGRP may be issued specifying a group which is not in the process' supplementary GID list. If CP performs the authorization checks, the only requirement is that the user be a member of the specified group. An ESM may have additional or different authorization rules.
2. This function does not change any supplementary group IDs of the process.

Function SPXFSGID — Change the supplementary group ID list

This function changes supplementary group ID list (SGID list) for the active POSIX process to the list specified. This function is used in conjunction with function codes 0 and 1 or 2 to change the identity of the active process to that of a particular user. The active process must have appropriate privileges to use this function.

Input:

SPXDIAGC
Contains the hexadecimal DIAGNOSE code, SPXDIAGV (value X'029C').

SPXFCODE
Contains the function code, SPXFSGID (value 3).

SPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 3.

SPXFLAGS
Must contain binary zeros.

SPXRSVD1
Reserved for IBM use; must contain binary zeros.

SPXSGCNT
Contains the size of the SGID-list area in units of 4 bytes, or zero if the SGID list is to be cleared. Note: the SGID list for the active process will contain only the effective GID if the count is specified as zero. To return the SGID list to its database values, the DIAGNOSE issuer must request that information from the database via DIAGNOSE code X'2A0' and then explicitly set the values via this function code.

SPXSGAL
When SPXSGCNT is not zero, contains the ALET for the address space in which the SGID-list area resides. This field is used only by XC virtual machines in access-register mode.

SPXSGAD
When SPXSGCNT is not zero, contains the address of the SGID-list area. It is unpredictable whether this address is treated as guest real or guest absolute.

All other fields of the parameter list are reserved and should contain binary zeros. If they do not, the results of a DIAGNOSE code X'029C' are unpredictable.

Output:

When return code is SPXEOK:

The active processes's SGID list is set to that specified on this diagnose.

Return Codes for Change the Supplementary Group ID List:

SPXEOK (0)
The active processes's supplementary group ID list has been changed as requested.

SPXECODE (1)
Incorrect value in SPXDIAGC.

SPXEFUNC (2)
Invalid function code in SPXFCODE.

SPXELEN (3)
Length in SPXDWLEN is below the minimum allowed.

SPXEFLAG (4)
Invalid option flags in SPXFLAGS or SPXRSVD1 field is nonzero.

SPXEAUTH (5)
The user is not authorized to perform this function.

SPXEPXCA (11)
POSIX communication area has not been defined.

SPXEAPID (12)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

SPXERRCP (13)
Active PID in POSIX communication area is not a PID allocated

See Program Exceptions and Machine Checks for possible guest interruptions.

Responses

Program Exceptions and Machine Checks: One of the following Program Exceptions or Machine Checks may be reflected to the issuing virtual machine indicating guest or host error conditions. In all cases, no meaningful return code is given, and the guest instruction (Diagnose) is nullified, suppressed, or terminated according to the architecture.

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.
Specification exception	Any of the following: • Rx is the same register as Ry. • The address of the parameter list (SPXBK) specified in Rx is not on a doubleword boundary.
Access exception (See “Access Exceptions” on page 8.)	Attempt to fetch from or store into the SPXBK failed. Regardless of the value of SPXDWLEN, an access exception may be recognized for a minimum of one doubleword at the SPXBK address. Conversely, an access exception may not be recognized for the portion of the operand beyond the length supported by CP for the function requested.
Storage-error machine check	A real storage or paging error was encountered.
Processing-damage machine check	A CP internal logic error occurred. (A CP abend usually accompanies this result.)

DIAGNOSE Code X'2A0' – Query POSIX IDs

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'2A0' specifies one of several function codes that designate Query-POSIX-ID services. DIAGNOSE code X'2A0' uses the parameter list (QPXBK) to receive input data and return output data. Optionally the services return information to another guest-specified data area.

Any virtual machine can use this DIAGNOSE. However functions which interrogate information about other users or groups may require permission from an External Security Manager (ESM), if any, or authorization in the CP directory or system configuration definition.

The following Query-POSIX-ID data functions can be invoked using this DIAGNOSE code:

- Query process attributes — function code QPXFPROC (value 0)
- Query the user database — function code QPXFUSER (value 1)
- Query the group database — function code QPXFGRP (value 2)
- Query the supplementary group IDs (SGIDs) for the active process or for a given user name — function code QPXFSGID (value 3)
- Query POSIX configuration information — function code QPXFCONF (value 4)

Entry Values:

Rx
The real address of a function parameter list (QPXBK) in guest storage. The QPXBK must be on a doubleword boundary. The format of the parameter list is determined by the function code. The QPXBK contains all the input parameters.

Ax
Is used only for XC virtual machines in access-register mode, in which case it contains the ALET for the address space containing the parameter list. If Rx designates general register 0, if Ax contains X'00000000', or if the virtual machine is not in XC mode, the parameter list is in the host-primary address space.

Active Process ID
The active process is specified in the active process ID (PID) field in the POSIX communication area. (Before DIAGNOSE code X'2A0' is used, DIAGNOSE code X'2A4' must have been issued in this virtual configuration to define this area and to allocate one or more PIDs.)

Exit Values:

Rx
The real address of the function parameter list supplied as input. If sufficient length (QPXDWLEN) is specified, the parameter list output area contains the values of the DIAGNOSE function. The format of this output parameter list is function dependent.

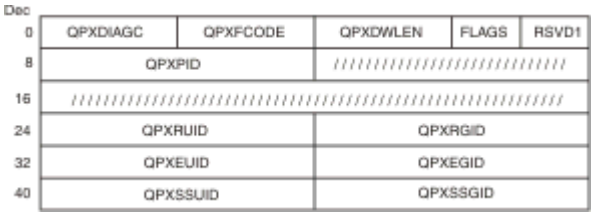
Ry
On return, contains a return code. Ry can not be register 15 and can not be the same as Rx.

Ry+1
On return, contains additional information based on the function requested. Ry+1 can not be same as Rx.

Function QPXFPROC - Query Process Attributes

This function returns the real, effective, and saved set IDs of the active POSIX process or of a specified process within the requesting virtual configuration (specified by a process ID (PID)).

Format of the HCPQPXBK for Query Process Attributes:



Input:

QPXDIAGC
Contains the hexadecimal DIAGNOSE code, QPXDIAGV (value X'02A0').

QPXFCODE
Contains the function code, QPXFPROC (value 0).

QPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 6.

QPXFLAGS
Contains flags as follows:

QPXIDIN
(bit 0) If this bit is on then report on the process whose PID is specified in QPXPID. If it is off then report on the active process.

All other flag bits must be binary zeros.

QPXRSVD1
Reserved for IBM use; must contain binary zeros.

QPXPID
Contains the process ID (PID) of the target process. This PID must have been allocated to the requesting virtual configuration (with DIAGNOSE code X'2A4'). QPXPID is used only if QPXIDIN is on.

All fields of the parameter list not listed by name are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE code X'2A0' are unpredictable.

Output: When the return code is QPXEOK, the following information is returned:

QPXRUID
Set to the real user ID of the target process.

QPXRGID
Set to the real group ID of the target process.

QPXEUID
Set to the effective user ID of the target process.

QPXEGID
Set to the effective group ID of the target process.

QPXSSUID
Set to the saved set-user-ID of the target process.

QPXSSGID
Set to the saved set-group-ID of the target process.

Ry+1
Set to the meaningful length of the QPXBK in bytes (the offset just beyond the last output field in).

Return Codes for Query Process Attributes:

QPXEOK (0)
Output fields have been set.

QPXECODE (1)
Incorrect value in QPXDIAGC.

QPXEFUNC (2)
Invalid function code in QPXFCODE.

QPXELEN (3)
Length in QPXDWLEN is below the minimum allowed.

QPXEFLAG (4)
Invalid option flags in QPXFLAGS or QPXRSVD1 field is nonzero.

QPXEPXCA (10)
POSIX communication area has not been defined.

QPXEAPID (11)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

QPXENFND (6)
Process id QPXPID was not found. (The PID does not exist or was not allowed to the requesting virtual configuration.)

See Program Exceptions and Machine Checks for possible guest interruptions.

Function QPXFUSER - Query the User Database

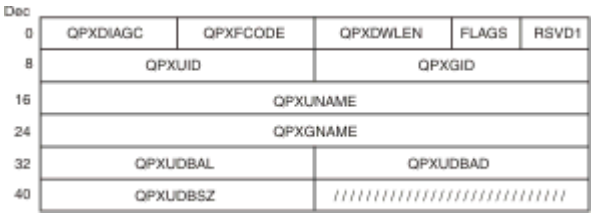
This function gets the POSIX attributes of a user. You can specify the user by POSIX user ID (UID) or user name.

To be authorized to obtain a user database entry, either:

- the ESM must grant the requestor authority to read the entry, or
- the ESM must not be installed or must defer authorization to CP, and
  - the UID in the entry must match the active process's real or effective UID, or

- the active process's effective UID must be 0, or
- the requesting VM user ID must have the attribute POSIXOPT QUERYDB ALLOW, either through a statement in its CP directory entry or through a setting, specified or defaulted, in the system configuration file, which is not overridden in the directory entry, or
- the requesting VM user ID is requesting information about themself.

Format of the HCPQPXBK for Query the User Database:



Input:

QPXDIAGC
Contains the hexadecimal DIAGNOSE code, QPXDIAGV (value X'02A0').

QPXFCODE
Contains the function code, QPXFUSER (value 1).

QPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 6.

QPXFLAGS
Contains flags as follows:

QPXIDIN
(bit 0) If this bit is on then report on the UID specified in QPXUID.

QPXNAMIN
(bit 1) If this bit is on then report on the user name specified in QPXUNAME.

If neither of these flags are set, information is returned on the current POSIX process environment. If no process environment has been established, information is returned on the default POSIX environment for the issuing user ID, or for the alternate user ID if one has been established.

QPXRSVD1
Reserved for IBM use; must contain binary zeros.

QPXUID
Contains the search argument user ID. QPXUID is used only if QPXIDIN is on.

QPXUNAME
Contains the search argument user name. QPXUNAME is used only if QPXNAMIN is on.

QPXUDBAL
When QPXUDBSZ is not zero, contains the ALET for the address space in which a user-database information area resides. This field is used only in XC virtual machine access-register mode.

QPXUDBAD
When QPXUDBSZ is not zero, contains the address of a user-database information area. It is unpredictable whether this address is treated as guest real or guest absolute.

QPXUDBSZ
Contains the size of the user-database information area in bytes, or zero if user-database information is not to be returned.

All fields of the parameter list not listed by name are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE code X'2A0' are unpredictable.

Output:

When the return code is QPXEOK, the following information is returned in the parameter list and user-database area:

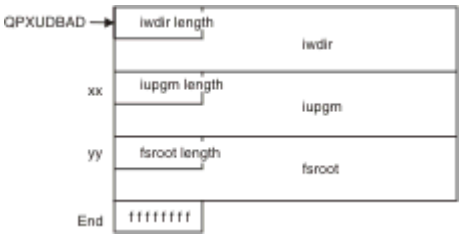
QPXUID
Set to the user ID corresponding to the given QPXUNAME when QPXNAMIN is on. Set to the user ID corresponding to the current user name when QPXIDIN and QPXNAMIN are off and no POSIX environment exists.

QPXGID
Set to the primary group ID assigned to this user.

QPXUNAME
Set to the user name corresponding to the given QPXUID when QPXIDIN is on. Set to the user name corresponding to the given QPXUNAME when QPXNAMIN is on; in this case, the input QPXUNAME and the output QPXUNAME can be different. Set to the user name corresponding to the current user's UID when QPXIDIN and QPXNAMIN are off and a POSIX environment exists.

QPXGNAME
Set to the primary group name assigned to this user.

When QPXUDBSZ is not zero, the user-database information area contains the following data in immediate succession.



iwdir length
Four-byte field containing the length of the user's initial working directory.

iwdir
Contains the user's initial working directory. It is a variable-length string of characters from 0 to 1023 bytes long.

iupgm length
Four-byte field containing the length of the user's initial user program. It immediately follows the end of IWDIR.

iupgm
Contains the user's initial user program, a variable-length string of characters from 0 to 1023 bytes long.

fsroot length
Four-byte field containing the length of the user's file system root. It immediately follows the end of the IUPGM.

fsroot
Contains the name of a user's file system root, a variable-length string of characters from 0 to 1023 bytes long.

ffffffff
Contains a four-byte value X'FFFFFFFF' as end-of-data delimiter.

These string values are not null-terminated; that is, they do not end in the null character (X'00'). String values not specified in the user database entry are returned as null strings (length zero).

When the return code is QPXEOLEN, the following information is returned:

Ry + 1
Set to the actual length required for the user-database area, in bytes.

When the return code is not QPXEOK, the contents of any portion of the user-database information area which is not subject to an access exception are unpredictable.

Return Codes for Query the User Database:

QPXEOK (0)
Output fields have been set.

QPXECODE (1)
Incorrect value in QPXDIAGC.

QPXEFUNC (2)
Invalid function code in QPXFCODE.

QPXELEN (3)
Length in QPXDWLEN is below the minimum allowed.

QPXEFLAG (4)
Invalid option flags in QPXFLAGS, or QPXRSVD1 field is nonzero.

QPXEPXCA (10)
POSIX communication area has not been defined.

QPXEAPID (11)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

QPXEAUTH (5)
Requestor is not authorized to obtain requested information.

QPXENFND (6)
Specified UID or UNAME is not found in user database.

QPXEOLEN (7)
User-database information area size specified in QPXUDBSZ is too small to contain the information. The actual length required, in bytes, is returned in register Ry+1.

QPXEOADR (8)
The ALET or address range of the user-database information area is invalid, or the designated area is protected against storing.

QPXEDBAS (9)
The user database contents are invalid or inaccessible.

See Program Exceptions and Machine Checks for possible guest interruptions.

Notes:

1. If the QPXIDIN flag is on, the information returned is from a random CP directory entry or External Security Manager (ESM) entry which contains a UID matching the specified UID. If there is more than one such entry, it is unpredictable which one is returned.
2. If the named user has no UID defined in the CP directory entry or the ESM user database, the default value 4294967295 (X'FFFFFFFF') is returned in the QPXUID field.
3. If the named user has no GID defined in the CP directory entry or the ESM user database, the default value 4294967295 (X'FFFFFFFF') is returned in the QPXGID field.
4. If both QPXIDIN and QPXNAMIN are OFF, this command can be used to query information about yourself without starting the POSIX environment.

Function QPXFGRP - Query the Group Database

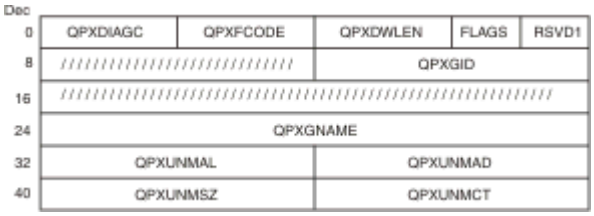
This function gets the attributes of a given group ID (GID) or POSIX group name, and optionally, a list of its members.

To be authorized to obtain a group database entry, either

- the ESM must grant the requestor authority to read the entry, or
- the ESM must not be installed or must defer authorization to CP, and
  - the active process's effective UID must be 0, or
  - the active process's real or effective GID must match the GID of the designated group, or
  - the requesting user name must be a member of the designated group, or

– the requesting VM user ID must have the attribute POSIXOPT QUERYDB ALLOW, either through a statement in its CP directory entry or through a setting, specified or defaulted, in the system configuration file, which is not overridden in the directory entry.

Format of the HCPQPXBK for Query the Group Database:



Input:

QPXDIAGC
Contains the hexadecimal DIAGNOSE code, QPXDIAGV (value X'02A0').

QPXFCODE
Contains the function code, QPXFGRP (value 2).

QPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 6.

QPXFLAGS
Contains flags as follows:

QPXIDIN
(bit 0) If on then report on the group with GID as specified in QPXGID.

QPXNAMIN
(bit 1) If on then report on the group name specified in QPXGNAME.

Exactly one of these bits must be set. All other flag bits must be binary zeros.

QPXRSVD1
Reserved for IBM use; must contain binary zeros.

QPXGID
Contains the search argument group id. QPXGID is used only if QPXIDIN is on.

If there are multiple groups associated with this GID, then it is unpredictable which of them is considered the designated group.

QPXGNAME
Contains the search argument group name. QPXGNAME is used only if QPXNAMIN is on.

QPXUNMSZ
Contains the size of the member-list area in units of 8 bytes, or zero if no member list is to be returned.

QPXUNMAL
When QPXUNMSZ is not zero, contains the ALET for the address space in which the member-list area resides. This field is used only by XC virtual machines in access-register mode.

QPXUNMAD
When QPXUNMSZ is not zero, contains the address of the member-list area. It is unpredictable whether this address is treated as guest real or guest absolute.

All fields of the parameter list not listed by name are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE code X'02A0' are unpredictable.

Output:

When the return code is QPXEOK, the following information is returned in the parameter list and member-list area:

QPXGID
Set to the group ID corresponding to the given QPXGNAME when QPXNAMIN is on, unchanged otherwise.

QPXGNAME
Set to the group name corresponding to the given QPXGID when QPXIDIN is on, unchanged otherwise.

QPXUNMCT
Set to the actual count of members in the group.

When QPXUNMSZ is not zero, an array of user names which belong to the designated group is stored in the member-list area. Each name is left-justified and blank-padded to 8 bytes.

When the return code is QPXEOLEN, the following information is returned in the parameter list:

QPXUNMCT
Set to the actual length required for the member-list area in units of 8 bytes.

When the return code is not QPXEOK, the contents of any portion of the member-list area which is not subject to an access exception is unpredictable.

Return Codes for Query the Group Database:

QPXEOK (0)
Output fields have been set.

QPXECODE (1)
Incorrect value in QPXDIAGC.

QPXEFUNC (2)
Invalid function code in QPXFCODE.

QPXELEN (3)
Length in QPXDWLEN is below the minimum allowed.

QPXEFLAG (4)
Invalid option flags in QPXFLAGS or QPXRSVD1 field is nonzero.

QPXEPXCA (10)
POSIX communication area has not been defined.

QPXEAPID (11)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

QPXEAUTH (5)
Requestor is not authorized to obtain requested information.

QPXENFND (6)
Specified group ID or group name not found in group database.

QPXEOLEN (7)
Value in QPXUNMSZ is not zero but is smaller than the number of members in the group. The member-list area is too small. Output fields have been set.

QPXEOADR (8)
The ALET or address range of the member-list area is invalid, or the designated area is protected against storing.

QPXEDBAS (9)
The group database contents are invalid or inaccessible.

See Program Exceptions and Machine Checks for possible guest interruptions.

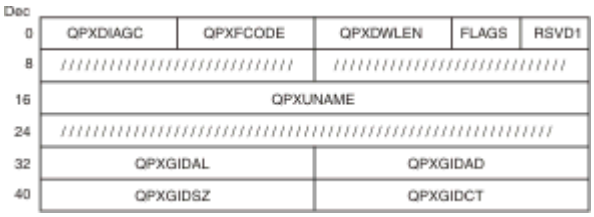
Function QPXFSGID - Query the Supplementary Group IDs

This function gets the number, and optionally the list, of supplementary group IDs (SGIDs) currently associated with the active POSIX process, or associated with the given user name in the user database.

A process is always authorized to obtain its own SGID list. To be authorized to obtain a SGID list for a user name either

- the ESM must grant the requestor authority to obtain the list, or
- the ESM must not be installed or must defer authorization to CP, and
  - the UID in the entry must match the active process's real or effective UID, or
  - the active process's effective UID must be 0, or
  - the requesting VM user ID must have the attribute POSIXOPT QUERYDB ALLOW, either through a statement in its CP directory entry or through a setting, specified or defaulted, in the system configuration file, which is not overridden in the directory entry.

Format of the HCPQPXBK for Query the Group Database:



Input:

QPXDIAGC
Contains the hexadecimal DIAGNOSE code, QPXDIAGV (X'02A0').

QPXFCODE
Contains the function code, QPXFSGID (value 3).

QPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 6.

QPXFLAGS
Contains flags as follows:

QPXNAMIN
(bit 1) If on then report the information from the user database on the user name specified in QPXUNAME. If off then report the current attributes of the active process.

All other flag bits must be binary zeros.

QPXRSVD1
Reserved for IBM use; must contain binary zeros.

QPXUNAME
Contains the search argument user name. QPXUNAME is used only if QPXNAMIN is on.

QPXGIDSZ
Contains the size of the SGID-list area in units of 4 bytes, or zero if no SGID list is to be returned.

QPXGIDAL
When QPXGIDSZ is not zero, contains the ALET for the address space in which the SGID-list area resides. This field is used only by XC virtual machines in access-register mode.

QPXGIDAD
When QPXGIDSZ is not zero, contains the address of the SGID-list area. It is unpredictable whether this address is treated as guest real or guest absolute.

All fields of the parameter list not listed by name are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE code X'02A0' are unpredictable.

Output:

When the return code is QPXEOK, the following information is returned in the parameter list and SGID-list area:

QPXGIDCT
Set to the actual count of SGIDs stored in the SGID-list area or count of SGIDs assigned if QPXGIDSZ=0.

When QPXGIDSZ is not zero, an array of 4-byte unsigned supplementary group IDs assigned to the designated process or user is stored in the SGID-list area.

When the return code is QPXEOADR, the contents of any portion of the SGID-list area which is not subject to an access exception is unpredictable.

Return Codes for Query the Supplementary Groups IDs:

QPXEOK (0)
Output fields have been set.

QPXECODE (1)
Incorrect value in QPXDIAGC.

QPXEFUNC (2)
Invalid function code in QPXFCODE.

QPXELEN (3)
Length in QPXDWLEN is below the minimum allowed.

QPXEFLAG (4)
Invalid option flags in QPXFLAGS or QPXRSVD1 field is nonzero.

QPXEPXCA (10)
POSIX communication area has not been defined.

QPXEAPID (11)
Active PID in POSIX communication area is not a PID allocated to the requesting virtual configuration.

QPXEAUTH (5)
Requestor is not authorized to obtain requested information.

QPXENFND (6)
Specified user name not found in user database. Returned only when QPXNAMIN is on.

QPXEOLEN (7)
Value in QPXGIDSZ is not zero but is too small to contain the SGID-list. The actual length required is returned in QPXGIDCT, in units of 4 bytes.

QPXEOADR (8)
The ALET or address range of the SGID-list area is invalid, or the designated area is protected against storing.

QPXEDBAS (9)
The user database contents are invalid or inaccessible. Returned only when QPXNAMIN is on.

See Program Exceptions and Machine Checks for possible guest interruptions.

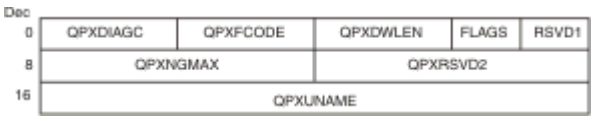
Note:

1. The list of the supplementary group IDs is only returned to the storage area pointed to by QPXGIDAD if the number in the QPXGIDSZ field is greater than or equal to the total number of supplementary group IDs for the specified user name or process.

Function QPXFCONF - Query POSIX Configuration Information

This function returns POSIX system configuration parameters and environmental information.

Format of the HCPQPXBK for Query POSIX Configuration Information:



Input:

QPXDIAGC
Contains the hexadecimal DIAGNOSE code, QPXDIAGV (value X'02A0').

QPXFCODE
Contains the function code, QPXFCONF (value 4).

QPXDWLEN
Contains the length of this parameter list in doublewords. This value must be at least 3.

QPXFLAGS
All flag bits must be binary zeros.

QPXRSVD1
Reserved for IBM use; must contain binary zeros.

All fields of the parameter list not listed by name are reserved and should contain binary zeros. If they do not, the results of DIAGNOSE code X'02A0' are unpredictable.

Output:

When the return code is QPXEOK, the following information is returned:

Ry + 1
Set to the meaningful length of the QPXBK in bytes (the offset just beyond the last output filled in).

QPXNGMAX
Set to the POSIX {NGROUPS_MAX} value supported in the current environment. This is the maximum number of entries in the supplementary group ID list for a user name or process.

QPXRSVD2
Reserved for IBM use; contents are unpredictable.

QPXUNAME
Set to the user (login) name of the requesting virtual configuration.

Return Codes for Query POSIX Configuration Information:

QPXEOK (0)
Output fields have been set.

QPXECODE (1)
Incorrect value in QPXDIAGC.

QPXEFUNC (2)
Invalid function code in QPXFCODE.

QPXELEN (3)
Length in QPXDWLEN is below the minimum allowed.

QPXEFLAG (4)
Invalid option flags in QPXFLAGS or QPXRSVD1 field is nonzero.

See Program Exceptions and Machine Checks for possible guest interruptions.

Usage Note

Certain functions and options of DIAGNOSE code X'2A0' report POSIX database contents rather than attributes of the active process. For these that report database contents, the active PID may still be used by an ESM for authority checking. Therefore, it is unpredictable whether or not the requirements to define a POSIX communication area and to supply a valid active PID are enforced for these functions.

Responses

Program Exceptions and Machine Checks: One of the following Program Exceptions or Machine Checks may be reflected to the issuing virtual machine indicating guest or host error conditions. In all cases, no meaningful return code is given, and the guest instruction (Diagnose) is nullified, suppressed, or terminated according to the architecture.

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.

Problem Encountered	Cause
Specification exception	Any of the following: • Rx is the same register as Ry or Ry+1. • Ry is register 15. • The address of the parameter list (QPXBK) specified in Rx is not on a doubleword boundary.
Access exception (See “Access Exceptions” on page 8.)	Attempt to fetch from or store into the QPXBK failed. Regardless of the value of QPXDWLEN, an access exception may be recognized for a minimum of one doubleword at the QPXBK address. Conversely, an access exception may not be recognized for the portion of the operand beyond the length supported by CP for the function requested.
Storage-error machine check	A real storage or paging error was encountered.
Processing-damage machine check	A CP internal logic error occurred. (A CP abend usually accompanies this result.)

DIAGNOSE Code X'2A4' – POSIX Process ID (PID) Services

Privilege Class: Any

Addressing Mode: 24-bit or 31-bit

DIAGNOSE code X'2A4' supports several function codes that designate various POSIX process ID services. They assist in the creation, deletion, identification and execution of POSIX processes in the virtual configuration.

Entry Values:

Rx
contains a function code specifying which function DIAGNOSE code X'2A4' is to perform. Rx can not be register 15 and can not be the same as Ry or Ry+1. Additional inputs for each function are described below.

The following POSIX process ID services can be invoked using this DIAGNOSE code:

- Function 0 - Identify the POSIX communication area
- Function 1 - Allocate a PID
- Function 2 - Deallocate a PID

Ry + 1
may contain additional information based upon the specific function being performed.

Exit Values:

Ry
on return, contains a return code. Ry can not be register 15 and can not be the same as Rx or Rx+1. The return codes and additional outputs for each function are described below.

Rx + 1
may contain additional information based upon the specific function being performed.

Function 0 - Identify the POSIX communication area

This function identifies the invoking virtual machine's POSIX communication area to CP.

Input:

Rx+1
specifies the length and location of the virtual machine's POSIX communication area. The POSIX communication area must reside entirely in the prefix area, above location 511 (decimal), and it must be at least 4 bytes in length. Bytes 0 and 1 of Rx+1 contain the length, in bytes, of the POSIX communication area. Bytes 2 and 3 contain the displacement into the prefix area of the first byte of the POSIX communication area. Key-controlled protection and low-address protection do not apply to accesses to this area.

The virtual subsystem reset operation causes the length and location of the POSIX communication area to become undefined to CP. This operation is performed during the system-reset-clear or system-reset-normal function performed by commands such as IPL, SYSTEM CLEAR, SYSTEM RESET and DETACH CPU.

The POSIX communication area has the following format:

Offset

0-3
is used by the virtual machine to identify to CP the "active" POSIX process executing on a virtual CPU. This process is identified by its PID. A value of X'00000000' indicates to CP that there is no currently active POSIX process executing on the virtual CPU.

Return Codes for Identify the POSIX communication area:

0
The POSIX communication area has been successfully identified to CP

4
Rx+1 contains an invalid length and/or address

See Program Exceptions for possible guest interruptions.

Notes:

1. This area may be referenced during the execution of other DIAGNOSES such as X'280', X'29C' and X'2A0'.
2. In a virtual machine with multiple virtual CPUs, the program controlling the virtual machine need only invoke this function once, because it identifies the displacement into each virtual CPU's prefix area. These programs normally use prefixing to identify a separate prefix area for each virtual CPU. This results in a separate POSIX communication area for each virtual CPU.

Function 1 - Allocate a PID

This function allocates a PID for use by a new POSIX process and initializes the new process' POSIX IDs. The POSIX IDs are the real UID, effective UID, saved set-UID, real GID, effective GID, saved set-GID and the supplementary GIDs.

Input:

Rx+1
contains the PID of the newly created process' parent process, or X'00000000'. If Rx+1 contains a valid, non-zero PID, the new process' POSIX IDs are initialized to those of the identified process. If Rx +1 contains X'00000000', the new process' POSIX IDs are initialized to the POSIX database values that were in effect when the invoker logged on; the most recent supplementary GID list from the ESM will be in effect for the new process.

Output:

Ry+1
upon successful completion, contains the newly allocated PID.

Return Codes for Allocate a PID:

0
A PID has been successfully allocated, and the new process' POSIX IDs have been initialized as described above. The new PID is returned to the invoker in Ry+1.

4
Rx+1 contains an invalid PID. The PID may be invalid for any of the following reasons:

- It is outside the valid range
- It is not currently allocated
- It is not allocated to the invoking virtual configuration

8
The system has no available PIDs to allocate

12
Allocating a new PID would exceed the invoking virtual configuration's PID limit

See Program Exceptions for possible guest interruptions.

Notes:

1. When the process associated with the PID is deleted, the virtual configuration should issue the Deallocate a PID function so CP will deallocate the PID and make it available for later use by the system.
2. The virtual subsystem reset operation causes all PIDs currently allocated to the virtual configuration to be deallocated. This operation is performed during the system-reset-clear or system-reset-normal function performed by commands such as IPL, SYSTEM CLEAR, SYSTEM RESET and DETACH CPU.

Function 2 - Deallocate a PID

This function deallocates a PID and makes it available for later use by the system.

Input:

Rx+1
contains the PID to be deallocated. This PID must be currently allocated to the invoking virtual configuration. It must have been allocated by a successful invocation of the Allocate a PID function.

Return Codes for Deallocate a PID:

0
The specified PID has been deallocated.

4
Rx+1 contains an invalid PID. The PID may be invalid for any of the following reasons:

- It is outside the valid range
- It is not currently allocated
- It is not allocated to the invoking virtual configuration

See Program Exceptions for possible guest interruptions.

Responses

Program Exceptions: One of the following Program Exceptions may be reflected to the issuing virtual machine indicating guest or host error conditions. In all cases, no meaningful return code is given, and the guest instruction (Diagnose) is nullified, suppressed, or terminated according to the architecture.

Problem Encountered	Cause
Privileged-operation exception	The virtual machine is in the problem state.

Problem Encountered	Cause
Specification exception	Any of the following: • An invalid function code is specified. • Rx is register 15. • Ry is register 15. • Rx is the same register as Ry or Ry+1. • Ry is the same register as Rx+1.

DIAGNOSE Code X'2AC' – HCD Dynamic I/O

Privilege Class: B

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'2AC' to enable the HCD service virtual machine to query I/O configuration information stored in CP (ICIBK) and to execute dynamic changes to CP's I/O configuration.

Function Codes:

X'00'
Query I/O Configuration Information

X'01'
Perform Dynamic I/O Changes

Entry Values (Function 00 and 01):

Rx
is a general register that contains the desired function code

Ry
is a general register that contains the guest real address of a buffer to store the I/O configuration information (function 0) or the guest real address of the configuration control block (CCB) that describes the dynamic I/O change requests (function 1).

Note: All addresses passed on DIAGNOSE code X'2AC' are guest absolute addresses in the host-primary address space.

Exit Values (Function 01 Only):

Rx
contains a return code indicating the result of the request. Refer to “Responses” on page 1076 for a description of the possible values.

Ry
when Rx is non-zero, Ry may contain a value that helps further describe the error. Refer to “Responses” on page 1076 for a description of the possible values.

The entire ARSPL may be replaced in guest storage upon completion of a DIAGNOSE code X'254' function.

Responses

Condition Codes and Return Codes: Upon completion of DIAGNOSE code X'2AC', control is returned to the issuer with a condition code and, when applicable, a return code set to indicate the status of the requested function.

Table 231. Condition Codes for Query I/O Configuration Information (Function 0)

Condition Code	Return Code in Rx	Description
0	N/A	I/O configuration information was successfully stored in the guest's buffer
3	N/A	I/O configuration information is unavailable; HCD is not controlling CP's I/O configuration

Table 232. Condition Codes and Return Codes for Perform Dynamic I/O Changes (Function 1)

Condition Code	Return Code in Rx	Description
0	N/A	All change requests were successfully executed
1	4 (X'04')	The specified IOCDS could not be switched to make it the active IOCDS. However, all other change requests (if any), were successfully executed. The guest Ry is set to a hardware response code indicating why the IOCDS file could not be made active. For more information about the hardware response code see *z/VM: I/O Configuration*.
1	8 (X'08')	None of the change requests were processed because pre-processing checks with one or more of the requests failed. No backout is required. The guest Ry is set to the number of requests that failed pre-processing. Each request (CCB entry) contains a return code indicating its pre-processing status. • RC0 - Request passed pre-processing • RC2 - CHPID type is unknown • RC4 - CHPID specified is a managed CHPID • RC8 - CHPID specified by or associated with the change was online • RC12 - CHPID is physically available • RC20 - FORCE was not specified • RC28 - Device is not subchannel disabled • RC32 - Device is a PAV base with a PAV alias still associated with it • RC36 - RDEV is not offline • RC40 - RDEV is offline but attached (boxed) • RC44 - RDEV class mismatch • RC48 - RDEV is an HCPRIO RDEV • RC56 - RDEV has an active I/O operation associated with it • RC60 - Unsupported request • RC76 - PCI Function is not offline
1	12 (X'0C')	None of the change requests were processed because it was determined that there was not enough Hardware System Area (HSA) storage to carry out the requested hardware changes. No backout is required. The guest's Ry is set to zero if this is the only error. If pre-processing checks with one or more of the individual requests failed also, then the guest's Ry is set to the number of requests that failed pre-processing. Each request (CCB entry) contains a return code (listed above) indicating its pre-processing status.

Table 232. Condition Codes and Return Codes for Perform Dynamic I/O Changes (Function 1) (continued)

Condition Code	Return Code in Rx	Description
1	16 (X'10')	All of the change requests passed pre-processing checks but none of them were actually processed due to the fact that CP could not cause the processor to enter configuration mode to perform dynamic hardware changes. No backout is required. The guest's Ry contains the hardware response code indicating why CP could not enter configuration mode. For more information about the hardware response code see *z/VM: I/O Configuration*.
1	20 (X'14')	An error was encountered during the processing of a dynamic change request. Backout is required! The guest's Ry contains the CCB entry index number of the request that failed. The request (CCB entry) that failed contains a non-zero return code (and possibly a hardware response code — for more information about the hardware response code see *z/VM: I/O Configuration*). All previous requests were processed successfully (RC0) while all subsequent requests were not executed. • RC64 - Hardware command failed (possibly accompanied by a hardware response code). • RC68 - Backout is required; backout is requested from this point • RC72 - Software command failed
1	24 (X'18')	Recovery information could not be retrieved.
1	92 (X'5C')	The data area size specified in the CCB header or CCBX header was zero or negative.
1	96 (X'60')	The dynamic I/O change request was too large for CP to process.
2	N/A	A dynamic I/O change request is currently being processed
3	N/A	I/O configuration information is unavailable; HCD is not controlling the I/O configuration

Program Exceptions: DIAGNOSE code X'2AC' may result in one of the following program exceptions:

Problem Encountered	Cause
Addressing exception	The guest real address specified in Ry is not within addressable guest real storage.
Protection exception	The guest real address specified in Ry is store or fetch protected.

Problem Encountered	Cause
Specification exception	Any of the following: • The function code value in Rx is invalid. • No guest real address was specified in Ry (Ry = 0). • The guest real address specified in Ry is not on a doubleword boundary. • The guest real address specified in Ry did not point to a CCB. • The guest does not have the privilege class necessary to issue this DIAGNOSE code.

DIAGNOSE Code X'2C0' – HMC Data Source Load

Privilege Class: A

Addressing Mode: 24-bit or 31-bit

Use DIAGNOSE code X'2C0' to load binary image files from z/Architecture Hardware Management Console's data sources. Using DIAGNOSE code X'2C0' ensures that CP will construct an appropriate Service Call instruction.

Results of the I/O operation are contained in the R15 register and the storage buffer pointed to by Ry before the instruction's execution will be overlaid with the data retrieved from the SCLP device.

Issuing DIAGNOSE code X'2C0' with Ry equal to zero will cause the system to return the size of the CP Buffer in R15. The size of the CP Buffer is fixed and may not be changed.

With Ry nonzero, the DIAGNOSE operates by first allocating a permanent CP internal buffer and clearing it to X'FF's. Next, the file is loaded from one of two sources, depending on the operating environment of the system: If z/VM is operating as a first level system within an LPAR, then a Service Call (Type 7) instruction is executed and the *filename* file is loaded into CP's buffer from the Hardware Console's DVD disk drive or FTP source from which the LPAR was IPLed. If z/VM is operating as a guest under z/VM, then the file "*filename* IMAGE *" is loaded from a CPACCESSed minidisk.

The operation of the DIAGNOSE fills the CP buffer with X'FF' bytes and then reads the requested file into the start of the buffer. If, for example, the buffer is 4MB in length and the file that is loaded is 800K long, then the buffer from 0K to 800K contains the file and from 800K to 4096K contains X'FF's. The entire 4MB buffer is then copied into the virtual machine's buffer area specified in Ry.

Note: If the requested file exceeds the length of the CP buffer, then host real storage will be overlaid and the system's integrity will be compromised. In the event of an overlay the z/VM operating system will probably abend with an LDF001 hard abend.

Entry Values:

Rx, Rx+1
Contains the eight-byte EBCDIC name of the file to be loaded. When running under the z/VM operating system, the file "*filename* IMAGE *" is loaded from a CPACCESS'ed minidisk.

Ry
Two different operations may be performed depending upon the value in Ry.

A value of 0 indicates that the size of the CP defined buffer is to be returned. This is the amount of data that the X'2C0' DIAGNOSE will return, regardless of the file's size, which is 4MB.

A nonzero value is the address of the buffer in the guest virtual machine where the data is to be placed. This buffer must be the size that is reported by issuing an Ry=0 call and the buffer must be page-aligned.

Example:

```
SR       Ry,Ry
DIAG  0,Ry,X'2C0'
ST       R15,BUFSIZE

CMSSTOR OBTAIN,BYTES=(R15),BNDRY=PAGE
ST       R1,BUFFERAD

LM       Rx,Rx+1,=CL8'filename'
L        Ry,BUFFERAD
DIAG     Rx,Ry,X'2C0'
LTR      R15,R15
BNZ      ERROR
...
```

Return Values:

A return code is provided in R15.

The Condition Code is unspecified.

When called with Ry=0, then the returned value in R15 is the size of the required virtual buffer.

When Ry specifies the virtual buffer address, the results of the data operation are returned in R15.

Return Code	Meaning
0 (X'00')	The buffer has been loaded with the specified file.
1 (X'01')	The virtual buffer pointed to by Ry is not page aligned.
2 (X'02')	The SCLP interface is currently is use. Retry at a later time.
4 (X'04')	The SCLP returned a "Not Normal" indication.
5 (X'05')	The SCLP returned a "Not Complete" indication.
6 (X'06')	Invalid virtual address supplied in Ry.
7 (X'07')	Invalid i-ASIT for address in Ry.
8 (X'08')	Paging subsystem error.
1000–1999	SCLP start request failed. Value is HCPPCRRQ return code + 1000.
2000–2999	SCLP completion error. Value is xxx + 2000. (NOTE: 2003 is SCLP File Load Error.)
3000–3999	Virtual page address translation error. Value is HCPTRANS RC + 3000.
4000–4999	z/VM HCPCLVLH failed. Value is HCPCLVLH return + 4000. (NOTE: 4059 is File Not Found.)

DIAGNOSE Code X'2C4' – FTP Services

Privilege Class: B (See notes below)

Addressing Mode: 24-bit, 31-bit or 64-bit

DIAGNOSE X'2C4' provides FTP services for files residing on a removable medium in an HMC device.

Notes:

1. A class Any user may issue DIAGNOSE X'2C4' if the user's directory includes the OPTION LXAPP statement, and if the user specifies a location indicator (any of bits 32-39 turned on).
2. If the user's directory does *not* include the OPTION LXAPP statement, then that user must be class B.

Entry Values:

Rx
The general register containing the guest absolute address of the FTP services parameter list (FPL). The FPL must be on a doubleword boundary.

Ry
The general register containing the FTP function code and the location indicator. The function code is in bits 56-63. The possible function codes are:

X'00'
NOOP

X'01'
GET

X'02'
PUT

X'03'
APPEND

X'04'
DIR

X'05'
NLST

X'06'
DELETE

The location indicator is in bits 32-39 and all values are reserved for IBM use.

Exit Values:

Ry
On return, contains a return code indicating the result of the request. See "Responses" on page 1082 for a description of the possible return codes.

FPL

The parameter list consists of an input area, an output area, and a file identifier. The area must be aligned on a doubleword boundary. The FPL format is as follows:

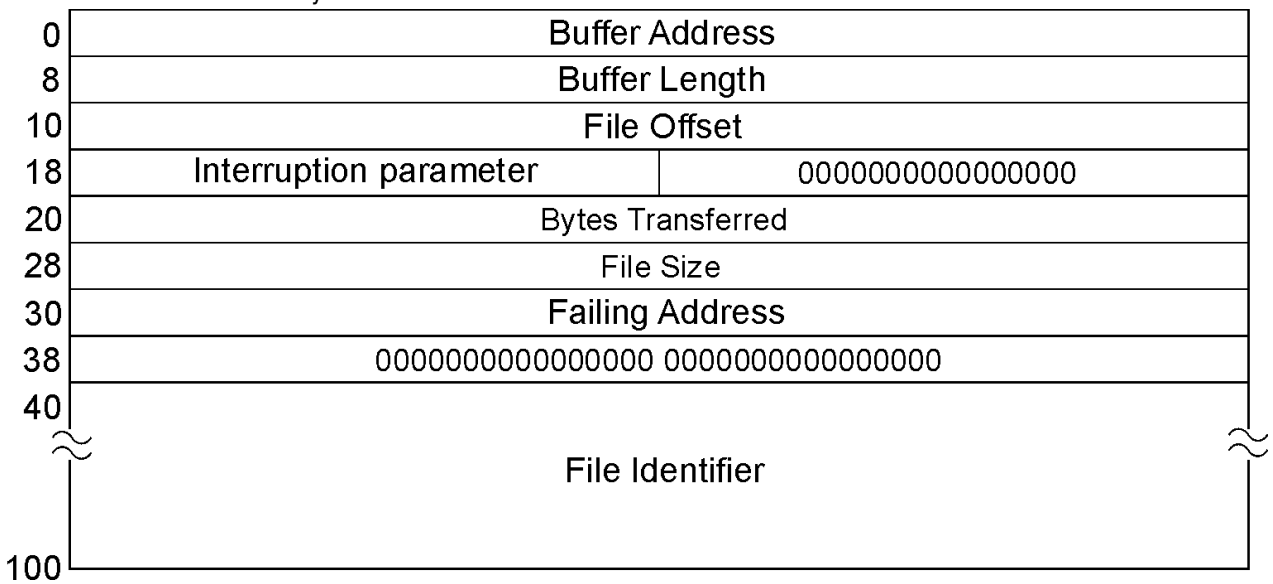


Figure 111. DIAGNOSE X'2C4' FPL Parameter List Format

Buffer Address
Is a doubleword containing the guest real address of the buffer to be used for the FTP function. The buffer must be aligned on a 4K page boundary.

Buffer Length
Is a doubleword containing the length of the buffer to be used for the FTP function.

File Offset
Is a doubleword containing the offset in the file to be used for the FTP function.

Interruption Parameter
Is a fullword field containing user data to be stored at guest real storage location 128-131 in the host-primary address space upon presentation of the FTP services external interruption, at the completion of the FTP services request.

Bytes Transferred
Is a doubleword that returns the number of bytes transferred by the FTP function, at the completion of the FTP services request.

File Size
Is a doubleword that returns the size in bytes of the file being processed by the FTP function, at the completion of the FTP services request.

Failing Address
Is a doubleword that returns the address in the buffer associated with a failure of the FTP function, at the completion of the FTP services request.

File Identifier
Is a 192-byte field containing the file identifier on which the FTP function is to operate, followed by a X'00' to mark the end of the identifier.

Responses

Upon completion, DIAGNOSE X'2C4' sets one of the following return codes in Ry:

Table 233. DIAGNOSE Code X'2C4' Return Codes in Ry

Return Code in Register Ry	Meaning
0 (X'00')	FTP function initiated.
4 (X'04')	FTP interface in use.
8 (X'08')	Interface error.

Program Exceptions: These program exceptions can occur if DIAGNOSE X'2C4' is given incorrect data:

Problem Encountered	Cause
Privileged-operation exception	The issuer of DIAGNOSE X'2C4' does not have the appropriate privilege class or directory authorization.
Specification exception	Any of the following: • The address contained in Rx is not on a doubleword boundary. • The value contained in Ry is not in range. • The buffer is not on a page boundary. • The buffer length is not in range. • The buffer is not addressable. • The buffer is not read/write.

FTP Services External Interruption: An FTP services external interruption is presented when a DIAGNOSE X'2C4' request has completed. The interruption is a floating condition and is presented to the first virtual CPU in the virtual configuration that is enabled for the interruption subclass. The interruption condition is cleared once the interruption has been presented, as well as by a virtual subsystem reset (for example, a SYSTEM RESET or IPL command).

The subclass mask to enable for the interruption is bit 22 of control register 0.

The FTP services condition is indicated by an external-interruption code of X'2603' stored at guest real location 134-135 and a sub-interruption code of X'08' stored at guest real location 132. The interruption parameter associated with the original DIAGNOSE X'2C4' request is stored at guest real locations 128-131. In addition, one of the following status codes will be stored at guest real location 133:

X'00'
Request completed successfully.

X'04'
Program check condition detected when storing results in original request buffer.

X'08'
Paging I/O error encountered when storing results in original request buffer.

X'0C'
Request was cancelled due to time out condition sensed by z/VM. The function can now be retried by the guest.

X'10 + error code'
Request completed with error.

Note: All locations updated as a result of the external interruption are in the host-primary address space.

The Bytes Transferred, File Size, and Failing Address fields in the original FPL are filled in when an FTP services external interruption is made pending. In the case of a X'0C' status code, these fields are not returned to the guest.

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

© Copyright IBM Corp. 1991, 2020

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 information primarily documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/VM.

This information also documents information that is NOT intended to be used as Programming Interfaces of z/VM. This information is identified where it occurs by an introductory statement to a chapter or section.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at IBM copyright and trademark information - United States (www.ibm.com/legal/us/en/copytrade.shtml).

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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:

- IBM Privacy Statement at http://www.ibm.com/privacy/us/en/
- IBM Online Privacy Statement at http://www.ibm.com/privacy/details/us/en/ in the section entitled "Cookies, Web Beacons and Other 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 Knowledge Center - z/VM (www.ibm.com/support/knowledgecenter/SSB27U).

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
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- z/OS and z/VM: Hardware Configuration Manager User's Guide (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sc342670/$file/eequ100_v2r4.pdf), SC34-2670

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6269
- *z/VM: Performance*, SC24-6301

© Copyright IBM Corp. 1991, 2020

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: TCP/IP User's Guide*, SC24-6333
- *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: TCP/IP Programmer's Reference*, SC24-6332
- *CPI Communications Reference*, SC26-4399
- *Common Programming Interface Resource Recovery Reference*, SC31-6821
- z/OS: IBM Tivoli Directory Server Plug-in Reference for z/OS (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa760169/$file/glpa300_v2r4.pdf), SA76-0169
- z/OS: Language Environment Concepts Guide (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa380687/$file/ceea800_v2r4.pdf), SA38-0687
- z/OS: Language Environment Debugging Guide (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4ga320908/$file/ceea100_v2r4.pdf), GA32-0908
- z/OS: Language Environment Programming Guide (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa380682/$file/ceea200_v2r4.pdf), SA38-0682
- z/OS: Language Environment Programming Reference (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa380683/$file/ceea300_v2r4.pdf), SA38-0683

- z/OS: Language Environment Runtime Messages (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa380686/$file/ceea900_v2r4.pdf), SA38-0686
- z/OS: Language Environment Writing Interlanguage Communication Applications (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa380684/$file/ceea400_v2r4.pdf), SA38-0684
- z/OS: MVS Program Management Advanced Facilities (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa231392/$file/ieab200_v2r4.pdf), SA23-1392
- z/OS: MVS Program Management User's Guide and Reference (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa231393/$file/ieab100_v2r4.pdf), SA23-1393

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: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: VM Dump Tool*, GC24-6335
- z/OS and z/VM: Hardware Configuration Definition Messages (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sc342668/$file/cbdm100_v2r4.pdf), SC34-2668

z/VM Facilities and Features

Data Facility Storage Management Subsystem for 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-Express Customer's Guide and Reference (www.ibm.com/support/pages/open-systems-adapter-express-customers-guide-and-reference-0), SA22-7935
- Open Systems Adapter-Express Integrated Console Controller User's Guide (www.ibm.com/support/pages/node/6019810), SC27-9003
- Open Systems Adapter-Express Integrated Console Controller 3215 Support (www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.ioa/ioa.htm), SA23-2247
- Open Systems Adapter/Support Facility on the Hardware Management Console (www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.ioa/ioa.htm), SC14-7580

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

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
- z/OS: Network Job Entry (NJE) Formats and Protocols (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sa320988/$file/hasa600_v2r4.pdf), SA32-0988

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4gc350033/$file/ickug00_v2r4.pdf), GC35-0033

Environmental Record Editing and Printing Program

- Environmental Record Editing and Printing Program (EREP): Reference (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4gc350152/$file/ifc2000_v2r4.pdf), GC35-0152
- Environmental Record Editing and Printing Program (EREP): User's Guide (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4gc350151/$file/ifc1000_v2r4.pdf), GC35-0151

Index

Special Characters

*ACCOUNT 699
*ACCOUNT (account system service) 699
*ASYNCMD 719
*ASYNCMD (asynchronous CP command response system service) 719
*BLOCKIO
  condition and return codes 723, 726
  ending communication 727
  IUCV ACCEPT 722
  IUCV CONNECT 773
  IUCV SEND 723
  IUCV SEVER 722
  multiple chained block I/O 725
  sample programs 981
  single block I/O 723
*BLOCKIO (DASD block I/O system service)
  condition and return codes 723, 726
  ending communication 727
  establishing communication 721
  IUCV ACCEPT 722
  IUCV CONNECT 721, 773
  IUCV SEND 723
  IUCV SEVER 722
  multiple chained block I/O 725
  sample programs 981
  single block I/O 723
*IDENT 393, 731
*IDENT (identify system service)
  establishing a connection 731
  processing requests to manage a resource 733
  sever reason codes 736
*IDENT (LU name qualifier) 419
*LOGREC
  IUCV ACCEPT 729
  IUCV CONNECT 729
  IUCV SEVER 729
*LOGREC (error logging system service)
  IUCV ACCEPT 729
  IUCV CONNECT 729
  IUCV SEVER 729
*MSG 739
*MSG (message system service) 739
*MSGALL 741
*MSGALL (message all system service) 741
*RPI 591
*RPI (access verification system service) 591
*RPI system service, defining 593
*SIGNAL
  connecting with 747
  establishing communications with 747
  leaving 750
  receiving signals 749
  sending signals 749
*SIGNAL (signal system service)
  connecting with 747
*SIGNAL (signal system service) *(continued)*
  establishing communications with 747
*SPL 752
*SPL (spool system service)
  AFP printing interface 752
  CLOSE function 757
  DISABLE function 763
  ENABLE function 763
  generic interface 765
  MESSAGE function 758
  NOTIFY function 765
  PURGE function 765
  READ-SFBLOK function 759
  READ-SPLINK function 762
  READ-XAB function 761
  SELECT function 754
  SEND function 764
*SYMPTOM
  disconnecting 774
  IUCV ACCEPT 773
  IUCV SEVER 773
*SYMPTOM (symptom system service)
  disconnecting 774
  IUCV ACCEPT 773
  IUCV SEVER 773
*USERID (LU name qualifier) 419

Numerics

3270 display information, accessing with DIAGNOSE code X'8C' 110
370 accommodation services, DIAGNOSE code X'268' 225
370 accommodation support
  370-mode constraints 893
  architecture 901
  background 893
  CMS 370 accommodation facility 909
  CMS370AC 909
  definition 901
  ESA-family instructions 902
  facilities not provided 898
  high-level description 893
  how to activate it 895
  hybrid interruptions 907
  interruption parameters 906
  interval timer 903
  mapped PSWs 904, 905, 910
  other instructions 903
  overview 893
  possible adverse effects on working programs 898
  PSW conversions 904
  PSW stealing programs 894, 895, 906, 908, 909
  running a restricted CMS MODULE 897
  SET CMS370AC ON 909
  System/370 channel-status word (CSW) 907
  System/370 I/O instructions 901
  System/370 instructions 901

370 accommodation support *(continued)*
vestigial status 901–903, 907, 908
when to activate it 894
370 synchronous I/O, DIAGNOSE code X'20' 38

A

abend codes, DIAGNOSE code X'84' 102
abend conditions, APPC
ALLOCATE 578
CONFIRM 579
CONFIRMED 579
DEALLOCATE 580
RECEIVE_AND_WAIT 584
REQUEST_TO_SEND 584
SEND_DATA 585
SEND_ERROR 587
ACCEPT
IUCV function
condition codes 321
connection complete interrupt 322
format 319
from the DASD block I/O system service 722
from the error logging system service 729
from the Symptom system service 773
parameter descriptions 319
parameter list format 321
program exceptions 322
return codes 322
using 319
IUCV function used in APPC/VM
completion 528
condition codes 527
format 524
parameter descriptions 524
parameter list format 526
parameters for communication servers 525
program exceptions 528
return codes 527
state changes 528
using 524
logical device support facility function
DIAGNOSE code X'7C' subcode X'00000002' 90
status returned to CP 1013
access
an address space 788
data space storage 787
access 3270 display information, X'8C' 110
access exception
ADRSPACE macro 799
ALSERV macro 816
DIAGNOSE code X'00' 16
DIAGNOSE code X'04' 19
DIAGNOSE code X'08' 22
DIAGNOSE code X'10' 23, 25
DIAGNOSE code X'14' 35
DIAGNOSE code X'210' 197
DIAGNOSE code X'220' 225, 263
DIAGNOSE code X'248' 203
DIAGNOSE code X'250' 215
DIAGNOSE code X'2E0' 291
DIAGNOSE code X'34' 44
DIAGNOSE code X'4C' 50
DIAGNOSE code X'5C' 62
access exception *(continued)*
DIAGNOSE code X'64' 80
DIAGNOSE code X'68' 989
DIAGNOSE code X'70' 82
DIAGNOSE code X'74' 84
DIAGNOSE code X'84' 104
DIAGNOSE code X'88' 110
DIAGNOSE code X'8C' 112
DIAGNOSE code X'90' 113
DIAGNOSE code X'98' 132
DIAGNOSE code X'A0' 135
DIAGNOSE code X'A4' 144
DIAGNOSE code X'A8' 148
DIAGNOSE code X'B0' 151
DIAGNOSE code X'B4' 153
DIAGNOSE code X'B8' 155
DIAGNOSE code X'BC' 159
DIAGNOSE code X'CC' 161
DIAGNOSE code X'D4' 164
DIAGNOSE code X'D8' 166
DIAGNOSE code X'DC' 171
DIAGNOSE code X'E4' 184
DIAGNOSE code X'EC' 186
DIAGNOSE code X'F8' 189
general definition 8
MAPMDISK macro 827
PFAULT macro 855
REFPAGE macro 864
VMUDQ macro 880
access list services, ALSERV macro 815
access list services, DIAGNOSE code X'240' 1026
Access real subsystem external interruption 1040
access real subsystem, DIAGNOSE code X'254' 1032
access security types
of APPC ALLOCATE 577
with APPCVM CONNECT 420
access system information, DIAGNOSE code X'26C' 226
access verification system service (*RPI) 591
access virtual machine information, DIAGNOSE code X'260' 222
access-list entry (ALE) 815
Access-List Entry (ALE) 788
access-list-entry token (ALET) 816
Access-List-Entry token (ALET) 7, 787
access-register mode
execution of DIAGNOSE codes and 5
ACCESSLIST directory control statement 786
account system service (*ACCOUNT)
*ACCOUNT user ID 699
disconnecting from 700
establishing communication 699
receiving accounting records 700
accounting
accounting record formats 718
adding your own source code 718
record
adding your own 718
CPU capability 715
dedicated device 703
format of 700
Inter-System Facility for Communications 708
journaling 704
SET PRIVCLASS 710, 713
SNA/CCS 708

accounting *(continued)*
record *(continued)*
temporary disk space 703
type 04 704
type 05 705
type 06 705
type 07 708
type 08 706
type 09 708
type 0A 710
type 0I 707
type 1 701
type 2 703
type 3 703
type B 712
type C 713
type C0 718
type D 715
type E 715
type F 717
user-initiated 718
virtual disk space 712
virtual machine resource usage 701, 717
record formats 700
accounting interface for time-of day clock, DIAGNOSE code X'70' 80
accounting records, generating, DIAGNOSE code X'4C' 47
accounting records, receiving 700
ACI (Access Control Interface)
*RPI system service 593
ACIPARMS formats
APPC CONNECT 678
APPC SEVER 680
AUTOLOG command 641
CHANGE TO command 644, 665
CLOSE TO command 645
COUPLE command 646
COUPLEN command 647
DIAGNOSE code X'14' 670
DIAGNOSE code X'23C' 677
DIAGNOSE code X'290' 676
DIAGNOSE code X'64' 670
DIAGNOSE code X'68' 671, 672
DIAGNOSE code X'94' 668, 672
DIAGNOSE code X'B8' 673
DIAGNOSE code X'BC' 673
DIAGNOSE code X'D4' 674
DIAGNOSE code X'E4' 675
GIVE command 649
IUCV CONNECT 681
IUCV SEVER 682
LINK command 649
LOGOFF command 651
LOGON command 652
MDISK command 683
MESSAGE command 656
POSIX group database query function 686
POSIX Set ID functions 685
POSIX user database query function 688
PURGE command 657
QUERY RDR/PRT/PUN command 658
QUERY TAG command 657
Resource access authorization check 691
RSTDSEG command 693
ACI (Access Control Interface) *(continued)*
ACIPARMS formats *(continued)*
SCIF event 693, 694
SEND command 659
SPOOL command 660
SPXTAPE DUMP command 661
SPXTAPE LOAD command 661
START command 661
STORE HOST command 662
TAG command 663
TAG QUERY command 657
TRANSFER command 664, 665
TRSAVE TO command 665
TRSOURCE command 666
TRSOURCE ENABLE command 667
VMDUMP TO command 668
VMRELOCATE command 668
XAUTOLOG command 641, 669
called by ESM 602
commands that support calls to ACI 609
data area 606
DIAGNOSE code X'A0' processor 599
DIAGNOSE codes that support calls to ACI 609
function 591
HCPRPD module
function 599
interface specifications 599
return codes 601
HCPRPE module 602
HCPRPF module 606
HCPRPG module 606
HCPRPI module
*RPI system service 593
IUCV interface 593
request services from ESM 594
HCPRPL module 606
HCPRPP module 606
HCPRPW module
interface specifications 596, 598
HCPRPWEP module
function 596
HCPRWA module 607
IUCV interface 593
logon password prompting routine 598
logon password verification routine 596
overview 592
request services from ESM 594
security bits 609
security bits, checking 611
security bits, setting of 610
security process 592
work area 607
ACIPARMS control block (format) 622
ACIPARMS parameter list
call formats
APPC CONNECT 678
APPC SEVER 680
AUTOLOG command 641
CHANGE TO command 644, 665
CLOSE TO command 645
COUPLE command 646
COUPLEN command 647
DIAGNOSE code X'14' 670
DIAGNOSE code X'23C' 677

ACIPARMS parameter list *(continued)*
call formats *(continued)*
DIAGNOSE code X'290' 676
DIAGNOSE code X'64' 670
DIAGNOSE code X'68' 671, 672
DIAGNOSE code X'94' 668, 672
DIAGNOSE code X'B8' 673
DIAGNOSE code X'BC' 673
DIAGNOSE code X'D4' 674
DIAGNOSE code X'E4' 675
GIVE command 649
IUCV CONNECT 681
IUCV SEVER 682
LINK command 649
LOGOFF command 651
LOGON command 652
MDISK command 683
MESSAGE command 656
POSIX group database query function 686
POSIX Set ID functions 685
POSIX user database query function 688
PURGE command 657
QUERY RDR/PRT/PUN command 658
QUERY TAG command 657
Resource access authorization check 691
RSTDSEG command 693
SCIF event 693, 694
SEND command 659
SPOOL command 660
SPXTAPE DUMP command 661
SPXTAPE LOAD command 661
START command 661
STORE HOST command 662
TAG command 663
TAG QUERY command 657
TRANSFER command 664, 665
TRSAVE TO command 665
TRSOURCE command 666
TRSOURCE ENABLE command 667
VMDUMP TO command 668
VMRELOCATE command 668
XAUTOLOG command 641, 669
function 592
ACNT command 701, 717
activate CP directory, DIAGNOSE code X'3C' 44
add
your own accounting records and source code 718
ADD function of ALSERV macro 817
adding an ALE to an access list 787
address
absolute 5
guest 5
processing of 5
real 5
spaces, selection of 7
address lists 307, 453, 479, 493, 512
address space identification token (ASIT) 787
address space services, ADRSPACE macro 797
address space services, DIAGNOSE code X'23C' 1021
address spaces
access-list entry (ALE)
X'240' 1026
creating 800
definition 783
address spaces *(continued)*
destroying 804
dropping addressability 789
identification 798
identification token (ASIT) 798
isolating a shared 789
management functions
X'23C' 1021
mapping
X'244' 1028
mapping minidisks to 790
name 798
overview 783
permitting another user to access 787
primary 785
querying information 812
restoring to private state 806
selection of 7
states 798, 806
support 784
addressability to an address space, establish 787
addressability, parameter 394
addresses
absolute 297
guest 297
processing of 297
real 297
addressing exception
APPCVM CONNECT 428
APPCVM QRYSTATE 450
APPCVM RECEIVE 460
APPCVM SENDCNF 469
APPCVM SENDCNFD 474
APPCVM SENDDATA 486
APPCVM SENDERR 498
APPCVM SENDREQ 503
APPCVM SETMODFY 507
APPCVM SEVER 517
DIAGNOSE code X'254' 1039
DIAGNOSE code X'D0' 162
general definition 8
IUCV ACCEPT 322, 528
IUCV CONNECT 329, 532
IUCV DECLARE BUFFER 332, 536
IUCV DESCRIBE 336, 539
IUCV INTERRUPT POLL 339, 542
IUCV PURGE 343
IUCV QUERY 546
IUCV QUIESCE 346
IUCV RECEIVE 351
IUCV REJECT 354
IUCV REPLY 358
IUCV RESUME 363
IUCV SEND 370
IUCV SET CONTROL MASK 552
IUCV SET MASK 555
IUCV SEVER 378, 559
IUCV TEST COMPLETION 382, 564
addressing-capability
DIAGNOSE code X'E0' 1017–1019
exception 1017, 1018
exception condition 1019
addressing-capability exception 8
ADRSPACE macro

ADRSPACE macro *(continued)*
CREATE function
address space identification token (ASIT) 787
creating a remote work area example 796
DECLARE function 803
DESTROY function 790, 804
forcing unique work areas example 795
ISOLATE function 789, 806
PERMIT function 787, 789, 808
QUERY function 812
ADSR macro 961
advanced function printer *SPL interface 752
Advanced Program-to-Program Communication (APPC)
conversation states 574
conversations in 573
currently-defined error codes 402
currently-defined sense codes 403
currently-defined sever codes 399
definition 387
error conditions 399
functions 576, 587
interrupts 573
mapped with APPC/VM 573
operator control verbs mapped to AVS commands 576
return codes 575
states
basic 388
for coordinated resource recovery 405
states, APPC/VM implementation of 574
verb names mapped to APPC/VM macro functions
ALLOCATE 576
CONFIRM 578
CONFIRMED 579
DEALLOCATE 580
FLUSH 581
GET_ATTRIBUTES 581
PREPARE_TO_RECEIVE 581
RECEIVE_AND_WAIT 582
Advanced Program-to-Program Communication/VM (APPC/VM)
APPC verb names mapped to APPC/VM macro functions 575
basics 387
condition codes and return codes 396
defined SENDERR codes 494
differences from IUCV 567
error/sever codes 399
example format of PIP variable 423
functions, assembler
CONNECT 412
invoking 390
QRYSTATE 447
RECEIVE 451
SENDCNF 465
SENDCNFD 471
SENDDATA 475
SENDERR 490
SENDREQ 501
SETMODFY 505
settings for optional parameters 395
SEVER 509
state table 403
using 392
implementation of APPC conversation states 574
Advanced Program-to-Program Communication/VM (APPC/VM) *(continued*
interrupts 388, 390, 573
IUCV functions for use in
ACCEPT 524
CONNECT 529
DCLBFR 533
DESCRIBE 538
IPOLL 540
QUERY 543
RTRVBFR 548
SETCMASK 550
SETMASK 553
SEVER 556
TESTCMPL 562
TESTMSG 566
using HELP for 523
IUCV macro functions for use in 521
logical record format 478
managing a resource 393
mapped with APPC 573
overview 387
parameter lists
formatting with MF=L 395
reading 395
paths 387
performance 534
pip variables 445
return codes 575
security subfield in an attach FMH5 for VM 439
shared functions that can be used in CMS 521
shared functions that should be avoided in 521
starting a conversation 392
state table for error conditions 407
state table for functions 403
states 388
VM communication server area 445
AFP printing *SPL interface 752
ALE (access-list entry) 815
ALE (Access-List Entry) 788
ALEN
DIAGNOSE code X'D0' 162
ALEN-translation exception
DIAGNOSE code X'E0' 1017, 1018
translation exception 1017, 1018
ALEN-translation exception condition
DIAGNOSE code X'E0' 1019
translation exception 1019
ALET (access-list-entry token) 816
ALET (Access-List-Entry token) 7, 787
ALET-specification exception
DIAGNOSE code X'E0' 1017, 1018
ALET-specification exception condition
DIAGNOSE code X'E0' 1019
allocate data
examples 444
receiving 451
ALLOCATE, APPC verb
abend conditions 578
mapped with APPC/VM 576
parameters 576
state changes 578
ALLOCD parameter of APPCVM CONNECT 415
ALSERV macro
ADD function 787, 817, 852

ALSERV macro *(continued)*
DECLARE function 820
list of functions 815
REMOVE function 790, 821
alternate user ID 162
answer data, APPCVM SENDDATA 475
APPC (Advanced Program-to-Program Communication)
conversation states 574
conversations in 573
currently-defined error codes 402
currently-defined sense codes 403
currently-defined sever codes 399
definition 387
error conditions 399
functions 576, 587
interrupts 573
mapped with APPC/VM 573
operator control verbs mapped to AVS commands 576
return codes 575
states
basic 388
for coordinated resource recovery 405
states, APPC/VM implementation of 574
verb names mapped to APPC/VM macro functions
ALLOCATE 576
CONFIRM 578
CONFIRMED 579
DEALLOCATE 580
FLUSH 581
GET_ATTRIBUTES 581
PREPARE_TO_RECEIVE 581
RECEIVE_AND_WAIT 582
APPC data 478
APPC/VM (Advanced Program-to-Program Communication/VM)
APPC verb names mapped to APPC/VM macro functions 575
basics 387
condition codes and return codes 396
defined SENDERR codes 494
differences from IUCV 567
error/sever codes 399
example format of PIP variable 423
functions, assembler
CONNECT 412
invoking 390
QRYSTATE 447
RECEIVE 451
SENDCNF 465
SENDCNFD 471
SENDDATA 475
SENDERR 490
SENDREQ 501
SETMODFY 505
settings for optional parameters 395
SEVER 509
state table 403
using 392
implementation of APPC conversation states 574
interrupts 388, 390, 573
IUCV functions for use in
ACCEPT 524
CONNECT 529
DCLBFR 533
APPC/VM (Advanced Program-to-Program Communication/VM) *(continued*
IUCV functions for use in *(continued)*
DESCRIBE 538
IPOLL 540
QUERY 543
RTRVBFR 548
SETCMASK 550
SETMASK 553
SEVER 556
TESTCMPL 562
TESTMSG 566
using HELP for 523
IUCV macro functions for use in 521
logical record format 478
managing a resource 393
mapped with APPC 573
overview 387
parameter lists
formatting with MF=L 395
reading 395
paths 387
performance 534
pip variables 445
return codes 575
security subfield in an attach FMH5 for VM 439
shared functions that can be used in CMS 521
shared functions that should be avoided in 521
starting a conversation 392
state table for error conditions 407
state table for functions 403
states 388
VM communication server area 445
APPCPASS directory statement 421
APPCVM macro functions
APPC verb names mapped to 575
CONNECT 412
invoking 390
QRYSTATE 447
RECEIVE 451
SENDCNF 465
SENDCNFD 471
SENDDATA 475
SENDERR 490
SENDREQ 501
SETMODFY 505
settings for optional parameters 395
SEVER 509
state table 403
using 392
using HELP for 411
APPCVM return codes
CONNECT 424
QRYSTATE 448
RECEIVE 454
SENDCNF 467
SENDCNFD 472
SENDDATA 480
SENDERR 495
SENDREQ 502
SETMODFY 506
SEVER 514
appendix of data areas used by DIAGNOSE codes 969
application programs' use for VM data spaces 784
applications, VMCF 990

architecture changes for 370 Accommodation Facility 901
architecture, ESA/XC
  ESA/XC architecture 784
area for VM allocate data 434, 441
areas, macro work 823
ASIT (address space identification token) 787, 798
assigned storage locations, collaborative memory management 886
asynchronous communications
  not based on APPC 573
asynchronous CP command response (*ASYNCMD)
  establishing communication 719
  message limit 719
  record types 720
  sending and receiving data 719
authorization
  connect to *IDENT 393
  revoking a resource 394
AUTHORIZE function of VMCF 996, 1001
AUTOLOG command 700
avoiding IUCV external interrupts 301

B

background information on 370 accommodation facility 893
Backout_Received conversation state 406
backout_received state 406
Backout_Required conversation state 406
backout_required state 406
backspace one record 29
base set of APPC verbs and APPC/VM functions 575
BASIC conversation type 415
bit map fields for DIAGNOSE code X'00' 13
block form of ordered paging-referencing 863
block I/O entry, format of 209
block I/O external interruption 215
block I/O operations for DASD in standard CMS blocksize 203
block numbers, pool-relative 824
block-content state 884
block-usage state 883
block-volatility exception 887
BPLBK DSECT 725
buffer extension, interrupt 535
buffer information
  subcode X'0000', DIAGNOSE code X'BC' 155
  subcode X'0004', DIAGNOSE code X'BC' 157
buffers
  application 534
  control 305, 534
  IUCV external interrupt 304
  use by APPC 478
  used by IUCV 302
bypass minidisk cache 723, 725

C

CANCEL function of PFAULT macro 857
CANCEL function of VMCF 996, 1001
CCED 429, 430
chained block I/O, multiple 725
changes, state
  ALLOCATE (APPC) 578
changes, state *(continued)*
  APPCVM CONNECT 429
  APPCVM QRYSTATE 450
  APPCVM RECEIVE 461
  APPCVM SENDCNF 469
  APPCVM SENDCNFD 474
  APPCVM SENDDATA 486
  APPCVM SENDERR 499
  APPCVM SENDREQ 503
  APPCVM SETMODFY 508
  APPCVM SEVER 518
  CONFIRM (APPC) 579
  CONFIRMED (APPC) 579
  DEALLOCATE (APPC) 580
  IUCV ACCEPT 528
  IUCV DCLBFR 537
  IUCV RTRVBFR 548
  RECEIVE_AND_WAIT (APPC) 584
  REQUEST_TO_SEND (APPC) 584
  SEND_DATA (APPC) 585
  SEND_ERROR (APPC) 587
changing data flow direction 476
channel program modification, DIAGNOSE code X'28' 41
CLASDASD 970
CLASFBA 970
CLASGRAF 970
CLASSPEC 970
CLASTAPE 970
CLASTERM 969
CLASURI 970
CLASURO 970
clean-up after virtual IPL by device, DIAGNOSE code X'40' 1015
CLEAR I/O (CLRIO) instruction 894, 901
CLEAR SUBCHANNEL (CSCH) instruction 903, 908
CLOSE function of spool system service 757
CLRIO (CLEAR I/O) instruction 894, 901
CMS communications directory, use in APPCVM CONNECT 392, 413, 414, 417, 418, 440
CMS interface to APPC/VM 733
coding example for the DIAGNOSE instruction 4
collaborative memory management assist
  assigned storage locations 886
  block-content states 884
  block-usage states 883
  block-volatility exception 887
  EXTRACT AND SET STORAGE ATTRIBUTES 888
  implications for ESA/390 and ESA/XC guests 892
  implications for saved systems and segments 892
  implications for the DIAGNOSE instruction and non-CPU accesses 891
  implications for the VMDUMP command 892
  interruptions 887
  overview 883
  program exceptions 888
  resets 886
collection of z/VM systems (TSAF Collection)
  revoking gateways when merging 394
  revoking resources when merging 394
command
  ACNT 701, 717
  AUTOLOG 700
  FORCE 700
  LINK 700

command *(continued)*
LOGON 700
SHUTDOWN 700
XAUTOLOG 700
communication
between virtual machines 297, 989
IUCV
DASD block I/O system service 721
example 302
identify system service 731
communication, signals between virtual machine groups 747
communications partner
connecting to 432
receiving from 463
sending to (SENDCNF) 470
sending to (SENDCNFD) 474
sending to (SENDDATA) 488
sending to (SENDERR) 500
sending to (SENDREQ) 504
severing from 519
severing paths 549
communications servers
accepting connections 525
making connections 441
parameters on IUCV ACCEPT 525
comparing APPC with APPC/VM 573
completion codes for DIAGNOSE codes in general 9
completion of functions
ACCEPT 528
APPCVM CONNECT 429
DCLBFR 537
DESCRIBE 539
RECEIVE 462
RETRIEVE BUFFER 549
SENDCNF 470
SENDCNFD 474
SENDDATA 488
SENDERR 499
SENDREQ 503
SETCMASK 552
SETMASK 555
SETMODFY 508
SEVER (APPCVM) 518
TESTCMPL 565
TESTMSG 566
COMSRV parameter of IUCV ACCEPT 525
condition codes
APPCVM
CONNECT 424
overview 396
QRYSTATE 448
RECEIVE 454
SENDCNF 467
SENDCNFD 472
SENDDATA 480
SENDERR 495
SENDREQ 502
SETMODFY 506
SEVER 514
DIAGNOSE codes
general description 9
X'08' 21
X'14' 35
X'18' 37
condition codes *(continued)*
DIAGNOSE codes *(continued)*
X'20' 39
X'210' 197
X'24' 41
X'250' 212
X'254' 1038
X'268' 226
X'28' 43
X'2AC' 1076
X'34' 44
X'3C' 45
X'4C' 49
X'5C' 61
X'64' 78, 79
X'74' 83
X'7C' 86, 87
X'84' 102
X'90' 112
X'94' 120
X'A4' 142, 143
X'A8' 147, 148
X'B4' 152
X'B8' 154
X'BC' 158
X'D0' 162
X'D8' 166
X'DC' 170
X'E4' 183
X'EC' 186
IUCV
ACCEPT 321, 527
CONNECT 328, 531
DECLARE BUFFER 332, 536
DESCRIBE 335, 538
INTERRUPT POLL 338, 541
PURGE 341
QUERY 344, 544
QUIESCE 346
RECEIVE 350
REJECT 353
REPLY 357
RESUME 363
RETRIEVE BUFFER 365, 548
SEND 370
SET CONTROL MASK 373, 551
SET MASK 375, 554
SEVER 377, 558
TEST COMPLETION 382, 563
TEST MESSAGE 383, 566
multiple block I/O 726
single block I/O 723
VMUDQ macro 879
conditions necessary for handshaking 854
Confirm state 388, 403, 404
CONFIRM synchronization level 414
CONFIRM, APPC verb
abend conditions 579
mapped with APPC/VM 578
parameters 578
state changes 579
CONFIRMED, APPC verb
abend conditions 579
mapped with APPC/VM 579

CONFIRMED, APPC verb *(continued)*
parameters 579
state changes 579
CONNECT function of APPCVM
allocate data 434
communication servers 441
completion 429
condition codes 424
connection complete extended data 429, 430
connection pending extended data 434, 438
connection pending interrupt 432
description 412
FMH5 438
format 412
input parameter list 416
mapped with APPC 576
output parameter list 426
parameter descriptions 413
parameter list extension 417, 418
parameter list format 416
pip variables 422, 445
program exceptions 428
return codes 424
security subfield in an attach FMH5 for VM 439
state changes 429
to communication partner 432
to start a conversation 392
VM architected area 434, 441
VM communication server area 445
CONNECT function of IUCV
*IDENT sever reason codes 736
condition codes 328
connection pending interrupt 329
format 324
parameter descriptions 325
parameter list format 327
program exceptions 329
return codes 328
signal system service 747
to DASD block I/O system service 721
to error logging system service 729
to signal system service 747
to spool system service 752, 765
to symptom system service 773
used in APPC/VM
condition codes 531
format 529
parameter descriptions 529
parameter list format 531
program exceptions 532
return codes 531
to revoke a resource 394
using 529
using 324
Connect state 388, 403, 404
connect to *IDENT
to manage a resource 394
to revoke a resource 394
connect to programs
resource manager 394
connection complete interrupts
extended data 429, 430
format 426, 429
connection parameter list extension 417, 418
connection pending interrupts
extended data 434, 438
format 432
connection quiesced interrupt 347
connection resumed interrupt 364
connection severed interrupt 378
control application monitor data generation, DIAGNOSE code X'DC' 166
control blocks used by DIAGNOSE codes 969
control buffer, IUCV 305, 331, 534
control functions of VMCF 995
control path, IUCV 305, 331, 534
control the PA2 function key, DIAGNOSE code X'54' 50
Control Virtual Machine Time Bomb, DIAGNOSE code X'288' 268
conversations
APPC 573
starting an APPC one 573
states 388
states, APPC/VM 574
copy-to-primary service, DIAGNOSE code X'248' 202
CP (Control Program)
ACI (access control interface)
*RPI system service 593
ACIPARMS formats 639
ACIPARMS general format 622
called by ESM 602
commands that support calls to ACI 609
data area 606
DIAGNOSE code X'A0' processor 599
DIAGNOSE codes that support calls to ACI 609
function 591
HCPA0LBK format 617
HCPA0UBK general format 620
HCPRPD module 599
HCPRPE module 602
HCPRPF module 606
HCPRPG module 606
HCPRPI module 593
HCPRPL module 606
HCPRPP module 606
HCPRPW module 596
HCPRWA module 607
IUCV interface 593
logon password verification routine 596, 598
overview 592
request services from ESM 594
security bits 609
security bits, checking 611
security bits, setting of 610
security process 592
work area 607
command
ACNT 701, 717
AUTOLOG 700
FORCE 700
LINK 700
LOGON 700
SHUTDOWN 700
XAUTOLOG 700
exits
access control interface 591
external security manager 591
HCPRPD module 599

CP (Control Program) *(continued)*
exits *(continued)*
HCPRPI module 593
HCPRPW module 596
macros
HCPEXIT 596, 598, 599, 602
system services and list of user IDs 317
CP and IFL capacity information, STHYI instruction 914
CP communication, DIAGNOSE code X'264' 1044
CP directory activation, DIAGNOSE code X'3C' 44
CP macros
ADSR 961
APPCVM
CONNECT 412
QRYSTATE 447
RECEIVE 451
SENDCNF 465
SENDCNFD 471
SENDDATA 475
SENDERR 490
SENDREQ 501
SEVER 509
data space
ADRSPACE 797
ALSERV 815
coding 793
DEFWORKA 823
MAPMDISK 824
PFAULT 852
preferred use 793
REFPAGE 863
using HELP for 793
data space considerations 786
DIAG 4
execution in access-register mode 5
IUCV
ACCEPT 319, 524
advantages of using 306
application considerations for virtual MP 308
condition codes and return codes 522
CONNECT 324, 529
DECLARE BUFFER 331, 533
DESCRIBE 334, 538
for use in APPC/VM 521
in a distributed environment 309
INTERRUPT POLL (IPOLL) 337, 540
PURGE 340
QUERY 344, 543
QUIESCE 345
RECEIVE 348
REJECT 352
related to APPC/VM 391
REPLY 355
RESUME 362
RETRIEVE BUFFER 365, 548
SEND 366
SET CONTROL MASK 372, 550
SET MASK 374, 553
SEVER 376, 556
shared function that can be used in CMS 521
shared function that should be avoided in 521
terminology 306
TEST COMPLETION 379, 562
TEST MESSAGE 383, 566
CP macros *(continued)*
VMUDQ 875
work areas 793
CP SET commands with an IUCV option 739
CP system services
access verification (*RPI) 591
account (*ACCOUNT) 699
asynchronous CP command response (*ASYNCMD) 719
DASD block I/O (*BLOCKIO) 721
error logging (*LOGREC) 729
identify (*IDENT) 731
IUCV communication 317
list of user IDs 317
message (*MSG) 739
message all (*MSGALL) 741
sample programs using DASD block I/O (*BLOCKIO) 981
signal (*SIGNAL) 747
spool (*SPL)
AFP printing interface 752
generic interface 765
symptom (*SYMPTOM) 773
CP370 device classes 969
CP370 device features 971
CP370 device types 969
CP370 virtual device flags 972
CP370 virtual device status 971
CPED 434, 438, 445
CPU capability accounting records 715
CPU identification, real 197
create
a data space 786
remote work area example, data spaces 796
create a full-pack overlay minidisk with DIAGNOSE code X'E4' 174
CREATE function of ADRSPACE macro 800
created data spaces being shared with other users 783
creating
a remote work area example, data spaces 796
address spaces 797, 800
data spaces 786
CSCH (CLEAR SUBCHANNEL) instruction 903, 908
currently-defined APPC/VM error codes 402
currently-defined APPC/VM sense codes 403
currently-defined APPC/VM sever codes 399

D

DASD Block I/O system service
condition and return codes 723, 726
ending communication 727
establishing communication 721
IUCV ACCEPT 722
IUCV CONNECT 721, 773
IUCV SEND 723
IUCV SEVER 722
multiple chained block I/O 725
sample programs 981
single block I/O 723
DASD I/O, standard, DIAGNOSE code X'18' 36
DASD in standard CMS blocksize, block I/O operations 203
data areas used by DIAGNOSE codes 969
DATA parameter (APPC)
of RECEIVE_AND_WAIT 582
of SEND_DATA 585

data space macros
ADRSPACE 797
ALSERV 815
coding 793
DEFWORKA 823
MAPMDISK 824
overview 783
PFAULT 852
REFPAGE 863
use considerations 786
using HELP for 793
data spaces
accessing storage 787
adding an ALE to an access list 787
address space identification token (ASIT) 787
coding macros 793
CP macro use considerations 786
created data spaces being shared with other users 783
creating 786
creating a remote work area example 796
creating with ADRSPACE CREATE 786
definition 783
DIAGNOSE code use considerations 786
forcing unique work areas example 795
instance 787
nonreentrant program example 794
overview 783
reentrant program example 794
summary of operations 785
use in applications 786
uses 784
data transfer error codes, VMCF 1009
data transfer functions of VMCF 997
data transfer, IUCV two-way 298
data, how it is sent
details 478
overview 393
DD8PARM0 DSECT 165
DEALLOCATE, APPC verb
abend conditions 580
mapped with APPC/VM 580
parameters 580
state changes 580
DECLARE BUFFER function of IUCV
condition codes 332
format 331
parameter descriptions 331
parameter list format 332
program exceptions 332
return codes 332
used with APPC/VM
condition codes 536
format 533
interrupt buffer extension 535
parameter descriptions 533
parameter list format 535
program exceptions 536
return codes 536
state changes 537
using 533
using 331
DECLARE function
ADRSPACE macro 803
ALSERV macro 820
DECLARE function *(continued)*
MAPMDISK macro 828
PFAULT macro 858
REFPAGE macro 865
declaring buffers for interrupts 388
dedicated
device accounting record 703
Defer_Receive conversation state 405
defer_receive state 405
Defer_Sever conversation 405
defer_sever state 405
define
full-pack overlay minidisk with DIAGNOSE code X'E4' 174
macro work areas with DEFWORKA macro 823
DEFINE function of the MAPMDISK 829
definition of 370 Accommodation Facility 901
DEFWORKA macro
forcing unique work areas example 795
nonreentrant program example 794
reentrant program example 794
DESCRIBE function of IUCV
condition codes 335
format 334
parameter list format 334
program exceptions 335
required parameters 334
return codes 335
used with APPC/VM
completion 539
condition codes 538
format 538
parameter descriptions 538
program exceptions 539
state changes 539
using 538
using 334
description of IUCV functions in general 306
designated guest information, STHYI instruction 943
designated resource pool information, STHYI instruction 952
DESTROY function of ADRSPACE macro 804
destroying
address spaces 797, 804
destroying a data space 790
determine virtual machine storage size, DIAGNOSE code X'60' 62
device
classes, CP370 969
features, CP370 971
information, DIAGNOSE code X'210' 189
types and features, DIAGNOSE code X'24' 39
types, CP370 969
DIAG macro
example 5
format 4
DIAGNOSE code
bit map fields for X'00' 13
condition codes and return codes 9
control blocks used by 969
data areas used by 969
data space considerations 786
examine host storage X'04' 16
example of coding 4
how address spaces are selected 7

DIAGNOSE code *(continued)*
how addresses are processed 5
how error conditions are reported 7
mode restrictions 5
overview 3
using to initiate accounting records 718
X'00', store extended-identification code 13
X'04', examine host storage 16
X'08', virtual console function 19
X'0C', pseudo timer 22
X'10', release pages 24
X'14', input spool file manipulation
subcode X'0000' 26
subcode X'0004' 27
subcode X'0008' 28
subcode X'000C' 28
subcode X'0010' 28
subcode X'0014' 29
subcode X'0018' 29
subcode X'001C' 29
subcode X'0020' 30
subcode X'0024' 30
subcode X'0028' 30
subcode X'002C' 30
subcode X'0FFE' 31
subcode X'0FFF' 34
X'18', standard DASD I/O 36
X'20', 370 synchronous I/O 38
X'210', retrieve device information
condition codes 197
program checks 197
X'214' 1019
X'218', Real CPU identification 197
X'238', Time-Based Unique Identifiers 201
X'24', device types and features 39
X'240' 1026
X'244' 1028
X'248', copy-to-primary service 202
X'248', use with address spaces 786
X'250', block I/O operations 203
X'254', access real subsystem 1032
X'258', page-reference services 216
X'260', access virtual machine information 222
X'268', 370 accommodation services 225
X'26C', access system information 226
X'270', pseudo timer extended 263
X'274', set timezone interrupt flag 265
X'27C', product enablement verification 266
X'28', dynamic channel program modification 41
X'288', Control Virtual Machine Time Bomb 268
X'290', perform privileged spool functions 270
X'2A8', Network Diagnose 274
X'2AC', HCD dynamic I/O 1076
X'2C0', HMC data source load 1079
X'2CC', SSI Diagnose 288
X'2E0', SYSEVENT Query Virtual Server (QVS) 290
X'2FC', obtain certain guest performance data 291
X'34', read system dump spool file 43
X'3C', activate CP directory 44
X'44', voluntary time slice end 46
X'48', second level SVC 76 46
X'4C' 718
X'4C', generate accounting records 47
X'54', control the PA2 function key 50
DIAGNOSE code *(continued)*
X'58', 3270 virtual console interface 51
X'5C', error message editing 60
X'60', determine virtual machine storage size 62
X'64', named saved segment manipulation
FINDSEG function 65
FINDSEG function (64-bit) 73
LOADNOLY function 66
LOADNSHR function 64
LOADNSHR function (64-bit) 73
LOADSHR function 63
LOADSHR function (64-bit) 73
PURGESEG function 65
SEGEXT function 66
SEGEXT function (64-bit) 73
X'68', VMCF function 987, 989
X'70', time-of-day clock accounting interface 80
X'74', saving and loading an image library file 82
X'7C', logical device support facility 84
X'84', directory update in-place
abend codes 102
condition codes 102
operations 93
return codes 103
X'88', validate user authorization 105
X'8C', access 3270 display information
program exceptions 111
return codes 111
X'90', read symbol table 112
X'94', VMDUMP and symptom record service
applications using SR option 962
condition codes 120
dump address list 116
parameters different than VMDUMP 115
reason codes 123
return codes 120
SR parameter 115
SRDW parameter 116
supported parameters 113
symptom record processing return codes 122
symptom record reporting 961
X'98', real I/O
Block DiagnoseX'98' 126
program exceptions 132
return codes 125, 126
X'9C', voluntary time slice 133
X'A0', obtain ACI groupname
program exceptions 135
X'A4', synchronous I/O operations
condition codes 142, 143
HCPSBIOP (synchronous block I/O parameter list) 136
return codes 142, 143
X'A8', synchronous I/O operations
HCPSGIOP (synchronous general I/O parameter list) 144
return codes and condition codes 148
X'B0', access re-IPL data
program exceptions 151
X'B4', read, write, and erase the virtual printer external attribute buffer
program exceptions 153
return codes 152
X'B8', spool file external attribute buffer manipulation

DIAGNOSE code *(continued)*
X'B8', spool file external attribute buffer manipulation *(continued)*
program checks 155
return codes 154
X'BC', open and query spool file characteristics
condition codes and return codes 158
subcode X'0000' 155
subcode X'0004' 157
X'C8', set language
program checks 160
return codes 159
X'CC', save message repository
program checks 161
return codes 160
X'D0', volume serial support
condition codes and return codes 162
X'D4', set alternate user ID
condition codes 283
program checks 164
return codes 109, 164
X'D8', reading spool file blocks on the system spool file queues
condition codes 166
program exceptions 166
X'DC', control application monitor data generation
condition codes and return codes 170
program exceptions 171
X'E0', system trace file interface 171
X'E4', define a full-pack overlay minidisk 174
X'E4', return minidisk real device information
condition codes and return codes 183
X'EC', query GUEST trace status
condition codes and return codes 186
program checks 186
X'F8', spool file origin information
program exceptions 189
return codes 188
DIAGNOSE code X'10'
protection exception 25
DIAGNOSE codes, reserved
X'040' 1015
X'214' 1019
X'23C' 1021
X'240' 1026
X'244' 1028
X'254' 1032
X'25C' 1040
X'264' 1044
X'278' 1046
X'280' 1049
X'29C' 1056
X'2A0' 1062
X'2A4' 1073
X'2AC' 1076
X'2C0' 1079
X'2C4' 1080
X'E0' 1015
DIAGNOSE instruction
370 accommodation services, X'268' 225
370 synchronous I/O, X'20' 38
access 3270 display information, X'8C' 110
access re-IPL data, X'B0' 149
access real subsystem, X'254' 1032
access system information, X'26C' 226
DIAGNOSE instruction *(continued)*
access virtual machine information, X'260' 222
activate CP directory, X'3C' 44
bit map fields for DIAGNOSE code X'00' 13
block I/O operations, X'250' 203
condition codes and return codes 9
control application monitor data generation,X' DC' 166
control blocks used by 969
control the PA2 function key, X'54' 50
Control Virtual Machine Time Bomb, X'288' 268
copy-to-primary service, X'248' 202
data space considerations 786
define a full-pack overlay minidisk, X'E4' 174
determine virtual machine storage size, X'60' 62
device types and features, X'24' 39
directory update in-place, X'84' 91
dynamic channel program modification, X'28' 41
dynamic I/O, HCD, X'2AC' 1076
error message editing, X'5C' 60
example of coding 4
execution in access-register mode 5
generate accounting records, X'4C' 47
HMC data source load, X'2C0' 1079
how address spaces are selected 7
how addresses are processed 5
how error conditions are reported 7
input spool file manipulation, X'14' 25
instruction format 3
logical device support facility, X'7C' 84
macro format 4
mode restrictions 5
named saved segment manipulation, X'64' 62
obtain ACI groupname, X'A0' 134
obtain certain guest performance data, X'2FC' 291
open and query spool file characteristics, X'BC' 155
page-reference services, X'258' 216
privileged spool functions, X'290' 270
product enablement verification, X'27C' 266
pseudo timer extended, X'270' 263
pseudo timer, X'0C' 22
query GUEST trace status, X'EC' 185
read symbol table, X'90' 112
read system dump spool file, X'34' 43
read, write, and erase the virtual printer external attribute buffer, X'B4' 151
reading spool file blocks on the system spool file queues, X'D8' 164
Real CPU identification, X'218' 197
real I/O, X'98' 124
release pages, X'10' 24
retrieve device information, X'210' 189
return minidisk real device information, X'E4' 174
save message repository, X'CC' 160
saving and loading an image library file, X'74' 82
second level SVC 76, X'48' 46
set alternate user ID, X'D4' 162
set language, X'C8' 159
set timezone interrupt flag, X'274' 265
spool file external attribute buffer manipulation, X'B8' 153
spool file origin information, X'F8' 187
standard DASD I/O, X'18' 36
store extended-identification code, X'00' 13
synchronous I/O operations, X'A4' 135

DIAGNOSE instruction *(continued)*
synchronous I/O operations, X'A8' 144
SYSEVENT Query Virtual Server (QVS), X'2E0' 290
system trace file interface, X'E0' 171
Time-Based Unique Identifiers, X'238' 201
time-of-day clock accounting interface, X'70' 80
validate user authorization, X'88' 105
virtual console function, X'08' 19
VMCF function, X'68'
data transfer error codes 1009
return codes 1007
VMCPARM parameter list 1000
VMDUMP and symptom record service, X'94' 113
volume serial support, X'D0' 161
voluntary time slice end, X'44' 46
voluntary time slice, X'9C' 133
directory
authorization for IUCV 301
control statement for IUCV 297, 299, 303, 320, 325
entries in IUCV 301, 320, 326
update in-place, DIAGNOSE code X'84' 91
VMUDQ macro 875
directory operations
ACCOUNT 93
CPU 94
DATEFMT 94
DISTRIB 94
EDITCHAR 95
IACCOUNT 95
IPL 95
LOGPASS 96
MACHINE 96
MAXSTOR 96
MDISK 97
OPTIONS 98
PRIORITY 98
PRIVLEGE 98
RMDISK 99
SCREEN 100
SPOOLF 100
STORAGE 101
TACCOUNT 102
XAUTOLOG 102
XSTORE 102
directory query, DIAGNOSE code X'25C' 1040
DISABLE function of Spool system service 763
double-byte character set (DBCS) 58
dropping addressability to an address space 789
DSECT, IPARMLX 418
dump
address list, DIAGNOSE Code X'094' 116
Dump Viewing Facility and DIAGNOSE code X'94' 113
dynamic channel program modification, DIAGNOSE code X'28' 41

E

editing error messages, DIAGNOSE code X'5C' 60
effects of mapping 825
ENABLE function of Spool system service 763
entry point
to ACI CP modules
HCPPWAPF 607
HCPRPEEP 602
entry point *(continued)*
to ACI CP modules *(continued)*
HCPRPEPX 602
HCPRPESG 604
HCPRPGPH 606
HCPRPICN 593
HCPRPIIL 593
HCPRPIQS 593
HCPRPIRA 594
HCPRPIRM 593
HCPRPISV 593
HCPRPWEP 596, 599
HCPRPWPR 598
HCPRWACP 607
erasing the virtual printer XAB 151
error codes, APPC/VM 399
error conditions state table, APPC/VM 407
error log GDS variable format 494, 513
error logging system service (*LOGREC)
IUCV ACCEPT 729
IUCV CONNECT 729
IUCV SEVER 729
error message editing, DIAGNOSE code X'5C' 60
ESA/XC address spaces
creating 800
definition 783
destroying 804
dropping addressability 789
identification 798
identification token (ASIT) 798
isolating a shared 789
mapping minidisks to 790
name 798
overview 783
permitting another user to access 787
primary 785
querying information 812
restoring to private state 806
selection of 7
states 798, 806
support 784
ESA/XC architecture 784
ESA/XC macros
ADRSPACE 797
ALSERV 815
coding 793
DEFWORKA 823
MAPMDISK 824
overview 783
PFAULT 852
REFPAGE 863
use considerations 786
using HELP for 793
ESM (External Security Manager)
CP interface
*RPI system service 593
ACIPARMS formats 639
ACIPARMS general format 622
called by ESM 602
checking ACI Security bits 611
commands that support calls to ACI 609
data area 606
DIAGNOSE code X'A0' processor 599
DIAGNOSE codes that support calls to ACI 609

ESM (External Security Manager) *(continued)*
CP interface *(continued)*
HCPA0LBK format 617
HCPA0UBK general format 620
HCPRPD exit module 599
HCPRPE module 602
HCPRPF module 606
HCPRPG module 606
HCPRPI exit module 593
HCPRPL module 606
HCPRPP module 606
HCPRPW exit module 596
HCPRWA module 607
IUCV interface 593
logon password prompting routine 598
logon password verification routine 596
overview 592
request services from ESM 594
security process 592
work area 607
establish addressability to an address space 787
establish an APPC conversation 573
establish communication
account system service 699
asynchronous CP command response system service 719
DASD block I/O system service 721
error logging system service 729
identify system service 731
message system service 739
signal system service 747
spool system service 752, 765
symptom system service 773
examine host storage, DIAGNOSE code X'04' 16
example of
coding the DIAGNOSE instruction 4
creating a remote work area, data spaces 796
forcing unique work areas, data spaces 795
format of PIP variable 423
IUCV virtual machine communication 302
nonreentrant program, data spaces 794
reentrant program, data spaces 794
exception
addressing-capability 1017, 1018
ADRSPACE macro 799
ALEN translation 1017–1019
ALSERV macro 816
APPCVM
CONNECT 428
QRYSTATE 450
RECEIVE 460
SENDCNF 469
SENDCNFD 474
SENDDATA 486
SENDERR 498
SENDREQ 503
SETMODFY 507
SEVER 517
DIAGNOSE code 7
DIAGNOSE code X'00' 16
DIAGNOSE code X'04' 19
DIAGNOSE code X'08' 22
DIAGNOSE code X'0C' 23
DIAGNOSE code X'10' 25
exception *(continued)*
DIAGNOSE code X'14' 35
DIAGNOSE code X'210' 197
DIAGNOSE code X'238' 201
DIAGNOSE code X'248' 203
DIAGNOSE code X'250' 214
DIAGNOSE code X'254' 1039
DIAGNOSE code X'258' 222
DIAGNOSE code X'260' 224
DIAGNOSE code X'268' 226
DIAGNOSE code X'26C' 263
DIAGNOSE code X'27C' 268
DIAGNOSE code X'288' 269
DIAGNOSE code X'2A4' 1075
DIAGNOSE code X'2AC' 1078
DIAGNOSE code X'2E0' 291
DIAGNOSE code X'34' 44
DIAGNOSE code X'3C' 45
DIAGNOSE code X'4C' 50
DIAGNOSE code X'5C' 61
DIAGNOSE code X'64' 79
DIAGNOSE code X'68' 88, 988
DIAGNOSE code X'70' 81
DIAGNOSE code X'74' 83
DIAGNOSE code X'84' 104
DIAGNOSE code X'88' 109
DIAGNOSE code X'8C' 111
DIAGNOSE code X'90' 113
DIAGNOSE code X'98' 132
DIAGNOSE code X'9C' 134
DIAGNOSE code X'A0' 135
DIAGNOSE code X'A4' 143
DIAGNOSE code X'A8' 148
DIAGNOSE code X'B0' 151
DIAGNOSE code X'B4' 153
DIAGNOSE code X'B8' 155
DIAGNOSE code X'BC' 158
DIAGNOSE code X'C8' 160
DIAGNOSE code X'CC' 161
DIAGNOSE code X'D4' 109, 164
DIAGNOSE code X'D8' 166
DIAGNOSE code X'DC' 171
DIAGNOSE code X'E0' 172, 1017, 1018
DIAGNOSE code X'E4' 184
DIAGNOSE code X'EC' 186
DIAGNOSE code X'F8' 189
IUCV
ACCEPT 322, 528
CONNECT 329, 532
DECLARE BUFFER 332, 536
DESCRIBE 335, 539
INTERRUPT POLL 338
IPOLL 542
PURGE 343
QUERY 344, 546
QUIESCE 346
RECEIVE 351
REJECT 354
REPLY 358
RESUME 363
RETRIEVE BUFFER 365, 548
SEND 370
SET CONTROL MASK 373, 552
SET MASK 375, 555

exception *(continued)*
IUCV *(continued)*
SEVER 377, 559
TEST COMPLETION 382, 564
TEST MESSAGE 383, 566
MAPMDISK macro 826
PFAULT macro 855
REFPAGE macro 864
VMUDQ macro 879
exception condition
addressing-capability 1019
DIAGNOSE code X'E0' 1019
exit results from the SEGEXT function, normal 77
exit with error results from DIAGNOSE code X'64' 78, 79
exits
CP
access control interface 591
external security manager 591
HCPRPD module 599
HCPRPI module 593
HCPRPW module 596
external security manager
CP 591
HCPRPD module 599
HCPRPI module 593
HCPRPW module 596
installation-wide
CP access control interface 591
HCPRPD module 599
HCPRPI module 593
HCPRPW module 596
extended data, connection complete 429, 430
extended data, connection pending 434, 438
Extended Spool File Block 978
extended-identification code, DIAGNOSE code X'00' 13
extension for connection parameter list 417, 418
extension, interrupt buffer 535
extent-list format, MAPMDISK 839
External Attribute Buffer 980
external attribute buffer (XAB) 151, 153
External Attribute Buffer (XAB)
format 979
external interrupt
APPC/VM
buffer 388
connection complete 389, 426
connection pending 389
creating buffers for 388
function complete 390
message pending 389, 489
request-to-send 389
SENDREQ 389
sever 389, 519
code X'2402', logical device 88
code X'2603' subcode X'02' from PFAULT 854
IUCV
avoiding 301
connection complete 322
connection pending 329
connection quiesced 347
connection resumed 364
connection severed 378
control 300
control, enabling or disabling 301
external interrupt *(continued)*
IUCV *(continued)*
enabling or disabling 300
functions controlling 307
message 300
message complete 358
message pending 370
sever 519
MAPMDISK SAVE
code X'2603' subcode X'01' 850
VMCF
message header 1004
External interruption, access real subsystem 1040
external interruption, block I/O 215
External Security Manager (ESM) 591
EXTRACT AND SET STORAGE ATTRIBUTES 888
extract XLINK control blocks, DIAGNOSE code X'278' 1046

F

fault resolution flag 788
fcode field of IUCV CONNECT 530, 731
features and types for devices, DIAGNOSE code X'24' 39
fetch-protection override (FPO) 9
fields
in VRDCBLOK DSECT 190
FILL parameter (APPC)
of RECEIVE_AND_WAIT 582
FINDNSSA operation code 73
FINDNSSA operation code (64-bit) 74
FINDSEG function (64-bit), DIAGNOSE code X'64' 73
FINDSEG function, DIAGNOSE code X'64' 65
FINDSEGA operation code 69
FINDSEGA operation code (64-bit) 74
FINDSKEL operation code 69
FINDSKEL operation code (64-bit) 74
FINDSKEL or FINDSEG operation, format of user-supplied areas 70
FINDSKEL, FINDSEGA, or FINDNSSA operation (64-bit), format of user-supplied areas 74
FINDSPACE operation (64-bit), format of user-supplied areas 73
FINDSPACE operation code 68
FINDSPACE operation code (64-bit) 73
FINDSPACE operation, format of user-supplied areas 68
FLAG field
of CONNECT request (*IDENT) 530, 731
FLUSH, APPC verb 581
FMH5 (Functional Management Header 5)
description 438
FORCE command 700
forcing unique work areas example, data spaces 795
foreign language, set 159
format of
PIP variable, example 423
user-supplied areas for FINDSKEL or FINDSEG operation 70
user-supplied areas for FINDSKEL, FINDSEGA, or FINDNSSA operation (64-bit) 74
user-supplied areas for FINDSPACE operation 68
user-supplied areas for FINDSPACE operation (64-bit) 73
user-supplied areas for SEGEXT function 66
user-supplied output area — 64-bit member list 74

format of *(continued)*
user-supplied output area — member list 71
format of a block I/O entry 209
format of symptom record 961
formatting IUCV and APPC/VM parameter lists with MF=L 395
FPO (fetch-protection override) 9
FTP
management functions
X'2C4' 1080
FTP services, DIAGNOSE code X'2C4' 1080
full-duplex communications 567
full-pack overlay minidisk 174
function complete interrupts
APPCVM RECEIVE 456
APPCVM SENDCNF 468
APPCVM SENDCNFD 473, 474
APPCVM SENDDATA 482
APPCVM SENDERR 496
APPCVM SEVER 515
connection complete interrupt 322
message complete interrupt 358
function completion
APPCVM CONNECT 429
IUCV ACCEPT 321, 322
IUCV CONNECT 326, 328
IUCV DCLBFR 332, 537
IUCV DESCRIBE 334, 335, 539
IUCV IPOLL 338
IUCV PURGE 341, 343
IUCV QUERY 344
IUCV QUIESCE 346
IUCV RECEIVE 350, 351
IUCV REJECT 353
IUCV REPLY 357, 358
IUCV RESUME 363
IUCV RTRVBFR 365, 549
IUCV SEND 369
IUCV SETCMASK 373
IUCV SETMASK 375, 555
IUCV SEVER 377, 559
IUCV TESTCMPL 380
IUCV TESTMSG 383
SENDERR 499
SENDREQ 503
SETCMASK 552
TESTCMPL 565
TESTMSG 566

G

gateways
managing, using *IDENT 731, 737
GDS (General Data Stream) log data variable format 513
GDS (General Data Stream) Log Data variable format 494
General data stream (GDS) Log Data variable format 494
General Data Stream (GDS) log data variable format 513
general description of IUCV functions 306
generate accounting records, DIAGNOSE code X'4C' 47
generic *SPL interface 765
GET_ATTRIBUTES, APPC verb
mapped with APPC/VM 581
GIDs
X'280' 1049
GIDs *(continued)*
X'29C' 1056
global resources
managing 393, 731
revoking your own 394
virtual machines connecting to 394
group, virtual machine 747
guest list, STHYI instruction 941

H

HALT DEVICE (HDV) instruction 894, 901, 908
HALT I/O (HIO) instruction 894, 901, 908
HALT SUBCHANNEL (HSCH) instruction 903, 908
handling page faults asynchronously 852
handshaking
necessary conditions 854
page-fault 853
process 854
services, page-fault 852
HCD dynamic I/O, DIAGNOSE code X'2AC' 1076
HCPA0LBK control block (format) 617
HCPA0UBK control block (format) 620
HCPEXIT macro
HCPRPD module 602
HCPRPI module 596
HCPRPW module 598, 599
HCPRPD exit module
function 599
HCPRPDEP entry point
interface specifications 599
return codes 601
HCPRPE module 602
HCPRPF module 606
HCPRPG module 606
HCPRPI exit module
HCPRPICN entry point
function 593
interface specifications 593
HCPRPIIL entry point
function 593
interface specifications 593
HCPRPIQS entry point
function 593
interface specifications 593
HCPRPIRA entry point
function 594
interface specifications 594
HCPRPIRM entry point
function 593
interface specifications 593
HCPRPISV entry point
function 593
interface specifications 593
HCPRPL module 606
HCPRPP module 606
HCPRPWEP exit module
function 596
HCPRPWEP entry point
interface specifications 596
HCPRPWPR exit module
HCPRPWPR entry point
interface specifications 598
HCPRWA module

HCPRWA module *(continued)*
HCPPWAPF entry point
interface specifications 607
HCPSBIOP (synchronous block I/O parameter list) 136
HCPSGIOP (synchronous general I/O parameter list) 144
HDV (HALT DEVICE) instruction 894, 901, 908
HELP for APPCVM functions 411
HELP for CP macros 793
HELP for IUCV macro functions with APPC/VM 523
HIO (HALT I/O) instruction 894, 901, 908
HMC data source load, DIAGNOSE code X'2C0' 1079
host access-list entries 815
host access-list-controlled protection 10
host address space
definition 7
host page protection 10
host storage, examine 16
host-primary address space
access list 786
adding an ALE to an access list 787
how address spaces are selected 7
how addresses are processed 5, 297
how error conditions are reported
access exceptions 8
condition and return codes 9
program interruptions 7
storage protection mechanisms 9
HSCH (HALT SUBCHANNEL) instruction 903, 908
hybrid interruptions 907
hypervisor environment information, STHYI instruction 926

I

I/O operations (block) for DASD in standard CMS blocksize 203
I/O, standard DASD, DIAGNOSE code X'18' 36
identification of address spaces 798
IDENTIFY function of MAPMDISK macro 837
IDENTIFY function of VMCF
protocol, VMCF 995
identify system service (*IDENT)
connecting to 393, 529
establishing a connection 731
overview 394
processing requests to manage a resource 733
sever reason codes 736
severing connection to 394
image library file, saving and loading with DIAGNOSE code X'74' 82
IMMED value on APPCVM CONNECT 415
INFORMB function of REFPAGE macro 866
INFORML function of REFPAGE macro 872
initialize block I/O to a device 204
INITIATE function, logical device support facility
DIAGNOSE code X'7C' subcode X'00000001' 89
input parameter lists
APPCVM
CONNECT 416
QRYSTATE 447
RECEIVE 453
SENDCNF 466
SENDCNFD 471
SENDDATA 477
SENDERR 492
input parameter lists *(continued)*
APPCVM *(continued)*
SENDREQ 501
SETMODFY 506
SEVER 511
IPARML 304
IUCV
ACCEPT 321, 526
CONNECT 327, 531
DECLARE BUFFER 332, 535
DESCRIBE 334
INTERRUPT POLL 338, 541
PURGE 341
QUERY 544
QUIESCE 346
RECEIVE 349
REJECT 353
REPLY 356
RESUME 362
SEND 368
SET CONTROL MASK 372, 551
SET MASK 375, 554
SEVER 377, 557
TEST COMPLETION 380, 562
VMCF 1000
VMCPARM 1000
input spool file manipulation, DIAGNOSE code X'14' 25
INSERT STORAGE KEY (ISK) instruction 894, 902
instance of a data space 787
intermediate communications servers
accepting connections 525
making connections 441
parameters on IUCV ACCEPT 525
interrupt buffer extension 535
INTERRUPT POLL function of IUCV
condition codes 338
format 337
parameter list format 338
program exceptions 338
return codes 338
used with APPC/VM
condition codes 541
format 540
parameter descriptions 540
parameter list format 541
program exceptions 542
return codes 542
using 337
interrupt, external
APPC/VM 388
creating buffers for 388
IUCV
avoiding 301
control 300
control, enabling or disabling 301
enabling or disabling 300
functions controlling 307
message 300
message pending 370
interruption parameters 906
interrupts
APPC/VM 388, 573
connection complete 389, 426, 429
connection pending 389, 432

interrupts *(continued)*
disabling for 550, 553
enabling for 550, 553
function complete 390, 456, 468, 482, 496
message pending 389, 489
SENDREQ 389, 504
sever 389, 519
interval timer 894, 895, 903
IPARML
DASD block I/O system service 722
signal system service 747
IPARML COPY file 304, 307, 394
IPARMLX COPY file 418
IPARMLX DSECT 418
IPAUDIT field, description 396, 397
IPCODE field, description 396, 398
IPOLL function of IUCV
condition codes 338
format 337
parameter list format 338
program exceptions 338
return codes 338
used with APPC/VM
condition codes 541
format 540
parameter descriptions 540
parameter list format 541
program exceptions 542
return codes 542
using 337
IPRCODE field, description 396, 397
IPTYPE, external interrupt field
APPCVM function complete 457, 468, 473, 482, 496, 503, 516
connection complete 323, 427
connection pending 330
connection quiesced 347
connection resumed 364
connection severed 378, 427, 560
message complete 359
message pending 371, 539
request-to-send 539
to differentiate IUCV and APPC/VM interrupts 567
IPWHATRC field, description 396, 397
ISFC (Inter-System Facility for Communications)
accounting record 708
ISK (INSERT STORAGE KEY) instruction 894, 902
ISOLATE function of ADRSPACE macro 806
isolating a shared address space 789
IUCV (Inter-User Communication Vehicle)
interface to external security manager 593
IUCV (Inter-User Communications Vehicle)
authorization 301
basic communication functions 307
communication
account system service 699
asynchronous CP command response system service 719
DASD block I/O system service 721
error logging system service 729
example 302
identify system service 731
message system service 739
signal system service 747
IUCV (Inter-User Communications Vehicle) *(continued)*
communication *(continued)*
spool system service 752, 765
symptom system service 773
using control paths 305
using data in a buffer 302
using data in a parameter list 304
with CP system services 317
condition codes and return codes with APPC/VM 522
control paths 305
CP system services and list of user IDs 317
differences from APPC/VM 567
directory control statement 297
directory entries 301
external interrupt
avoiding 301
connection complete 322
connection pending 329
connection quiesced 347
connection resumed 364
connection severed 378
control 300
control, enabling or disabling 301
enabling or disabling 300
message 300
message complete 358
message pending 370
functions controlling external interrupts 307
general description of functions 306
introduction 297
invoking 306
macro description 306
macro functions for use in APPC/VM 521
messages
data transfer 298
identification 299
option on CP SET commands 739
parameters, specifying 306
paths
control 305
protocols
CONNECT/ACCEPT 311
SEND 314
SEND, logic flow 315
SEND/RECEIVE 312
SEND/RECEIVE, logic flow 312
SEND/RECEIVE/REPLY 313
SEND/RECEIVE/REPLY, logic flow 313
SEND/REPLY 315
SEND/REPLY, logic flow 315
receiving messages from the special message facility 1011
security considerations 301
sequence of functions 302, 304
shared functions that can be used in CMS 521
shared functions that should be avoided in 521
special message facility 1011
system services
identify 731
two-way data transfer 298
IUCV control paths 305
IUCV in a distributed environment 309
IUCV macro functions
ACCEPT

IUCV macro functions *(continued)*
ACCEPT *(continued)*
from the DASD block I/O system service 722
from the error logging system service 729
from the Symptom system service 773
parameter list format 321, 526
return codes 322, 527
using 319, 524
CONNECT
parameter list format 327, 531
return codes 328, 531
to DASD block I/O system service 721
to error logging system service 729
to signal system service 747
to symptom system service 773
using 324, 529
DECLARE BUFFER
parameter list format 332, 535
return codes 332, 536
using 331, 533
DESCRIBE
parameter list format 334, 539
return codes 335
using 334, 538
for use in APPC/VM 521
INTERRUPT POLL
IPOLL 542
parameter list format 338, 541
return codes 338
using 337, 540
PURGE
parameter list format 341
return codes 343
using 340
QUERY 344, 543
QUIESCE
parameter list format 346
return codes 346
using 345
RECEIVE
parameter list format 349
return codes 351
using 348
REJECT
parameter list format 353
return codes 353
using 352
related to APPC/VM 391
REPLY
parameter list format 356
return codes 358
using 355
RESUME
parameter list format 362
return codes 363
using 362
RETRIEVE BUFFER 365, 548
SEND
parameter list format 368
return codes 369
to the DASD block I/O system service 723
to the signal system service 749
using 366
SET CONTROL MASK
IUCV macro functions *(continued)*
SET CONTROL MASK *(continued)*
parameter list format 372
return codes 373
using 372
SET MASK
parameter list format 375, 554
return codes 375
using 374, 553
settings for optional parameters 395
SEVER
from the DASD block I/O system service 722
from the error logging system service 729
from the signal system service 750
from the symptom system service 773
parameter list format 377, 557
return codes 377, 559
using 376, 556
shared functions that can be used in CMS 521
shared functions that should be avoided in 521
terminology 306
TEST COMPLETION
parameter list format 380
return codes 382, 564
using 379, 562
TEST MESSAGE 383, 566
using HELP for 523
IUCV return codes
ACCEPT 322, 527
CONNECT 328, 531
DECLARE BUFFER (DCLBFR) 332, 536
DESCRIBE 335
INTERRUPT POLL (IPOLL) 338, 542
PURGE 343
QUIESCE 346
RECEIVE 351
REJECT 353
REPLY 358
RESUME 363
SEND 369
SET CONTROL MASK 373
SETMASK 375
SEVER 377, 559
TEST COMPLETION 382, 564

J

journal
accounting record 704

K

key-controlled protection
fetch-protection override (FPO) 9
host access-list-controlled protection 10
host page protection 10
low-address protection (LAP) 10
storage-protection override (SPO) 10

L

language, set 159
LAP (low-address protection) 10

leave the signal system service 750
LENGTH parameter (APPC)
of RECEIVE_AND_WAIT 582
of SEND_DATA 585
linemode console output
double-byte character set (DBCS) 58
single-byte character set (SBCS) 58
LINK command 700
list
dump address 116
list form of ordered paging-referencing 864
list format for macros
formatting parameter lists 395
IUCV DECLARE BUFFER 331
list of CP system services and user IDs 317
lists, address 307, 453, 479, 493, 512
LOAD PSW (LPSW) instruction 894, 904
loading and saving an image library file, DIAGNOSE code X'74' 82
LOADNOLY function, DIAGNOSE code X'64' 66
LOADNSHR function (64-bit), DIAGNOSE code X'64' 73
LOADNSHR function, DIAGNOSE code X'64' 64
LOADSHR function (64-bit), DIAGNOSE code X'64' 73
LOADSHR function, DIAGNOSE code X'64' 63
local resources
managing 393, 731
revoking your own 394
virtual machines connecting to 394
locally known LU name 419
log data
specifying 512
log data GDS variable format 513
Log Data GDS variable format 494
LOG_DATA parameter (APPC)
of DEALLOCATE 580
of SEND_ERROR 586
logical device support facility
description 1013
DIAGNOSE code X'7C' 84
external interrupt code X'2402' 88
logical records
definition 480
description 478
format 478
length 480
LOGON command 700
logon password prompting routine 598
logon password verification routine 596
low-address protection (LAP) 10
LPSW (LOAD PSW) instruction 894, 904
LSTMDISK function of VMUDQ macro
parameter list 876
LU name, locally known 419
LU_NAME parameter (APPC)
of ALLOCATE 576

M

macro work areas 823
macroinstruction
CP
HCPEXIT 596, 598, 599, 602
macros
ADSR 961
macros *(continued)*
APPCVM
CONNECT 412
QRYSTATE 447
RECEIVE 451
SENDCNF 465
SENDCNFD 471
SENDDATA 475
SENDERR 490
SENDREQ 501
SEVER 509
data space
ADRSPACE 797
ALSERV 815
coding 793
DEFWORKA 823
MAPMDISK 824
PFAULT 852
preferred use 793
REFPAGE 863
using HELP for 793
data space considerations 786
DIAG 4
execution in access-register mode 5
IUCV
ACCEPT 319, 524
advantages of using 306
application considerations for virtual MP 308
condition codes and return codes 522
CONNECT 324, 529
DECLARE BUFFER 331, 533
DESCRIBE 334, 538
for use in APPC/VM 521
in a distributed environment 309
INTERRUPT POLL (IPOLL) 337, 540
PURGE 340
QUERY 344, 543
QUIESCE 345
RECEIVE 348
REJECT 352
related to APPC/VM 391
REPLY 355
RESUME 362
RETRIEVE BUFFER 365, 548
SEND 366
SET CONTROL MASK 372, 550
SET MASK 374, 553
SEVER 376, 556
shared function that can be used in CMS 521
shared function that should be avoided in 521
terminology 306
TEST COMPLETION 379, 562
TEST MESSAGE 383, 566
VMUDQ 875
work areas 793
managing a resource 393
manipulation of saved segments with DIAGNOSE code X'64' 62
MAPMDISK macro
DECLARE function 828
DEFINE function 791, 829
extent-list format 839
IDENTIFY function 790, 837
list of functions 824

MAPMDISK macro *(continued)*
mapping-list format 832
REMOVE function 791, 843
SAVE function 791, 846
save-list format 848
MAPPED conversation type 415
mapped page of a data space 791
mapping APPC with APPC/VM
parameters and conditions 576
mapping minidisks to address spaces 790
mapping services, DIAGNOSE code X'244' 1028
mapping-list format, MAPMDISK macro 832
mapping, effects of 825
MDISK parameter list 876
member list, 64-bit, format of user-supplied output area 74
member list, format of user-supplied output area 71
merging collections
revoking gateways in 394
revoking resources in 394
message all system service (*MSGALL) 741
message complete external interrupt, IUCV 358
message editing, error, with DIAGNOSE code X'5C' 60
message examples, notation used in xxxvii
MESSAGE function of Spool system service 758
message pending interrupts
format 370, 489
message repository, saving, DIAGNOSE code X'CC' 160
message system service (*MSG) 739
messages
IUCV
data transfer 298
identification 299
messages, sending 478
MF=L parameter
formatting parameter lists 395
IUCV DECLARE BUFFER 331
minidisk cache, bypass 723, 725
minidisk real device information 174
minidisks to address spaces, mapping 790
mode name 419
MODE_NAME parameter (APPC)
of ALLOCATE 576
modes, types of
DIAGNOSE code restrictions 5
MODIFY SUBCHANNEL (MSCH) instruction 903, 908
modules for ACI
HCPRPD 591, 599
HCPRPF 591
HCPRPG 591, 606
HCPRPI 591, 593
HCPRPL 606
HCPRPPF 606
HCPRPW 591, 596
HCPRWA 591, 607
MSCH (MODIFY SUBCHANNEL) instruction 903, 908
multiple block I/O parameter list 725
multiple chained block I/O 725

N

named saved segment manipulation with DIAGNOSE code X'64' 62
national language, set 159
Network Diagnose, DIAGNOSE code X'2A8' 274
non-CMS environment 569
non-CMS environment, shared IUCV function for use in 521
NONE synchronization level 414
nonreentrant program example, data spaces 794
normal exit results from the SEGEXT function 77
not programming interfaces
DIAGNOSE code 1015
notation used in message and response examples xxxvii
NOTIFY function of Spool system service 765
notifying CP of future reference patterns 791

O

obtain ACI groupname, X'A0' 134
obtain certain guest performance data, DIAGNOSE code X'2FC' 291
online HELP for APPCVM functions 411
online HELP for CP macros 793
online HELP for IUCV macro functions with APPC/VM 523
open and query spool file characteristics with DIAGNOSE code X'BC' 155
operand exception
DIAGNOSE code X'254' 1039
DIAGNOSE code X'A4' 144
DIAGNOSE code X'A8' 148
general definition 8
operation exception
APPCVM CONNECT 428
APPCVM QRYSTATE 450
APPCVM RECEIVE 460
APPCVM SENDCNF 469
APPCVM SENDCNFD 474
APPCVM SENDDATA 486
APPCVM SENDERR 498
APPCVM SENDREQ 503
APPCVM SETMODFY 508
APPCVM SEVER 517
DIAGNOSE code X'98' 132
general definition 7
IUCV ACCEPT 322, 528
IUCV CONNECT 329, 532
IUCV DECLARE BUFFER 333, 536
IUCV DESCRIBE 336, 539
IUCV INTERRUPT POLL 338, 542
IUCV PURGE 343
IUCV QUERY 344, 546
IUCV QUIESCE 346
IUCV RECEIVE 351
IUCV REJECT 354
IUCV REPLY 358
IUCV RESUME 363
IUCV RETRIEVE BUFFER 365, 548
IUCV SEND 370
IUCV SET CONTROL MASK 373, 552
IUCV SET MASK 375, 555
IUCV SEVER 378, 559
IUCV TEST COMPLETION 382, 564
IUCV TEST MESSAGE 383, 566
operations of DIAGNOSE code X'84' 93
order a file to the front of a queue 28
order of functions in IUCV 302, 304
ordered paging-referencing
block form 863
list form 864

output parameter lists
APPCVM
CONNECT 426
QRYSTATE 448
RECEIVE 456
SENDCNF 468
SENDCNFD 473
SENDDATA 482
SENDERR 496
SENDREQ 503
SETMODFY 507
SEVER 515
IPARML 304
IUCV
ACCEPT 321, 527
CONNECT 328, 531
DECLARE BUFFER 332, 536
DESCRIBE 335, 539
PURGE 341
QUERY 545
QUIESCE 346
RECEIVE 350
REJECT 353
REPLY 357
RESUME 363
SEND 369
SET CONTROL MASK 373
SET MASK 375
SEVER 377, 558
TEST COMPLETION 380, 563
SEVER (APPCVM) 515
VMCF 1000
VMCPARM 1000
overview
370 Accommodation Facility 893
APPC/VM assembler interface 387
DIAGNOSE instruction 3
VM data spaces 783

P

PA2 function key, DIAGNOSE code X'54' 50
page-fault handshaking
necessary conditions 854
page-fault 853
process 854
services, page-fault 852
page-fault-cancel function 219
page-fault-token function 216
page-reference services 216
page-reference-inform function 219
paging-referencing, ordered
block form 863
list form 864
parameter addressability 394
parameter list
ACIPARMS
formats 639
function 592
general format 622
when created by CP 592
APPCVM
CONNECT 416
formatting with MF=L 395
parameter list *(continued)*
APPCVM *(continued)*
QRYSTATE 447, 448
reading 395
RECEIVE 453, 456
SENDCNF 466, 468
SENDCNFD 471, 473
SENDDATA 477, 482
SENDERR 492, 496
SENDREQ 501, 503
SETMODFY 506, 507
SEVER 511, 515
DIAGNOSE code
X'D8', DD8PARM0 DSECT 165
X'E4' 176, 179, 182
IPARML 304, 307
IUCV
ACCEPT 321, 526
CONNECT 327, 531
DECLARE BUFFER 332, 535
DESCRIBE 334
formatting with MF=L 395
INTERRUPT POLL 338, 541
parameter list data 304
PURGE 341
QUERY 544
QUIESCE 346
reading 395
RECEIVE 349
REJECT 353
REPLY 356
RESUME 362
SEND 368
SET CONTROL MASK 372, 551
SET MASK 375, 554
SEVER 377, 557
TEST COMPLETION 380, 562
LSTMDISK function 876
MDISK 876
multiple block I/O 725
VMCF 1000
VMCPARM 1000
parameters for communication servers on IUCV ACCEPT 525
parameters unique to DIAGNOSE code X'94' 115
parameters, APPC
on ALLOCATE 576
on CONFIRM 578
on CONFIRMED 579
on DEALLOCATE 580
on RECEIVE_AND_WAIT 582
on REQUEST_TO_SEND 584
on SEND_DATA 585
on SEND_ERROR 586
parameters, IUCV, specifying 306
partner, communications
connecting to 432
receiving from 463
sending to (SENDCNF) 470
sending to (SENDCNFD) 474
sending to (SENDDATA) 488
sending to (SENDERR) 500
sending to (SENDREQ) 504
severing from 519
severing paths 549

passwords
replacing directory stanza 91
paths
APPC/VM 387
IUCV 297
IUCV control 305
pending interrupt, message 370, 489
pending page release, DIAGNOSE code X'214' 1019
performance
APPC/VM 534
PERMIT function of ADRSPACE macro 808
permitting another user to access an address space 787
PFAULT macro
CANCEL function 857
DECLARE function 858
handling page faults asynchronously 852
list of functions 853
TOKEN function 852, 859
PIP parameter of APPC ALLOCATE 577
PIP variable
example format 423
handling 445
receiving 451
specifying 422
pool-relative block numbers 824
position a spool file to the designated record 30
POSIX IDs
process services 1073
querying 1062
setting 1049, 1056
POSIX process ID (PID) services, DIAGNOSE code X'2A4' 1073
Prepare_Received conversation state 405
prepare_received state 405
PREPARE_TO_RECEIVE, APPC verb 581
PRESENT function, logical device support facility
DIAGNOSE code X'7C' subcode X'00000003' 90
primary address space
access list 786
adding an ALE to an access list 787
priority messages 996, 999
PRIORITY option of VMCF AUTHORIZE 996
PRIORITY option of VMCF SEND/RECV 999
privilege classes
accounting record 710, 713
privilege-operation exception
DIAGNOSE code X'00' 16
DIAGNOSE code X'04' 19
DIAGNOSE code X'08' 22
DIAGNOSE code X'0C' 23
DIAGNOSE code X'10' 25
DIAGNOSE code X'14' 35
DIAGNOSE code X'210' 197
DIAGNOSE code X'238' 202
DIAGNOSE code X'248' 203
DIAGNOSE code X'254' 1039
DIAGNOSE code X'260' 225
DIAGNOSE code X'268' 226
DIAGNOSE code X'26C' 263
DIAGNOSE code X'2A4' 1075
DIAGNOSE code X'34' 44
DIAGNOSE code X'4C' 50
DIAGNOSE code X'5C' 62
DIAGNOSE code X'64' 80
privilege-operation exception *(continued)*
DIAGNOSE code X'68' 989
DIAGNOSE code X'70' 82
DIAGNOSE code X'74' 84
DIAGNOSE code X'7C' 88
DIAGNOSE code X'84' 104
DIAGNOSE code X'88' 110
DIAGNOSE code X'8C' 112
DIAGNOSE code X'90' 113
DIAGNOSE code X'98' 132
DIAGNOSE code X'9C' 134
DIAGNOSE code X'A0' 135
DIAGNOSE code X'A4' 144
DIAGNOSE code X'A8' 148
DIAGNOSE code X'B0' 151
DIAGNOSE code X'B4' 153
DIAGNOSE code X'B8' 155
DIAGNOSE code X'BC' 159
DIAGNOSE code X'C8' 160
DIAGNOSE code X'CC' 161
DIAGNOSE code X'D4' 164
DIAGNOSE code X'D8' 166
DIAGNOSE code X'DC' 171
DIAGNOSE code X'E4' 184
DIAGNOSE code X'EC' 186
DIAGNOSE code X'F8' 189
process, handshaking 854
product enablement verification, DIAGNOSE code X'27C' 266
program exception
ADRSPACE macro 799
ALSERV macro 816
APPCVM
CONNECT 428
QRYSTATE 450
RECEIVE 460
SENDCNF 469
SENDCNFD 474
SENDDATA 486
SENDERR 498
SENDREQ 503
SETMODFY 507
SEVER 517
DIAGNOSE code 7
DIAGNOSE code X'00' 16
DIAGNOSE code X'04' 19
DIAGNOSE code X'08' 22
DIAGNOSE code X'0C' 23
DIAGNOSE code X'10' 25
DIAGNOSE code X'14' 35
DIAGNOSE code X'210' 197
DIAGNOSE code X'238' 201
DIAGNOSE code X'248' 203
DIAGNOSE code X'250' 214
DIAGNOSE code X'254' 1039
DIAGNOSE code X'258' 222
DIAGNOSE code X'260' 224
DIAGNOSE code X'268' 226
DIAGNOSE code X'26C' 263
DIAGNOSE code X'27C' 268
DIAGNOSE code X'288' 269
DIAGNOSE code X'2A4' 1075
DIAGNOSE code X'2AC' 1078
DIAGNOSE code X'2E0' 291

program exception *(continued)*
DIAGNOSE code X'34' 44
DIAGNOSE code X'3C' 45
DIAGNOSE code X'4C' 50
DIAGNOSE code X'5C' 61
DIAGNOSE code X'64' 79
DIAGNOSE code X'68' 88, 988
DIAGNOSE code X'70' 81
DIAGNOSE code X'74' 83
DIAGNOSE code X'84' 104
DIAGNOSE code X'88' 109
DIAGNOSE code X'8C' 111
DIAGNOSE code X'90' 113
DIAGNOSE code X'98' 132
DIAGNOSE code X'9C' 134
DIAGNOSE code X'A0' 135
DIAGNOSE code X'A4' 143
DIAGNOSE code X'A8' 148
DIAGNOSE code X'B0' 151
DIAGNOSE code X'B4' 153
DIAGNOSE code X'B8' 155
DIAGNOSE code X'BC' 158
DIAGNOSE code X'C8' 160
DIAGNOSE code X'CC' 161
DIAGNOSE code X'D4' 109, 164
DIAGNOSE code X'D8' 166
DIAGNOSE code X'DC' 171
DIAGNOSE code X'E0' 172
DIAGNOSE code X'E4' 184
DIAGNOSE code X'EC' 186
DIAGNOSE code X'F8' 189
IUCV
ACCEPT 322, 528
CONNECT 329, 532
DECLARE BUFFER 332, 536
DESCRIBE 335, 539
INTERRUPT POLL 338
IPOLL 542
PURGE 343
QUERY 344, 546
QUIESCE 346
RECEIVE 351
REJECT 354
REPLY 358
RESUME 363
RETRIEVE BUFFER 365, 548
SEND 370
SET CONTROL MASK 373, 552
SET MASK 375, 555
SEVER 377, 559
TEST COMPLETION 382, 564
TEST MESSAGE 383, 566
MAPMDISK macro 826
PFAULT macro 855
REFPAGE macro 864
VMUDQ macro 879
protection exception
APPCVM CONNECT 428
APPCVM QRYSTATE 450
APPCVM RECEIVE 461
APPCVM SENDCNF 469
APPCVM SENDCNFD 474
APPCVM SENDDATA 486
APPCVM SENDERR 498
protection exception *(continued)*
APPCVM SENDREQ 503
APPCVM SETMODFY 508
APPCVM SEVER 517
DIAGNOSE code X'254' 1039
general definition 8
IUCV ACCEPT 322, 528
IUCV CONNECT 329, 532
IUCV DECLARE BUFFER 333, 536
IUCV DESCRIBE 336, 539
IUCV PURGE 343
IUCV QUERY 546
IUCV QUIESCE 347
IUCV RECEIVE 351
IUCV REJECT 354
IUCV REPLY 358
IUCV RESUME 363
IUCV SEND 370
IUCV SET CONTROL MASK 373, 552
IUCV SET MASK 375, 555
IUCV SEVER 378, 559
IUCV TEST COMPLETION 382, 564
protection mechanisms, storage 9
protocol
IUCV 311
VMCF
IDENTIFY 995
SEND 993
SEND/RECV 993
SENDX 994
pseudo timer, DIAGNOSE code X'0C' 22
PSW stealing programs 906
PURGE
IUCV function
condition codes 341
format 340
parameter descriptions 340
parameter list format 341
program exceptions 343
return codes 343
using 340
Spool system service function 765
PURGESEG function, DIAGNOSE code X'64' 65

Q

QRYSTATE function of APPCVM
condition codes 448
description 447
format 447
input parameter list format 447
output parameter list format 448
parameter description 447
program exceptions 450
state changes 450
QUERY
ADRSPACE function 812
IUCV function
condition codes 344
format 344
program exceptions 344
using 344
IUCV function used with APPC/VM
condition codes 544

QUERY *(continued)*
IUCV function used with APPC/VM *(continued)*
format 543
parameter description 543
parameter list format 544
program exceptions 546
using 543
query GUEST trace status with DIAGNOSE code X'EC' 185
query POSIX IDs, DIAGNOSE code X'2A0' 1062
query spool file characteristics with DIAGNOSE code X'BC' 155
QUIESCE
IUCV function
condition codes 346
connection quiesced interrupt 347
format 345
parameter descriptions 345
parameter list format 346
return codes 346
using 345
VMCF function 996, 1001

R

R2 APAR VM51599 1043
RACF (Resource Access Control Facility) 591
read next print spool file block 27
read next punch spool file block 28
read next spool file buffer 26
read symbol table, DIAGNOSE code X'90' 112
read system dump spool file, X'34' 43
read the last spool file buffer 30
read the next monitor spool file block 29
read the next monitor spool record 30
READ-SFBLOK function 759
READ-SPLINK function of Spool system service 762
READ-XAB function of Spool system service 761
read/write to DASD 206
reading IUCV and APPV/VM parameter lists 395
reading spool file blocks on the system spool file queues, DIAGNOSE code X'D8' 164
reading the virtual printer XAB 151
real CPU identification, DIAGNOSE code X'218' 197
real device feature code 41, 192
real device model 192
real device model number 41
real device type 41, 192
real device type class 41, 192
real I/O, DIAGNOSE code X'98' 124
reason codes for DIAGNOSE code X'94' 123
reason codes from *IDENT, sever 736
reason codes, SEVER
identify system service (*IDENT) 736
RECEIVE
IUCV function
condition codes 350
format 348
parameter descriptions 348
parameter list format 349
program exceptions 351
return codes 351
using 348
VMCF function 999, 1001
receive accounting records 700
RECEIVE function of APPCVM
addressing for 453
completion 462
condition codes 454
description 451
format 451
from communication partner 463
input parameter list format 453
mapped with APPC 582
output parameter list format 456
overview 393
parameter descriptions 451
program exceptions 460
return codes 454
state changes 461
state checks 461
receive signals from the signal system service 749
Receive state 388, 393, 403, 404
RECEIVE_AND_WAIT, APPC verb
abend conditions 584
mapped with APPC/VM 582
parameters 582
state changes 584
receiving data using APPC/VM 393
reentrant program example, data spaces 794
REFPAGE macro
DECLARE function 865
INFORMB function 866
INFORML function 872
list of functions 863
notifying CP of future reference patterns 791
REJECT
IUCV function
condition codes 353
format 352
parameter descriptions 352
parameter list format 353
program exceptions 354
return codes 353
using 352
VMCF function 997, 1001
relating APPC with APPC/VM 573
release pages, DIAGNOSE code X'10' 24
REMOVE
ALSERV function 821
MAPMDISK function 843
remove the block I/O environment 212
repeat the active file a specified number of times 28
REPLY
IUCV function
condition codes 357
format 355
message complete interrupt 358
parameter descriptions 355
parameter list format 356
return codes 358
using 355
VMCF function 999, 1001
reporting symptom records 961
repository, saving the message 160
REQUEST_TO_SEND_RECEIVED parameter (APPC)
of CONFIRM 578
of RECEIVE_AND_WAIT 583
of SEND_DATA 585

REQUEST_TO_SEND_RECEIVED parameter (APPC) *(continued)*
of SEND_ERROR 586
REQUEST_TO_SEND, APPC verb
abend conditions 584
mapped with APPC/VM 584
parameters 584
state changes 584
request-to-send interrupts 389
required VMCPARM fields for VMCF functions 1003
reserved DIAGNOSE codes 1015
RESET REFERENCE BIT (RRB) instruction 894, 902
Reset state 388, 403, 404
Resource Access Control Facility (RACF) 591
resource access verification 591
resource ID
connecting to virtual machine 733
on CONNECT request to *IDENT 530
using *IDENT 731
RESOURCE ID field
of CONNECT request (*IDENT) 731
RESOURCE parameter (APPC)
of ALLOCATE 577
of CONFIRM 578
of CONFIRMED 579
of DEALLOCATE 580
of RECEIVE_AND_WAIT 582
of REQUEST_TO_SEND 584
of SEND_DATA 585
of SEND_ERROR 586
resource pool list, STHYI instruction 950
resource pool member list, STHYI instruction 955
resources
managing local, global, or system 393
managing or revoking local, global, or system 731
requesting to manage - how *IDENT processes 733
revoking your own 394
virtual machines connecting to 394
response examples, notation used in xxxvii
restart an active file at the beginning 29
results of exit with error from DIAGNOSE code X'64' 78, 79
RESUME
IUCV function
condition codes 363
connection resumed interrupt 364
parameter descriptions 362
parameter list format 362
program exceptions 363
return codes 363
using 362
VMCF function 997, 1001
RESUME SUBCHANNEL (RSCH) instruction 903, 908
RETRIEVE BUFFER function of IUCV
condition codes 365
format 365
program interruptions 365
used in APPC/VM
completion 549
condition codes 548
format 548
program exceptions 548
state changes 548
using 548
using 365
retrieve device information, DIAGNOSE code X'210' 189
retrieve next file descriptor 34
return codes
ADRSPACE
CREATE function 802
DESTROY function 805
ISOLATE function 807
PERMIT function 811
QUERY function 814
ALSERV
ADD function 819
REMOVE function 822
APPC/VM 575
APPCVM
CONNECT 424
overview 396, 397
QRYSTATE 448
RECEIVE 454
SENDCNF 467
SENDCNFD 472
SENDDATA 480
SENDERR 495
SENDREQ 502
SETMODFY 506
SEVER 514
DIAGNOSE codes
general description 9
X'14' 35
X'18' 37
X'20' 39
X'218' 199
X'250' 212
X'254' 1038
X'28' 43
X'2A8' 283
X'2AC' 1076
X'34' 44
X'64' 78, 79
X'68' 988, 1003, 1007
X'74' 83
X'7C' 87
X'84' 103
X'88' 109
X'8C' 111
X'94' 120
X'94' for symptom records 122
X'98' 125, 126
X'A4' 142, 143
X'A8' 148
X'B4' 152
X'B8' 154
X'BC' 158
X'C8' 159
X'CC' 160
X'D0' 162
X'D4' 164
X'DC' 170
X'E4' 183
X'F8' 188
IUCV
ACCEPT 322
CONNECT 328
DCLBFR function 536
DECLARE BUFFER 332
DESCRIBE 335

return codes *(continued)*
IUCV *(continued)*
INTERRUPT POLL 338
IPOLL 542
IUCV ACCEPT 527
IUCV CONNECT 531
PURGE 343
QUIESCE 346
RECEIVE 351
REJECT 353
REPLY 358
RESUME 363
SEND 369
SET CONTROL MASK 373
SETMASK 375
SEVER 377, 559
TEST COMPLETION 382, 564
MAPMDISK
DEFINE function 834
IDENTIFY function 841
REMOVE function 844
SAVE function 849
multiple block I/O 726
PFAULT
CANCEL function 857
TOKEN function 862
REFPAGE
INFORMB function 869
INFORML function 873
single block I/O 723
VMCF 1007
VMUDQ
LSTMDISK function 879
return minidisk real device information, DIAGNOSE code X'E4' 174
RETURN_CODE parameter (APPC)
of ALLOCATE 577
of CONFIRM 578
of DEALLOCATE 580
of RECEIVE_AND_WAIT 583
of SEND_DATA 585
of SEND_ERROR 586
RETURN_CONTROL (WHEN_SESSION_ALLOCATED) parameter (APPC) 577
revoking
resources you do not own 394
your own resources 394
RRB (RESET REFERENCE BIT) instruction 894, 902
RSCH (RESUME SUBCHANNEL) instruction 903, 908

S

sample programs using the DASD Block I/O system service 981
SAVE function of MAPMDISK macro 846
save message repository, DIAGNOSE code X'CC' 160
save-list format, MAPMDISK macro 848
saved segment manipulation with DIAGNOSE code X'64' 62
saving and loading an image library file, DIAGNOSE code X'74' 82
second level SVC 76, DIAGNOSE code X'48' 46
sections of a symptom record 961
security
CP interface to external security manager
security *(continued)*
CP interface to external security manager *(continued)*
*RPI system service 593
ACI bits 609
ACI bits, setting of 610
ACI Security bits and calling the ESM 611
ACIPARMS formats 639
ACIPARMS general format 622
called by ESM 602
commands that support calls to ACI 609
data area 606
DIAGNOSE code X'A0' processor 599
DIAGNOSE codes that support calls to ACI 609
event, definition 609
HCPA0LBK general format 617
HCPA0UBK general format 620
HCPRPD exit module 599
HCPRPE module 602
HCPRPF module 606
HCPRPG module 606
HCPRPI exit module 593
HCPRPL module 606
HCPRPP module 606
HCPRPW exit module 596
HCPRWA module 607
IUCV interface 593
logon password prompting routine 598
logon password verification routine 596
overview 592
request services from ESM 594
security process 592
work area 607
external security manager
calling 611
CP interface 591
security considerations, IUCV 301
security subfield in an attach FMH5 for VM 439
SEGEXT function (64-bit), DIAGNOSE code X'64' 73
SEGEXT function operation codes 68
SEGEXT function, DIAGNOSE code X'64' 66
SEGEXT function, format of user-supplied areas 66
select a file for processing and read the next spool buffer 30
SELECT function of spool system service 754
SELECT function of Spool system service 765
select the next file not previously selected 31
SEND
IUCV function
condition codes 369
format 366
message pending interrupt 370
parameter descriptions 367
parameter list format 368
program exceptions 370
return codes 369
to the DASD block I/O system service 723
to the signal system service 749
to the spool system service 754, 765
using 366
protocol, VMCF 993
Spool system service function 764
VMCF function 998, 1001
send signals to the signal system service 749
Send state 388, 393, 403, 404
SEND_DATA, APPC verb

SEND_DATA, APPC verb *(continued)*
abend conditions 585
mapped with APPC/VM 585
parameters 585
state changes 585
SEND_ERROR, APPC verb
abend conditions 587
mapped with APPC/VM 586
parameters 586
state changes 587
SEND/RECV function of VMCF
protocol, VMCF 993
SENDCNF function of APPCVM
completion 470
condition codes 467
description 465
format 465
input parameter list format 466
mapped with APPC 578, 580–582
output parameter list format 468
parameter descriptions 465
program exceptions 469
return codes 467
state changes 470
state checks 469
to communication partner 470
SENDCNFD function of APPCVM
completion 474
condition codes 472
description 471
format 471
input parameter list format 471
mapped with APPC 579
output parameter list format 473
parameter descriptions 471
program exceptions 474
return codes 472
state changes 474
state checks 474
to communication partner 474
SENDDATA function of APPCVM
addressing for 479
completion 488
condition codes 480
description 475
format 475
input parameter list format 477
logical record format 478
mapped with APPC 581, 585
message pending interrupt 489
multiple 454, 480
output parameter list format 482
overview 393
parameter descriptions 475
program exceptions 486
return codes 480
setting up data 478
specifying buffers 479
state checks and changes 486
to communication partner 488
SENDERR function of APPCVM
addressing for 493, 512
completion 499
condition codes 495
SENDERR function of APPCVM *(continued)*
description 490
error codes 494
format 490
input parameter list format 492
mapped with APPC 586
output parameter list format 496
parameter descriptions 490
program exceptions 498
return codes 495
specifying log data 493
state checks and changes 499
to communication partner 500
sending data
details 478
overview 393
SENDREQ function of APPCVM
completion 503
condition codes 502
description 501
format 501
input parameter list format 501
interrupt 504
mapped with APPC 584
output parameter list format 503
parameter descriptions 501
program exceptions 503
return codes 502
state changes 503
state checks 503
to communication partner 504
SENDREQ interrupts
format 504
SENDX function of VMCF
protocol, VMCF 994
sense codes, currently-defined APPC/VM 403
sequence of functions in IUCV 302, 304
servers, communications
considerations for 441
set alternate user ID, DIAGNOSE code X'D4' 162
SET commands with IUCV options 739
SET CONTROL MASK function of IUCV
description 372
format 372
output from 373
parameter descriptions 372
parameter list format 372
program interruptions 373
return codes 373
used in APPC/VM
completion 552
condition codes 551
format 550
parameter descriptions 550
parameter list format 551
program exceptions 552
state changes 552
using 550
using 372
set language, DIAGNOSE code X'C8' 159
SET MASK function of IUCV
completion status (condition codes) 375
format 374
parameter descriptions 374

SET MASK function of IUCV *(continued)*
parameter list format 375
program interruptions 375
return codes 375
used in APPC/VM
completion 555
condition codes 554
format 553
parameter descriptions 553
parameter list format 554
program exceptions 555
state changes 555
using 553
using 374
set POSIX IDs - security values, DIAGNOSE code X'280' 1049
SET STORAGE KEY (SSK) instruction 894, 902
SET SYSTEM MASK (SSM) instruction 894
set timezone interrupt flag, DIAGNOSE code X'274' 265
set-POSIX-IDs services, DIAGNOSE code X'29C' 1056
SETMODFY function of APPCVM
completion 508
condition codes 506
description 505
format 505
input parameter list format 506
output parameter list format 507
parameter descriptions 505
program exceptions 507
return codes 506
state checks and changes 508
to communication partner 508
setting for optional parameters on APPC/VM functions 395
sever codes, APPC/VM 399, 513
SEVER function of APPCVM
completion 518
condition codes 514
description 509
error log GDS variable format 513
external interrupt format 519
format 509
from communication partner 519
input parameter list format 511
mapped with APPC 580
output parameter list format 515
overview 393
parameter descriptions 509
program exceptions 517
return codes 514
revoking your own resources 394
sever codes 513
specifying log data 512
state checks and changes 518
SEVER function of IUCV
*IDENT sever reason codes 736
condition codes 377
connection severed interrupt 378
format 376
from the DASD block I/O system service 722
from the error logging system service 729
from the signal system service 750
from the symptom system service 773
parameter descriptions 376
parameter list format 377
SEVER function of IUCV *(continued)*
program exceptions 377
return codes 377
used in APPC/VM
completion 559
condition codes 558
description 556
external interrupt 560
external interrupt format 519
format 556
overview 393
parameter descriptions 556
parameter list format 557
programming exceptions 559
return codes 559
state checks and state changes 559
to revoke a resource 394
using 556
using 376
sever interrupts
format 519
from IUCV SEVER 393
SEVER reason codes
identify system service (*IDENT) 736
sever reason codes from *IDENT 736
Sever state 388, 403, 405
severing connection to your partner
using APPC/VM 393
using APPCVM SEVER 393
using IUCV SEVER 393
SFBLOK 972
shared functions that can be used in CMS, IUCV 521
shared functions that should be avoided in CMS 521
SHUTDOWN command 700
signal system service (*SIGNAL)
connecting with 747
establishing communications with 747
leaving 750
receiving signals 749
sending signals 749
single block I/O 723
single-byte character set (SBCS) 58
SIO (START I/O) instruction 894, 901, 908
SIOF (START I/O FAST RELEASE) instruction 894, 901, 908
size of messages 478
SMSG command 1011
SNA (System Network Architecture)
relationship to APPC and APPC/VM 387
SNA/CCS (System Network Architecture/Console Communication Services)
accounting record 708
source code, adding your own 718
Special Message Facility
buffer length 1011
description 1011
introduction 1011
receiving messages through IUCV 1011
receiving messages through VMCF 1011
sending messages 1011
SMSG command 1011
special message flag (VMCPSMSG)
turning on or off 1011
special-operation exception
DIAGNOSE code X'68' 989

special-operation exception *(continued)*
DIAGNOSE code X'7C' 88
DIAGNOSE code X'A0' 135
SPECIFIC option of VMCF AUTHORIZE 996
specification exceptions
ADRSPACE macro 799
ALSERV macro 816
APPCVM CONNECT 428
APPCVM QRYSTATE 450
APPCVM RECEIVE 461
APPCVM SENDCNF 469
APPCVM SENDCNFD 474
APPCVM SENDDATA 486
APPCVM SENDERR 498
APPCVM SENDREQ 503
APPCVM SETMODFY 508
APPCVM SEVER 517
DIAGNOSE code X'00' 16
DIAGNOSE code X'04' 19
DIAGNOSE code X'08' 22
DIAGNOSE code X'0C' 23
DIAGNOSE code X'10' 25
DIAGNOSE code X'14' 35
DIAGNOSE code X'210' 197
DIAGNOSE code X'238' 202
DIAGNOSE code X'248' 203
DIAGNOSE code X'250' 215
DIAGNOSE code X'254' 1040
DIAGNOSE code X'260' 224
DIAGNOSE code X'268' 226
DIAGNOSE code X'26C' 263
DIAGNOSE code X'27C' 268
DIAGNOSE code X'288' 269
DIAGNOSE code X'2A4' 1076
DIAGNOSE code X'2C4' 1082
DIAGNOSE code X'2E0' 291
DIAGNOSE code X'34' 44
DIAGNOSE code X'4C' 50
DIAGNOSE code X'5C' 61
DIAGNOSE code X'64' 80
DIAGNOSE code X'70' 82
DIAGNOSE code X'74' 84
DIAGNOSE code X'7C' 88
DIAGNOSE code X'88' 110
DIAGNOSE code X'8C' 112
DIAGNOSE code X'90' 113
DIAGNOSE code X'98' 132
DIAGNOSE code X'A0' 135
DIAGNOSE code X'A4' 143
DIAGNOSE code X'A8' 148
DIAGNOSE code X'B0' 151
DIAGNOSE code X'B4' 153
DIAGNOSE code X'B8' 155
DIAGNOSE code X'BC' 159
DIAGNOSE code X'C8' 160
DIAGNOSE code X'CC' 161
DIAGNOSE code X'D4' 164
DIAGNOSE code X'D8' 166
DIAGNOSE code X'E0' 172
DIAGNOSE code X'E4' 184
DIAGNOSE code X'EC' 186
DIAGNOSE code X'F8' 189
general definition 8
IUCV ACCEPT 322, 528
specification exceptions *(continued)*
IUCV CONNECT 329, 532
IUCV DECLARE BUFFER 332, 536
IUCV DESCRIBE 335, 539
IUCV INTERRUPT POLL 338, 542
IUCV PURGE 343
IUCV QUERY 546
IUCV QUIESCE 346
IUCV RECEIVE 351
IUCV REJECT 354
IUCV REPLY 358
IUCV RESUME 363
IUCV SEND 370
IUCV SET CONTROL MASK 552
IUCV SET MASK 555
IUCV SEVER 378, 559
IUCV TEST COMPLETION 382, 564
MAPMDISK macro 827
PFAULT macro 856
REFPAGE macro 864
VMUDQ macro 880
specifying a PIP variables 422
specifying log data 493, 512
SPLINK 976
SPO (storage-protection override) 10
spool file external attribute buffer manipulation, DIAGNOSE code X'B8' 153
spool file manipulation for input, DIAGNOSE code X'14' 25
spool file opening and characteristic querying with DIAGNOSE code X'BC' 155
spool file origin information, DIAGNOSE code X'F8' 187
spool system service (*SPL)
AFP printing interface 752
CLOSE function 757
DISABLE function 763
ENABLE function 763
generic interface 765
MESSAGE function 758
NOTIFY function 765
PURGE function 765
READ-SFBLOK function 759
READ-SPLINK function 762
READ-XAB function 761
SELECT function 754
SEND function 764
SR parameter of DIAGNOSE code X'94' 115
SRDW parameter
SRDW option of DIAGNOSE code X'94' 116
SRDW parameter of DIAGNOSE code X'94' 116
SSCH (START SUBCHANNEL) instruction 903, 908
SSI Diagnose, DIAGNOSE X'2CC' 288
SSK (SET STORAGE KEY) instruction 894, 902
SSM (SET SYSTEM MASK) instruction 894
standard DASD I/O, DIAGNOSE code X'18' 36
START I/O (SIO) instruction 894, 901, 908
START I/O FAST RELEASE (SIOF) instruction 894, 901, 908
START SUBCHANNEL (SSCH) instruction 903, 908
starting a conversation with APPC/VM functions 392
starting an APPC conversation 573
state changes
ALLOCATE (APPC) 578
APPCVM CONNECT 429
APPCVM QRYSTATE 450
APPCVM RECEIVE 461

state changes *(continued)*
APPCVM SENDCNF 469
APPCVM SENDCNFD 474
APPCVM SENDDATA 486
APPCVM SENDERR 499
APPCVM SENDREQ 503
APPCVM SETMODFY 508
APPCVM SEVER 518
CONFIRM (APPC) 579
CONFIRMED (APPC) 579
DEALLOCATE (APPC) 580
IUCV ACCEPT 528
IUCV DCLBFR 537
IUCV RTRVBFR 548
RECEIVE_AND_WAIT (APPC) 584
REQUEST_TO_SEND (APPC) 584
SEND_DATA (APPC) 585
SEND_ERROR (APPC) 587
state table for APPC/VM error conditions 407
state table for APPC/VM functions 403
state table for coordinated resource recovery, APPC/VM 405
state table, APPC/VM 403
states, address space 798, 806
states, APPC/VM
backout_received 406
backout_required 406
Confirm 404
Connect 404
defer_receive 405
defer_sever 405
prepare_received 405
Receive 393, 404
Reset 404
Send 393, 404
Sever 405
table for error conditions 407
unsolicited_request_commit_ received 406
STATUS function, logical device support facility
DIAGNOSE code X'7C' subcode X'00000006' 90
STHYI instruction
common header section 925
CP and IFL capacity information 914
designated guest information 943
designated resource pool information 952
guest list 941
hypervisor environment information 926
resource pool list 950
resource pool member list 955
special conditions, exceptions, and usage notes 957
storage protection mechanisms
key-controlled protection
fetch-protection override (FPO) 9
host access-list-controlled protection 10
host page protection 10
low-address protection (LAP) 10
storage-protection override (SPO) 10
storage-protection override (SPO) 10
storage, examine host 16
store extended-identification code, DIAGNOSE code X'00' 13
STORE SUBCHANNEL (STSCH) instruction 902, 908
STORE THEN OR SYSTEM MASK (STOSM) instruction 894
STOSM (STORE THEN OR SYSTEM MASK) instruction 894
STSCH (STORE SUBCHANNEL) instruction 902, 908
subcode X'0000' of DIAGNOSE code X'14' 26
subcode X'0004' of DIAGNOSE code X'14' 27
subcode X'0008' of DIAGNOSE code X'14' 28
subcode X'000C' of DIAGNOSE code X'14' 28
subcode X'0010' of DIAGNOSE code X'14' 28
subcode X'0014' of DIAGNOSE code X'14' 29
subcode X'0018' of DIAGNOSE code X'14' 29
subcode X'001C' of DIAGNOSE code X'14' 29
subcode X'0020' of DIAGNOSE code X'14' 30
subcode X'0024' of DIAGNOSE code X'14' 30
subcode X'0028' of DIAGNOSE code X'14' 30
subcode X'002C' of DIAGNOSE code X'14' 30
subcode X'0FFE' of DIAGNOSE code X'14' 31
subcode X'0FFF' of DIAGNOSE code X'14' 34
summaries
state table 403, 405
SVC 76, second level with DIAGNOSE code X'48' 46
symptom records
and VMDUMP, DIAGNOSE code X'94' 113
applications using DIAGNOSE code X'94' SR option 962
description 961
DIAGNOSE code X'94' 961
format 961
processing return codes, DIAGNOSE code X'94' 122
reporting 961
sections 961
SR option of DIAGNOSE code X'94' 115
usage notes from DIAGNOSE code X'94' 119
symptom system service (*SYMPTOM)
disconnecting 774
IUCV ACCEPT 773
IUCV SEVER 773
SYNC_LEVEL parameter (APPC)
of ALLOCATE 577
synchronous block I/O parameter list (HCPSBIOP) 136
synchronous general I/O parameter list (HCPSGIOP) 144
synchronous I/O
370, DIAGNOSE code X'20' 38
operations, DIAGNOSE code X'270' 263
operations, DIAGNOSE code X'A4' 135
operations, DIAGNOSE code X'A8' 144
SYNCPT synchronization level 414
syntax diagrams, how to read xxxv
SYSEVENT Query Virtual Server (QVS), DIAGNOSE code X'2E0' 290
system dump spool file, DIAGNOSE code X'34' 43
system information, DIAGNOSE code X'26C' 226
System Network Architecture (SNA)
relationship to APPC and APPC/VM 387
system resource
managing 393, 731
virtual machines connecting to 394
system service, CP
access verification (*RPI) 591
account (*ACCOUNT) 699
asynchronous CP command response (*ASYNCMD) 719
DASD block I/O (*BLOCKIO) 721
error logging (*LOGREC) 729
identify (*IDENT) 731
IUCV communication 317
list of user IDs 317
message (*MSG) 739
message all (*MSGALL) 741
sample programs using DASD block I/O (*BLOCKIO) 981

system service, CP *(continued)*
signal (*SIGNAL) 747
spool (*SPL)
AFP printing interface 752
generic interface 765
symptom (*SYMPTOM) 773
system trace file interface, DIAGNOSE code X'E0' 171, 1015

T

TDISK (temporary disk)
accounting record 703
TERMINAL LINESIZE command 59
TERMINATE ALL, logical device support facility function 1013
TERMINATE function, logical device support facility
DIAGNOSE code X'7C' subcode X'00000004' 91
terminology of IUCV macro parameters 306
TEST COMPLETION function of IUCV
condition codes 380
format 379
parameter descriptions 379
parameter list format 380
program exceptions 382
return codes 382
used with APPC/VM
condition codes 563
format 562
parameter descriptions 562
parameter list format 562
program exceptions 564
return codes 564
state changes 564
using 379
TEST I/O (TIO) instruction 894, 901, 908
TEST MESSAGE function of IUCV
condition codes 383
format 383
program exceptions 383
used in APPC/VM
completion 566
condition codes 566
format 566
program exception 566
state changes 566
using 383
TEST PENDING INTERRUPTION (TPI) instruction 902, 908
TEST SUBCHANNEL (TSCH) instruction 902, 908
time slice end, voluntary, DIAGNOSE code X'44' 46
Time-Based Unique Identifiers, DIAGNOSE code X'238' 201
time-of-day clock accounting interface, DIAGNOSE code X'70' 80
timer, pseudo 22
TIO (TEST I/O) instruction 894, 901, 908
TOD clock accounting interface, DIAGNOSE code X'70' 80
TOKEN function of PFAULT macro 859
TPI (TEST PENDING INTERRUPTION) instruction 902, 908
TPN parameter (APPC)
of ALLOCATE 577
TSAF (Transparent Services Access Facility)
used to revoke a resource 735
TSCH (TEST SUBCHANNEL) instruction 902, 908
two-way data transfer, IUCV 298
TYPE parameter (APPC)
TYPE parameter (APPC) *(continued)*
of ALLOCATE 577
of DEALLOCATE 580
of SEND_ERROR 586
types and features for devices, DIAGNOSE code X'24' 39
types of virtual machines
DIAGNOSE code restrictions 5

U

UIDs
X'280' 1049
X'29C' 1056
UNAUTHORIZE function of VMCF 996, 1001
unmapped page of a data space 791
Unsolicited_Request_Commit_Received conversation state 406
unsolicited_request_commit_received state 406
update directory
for IUCV 301
in-place with DIAGNOSE code X'84' 91
user
accounting record 701, 717
user data field for IUCV CONNECT 530
user doubleword of VMCF 1006
user IDs of CP system services 317
user-initiated accounting records 718
user-supplied areas for FINDSKEL or FINDSEG operation, format of 70
user-supplied areas for FINDSKEL, FINDSEGA, or FINDNSSA operation (64-bit), format of 74
user-supplied areas for FINDSPACE operation (64-bit), format of 73
user-supplied areas for FINDSPACE operation, format of 68
user-supplied areas for SEGEXT function, format of 66
user-supplied output area — 64-bit member list 74
user-supplied output area — member list 71
uses for VM data spaces 784
using a remote macro work area 796
using basic APPC/VM functions 392
using data spaces in your applications 786
using DEFWORKA to force unique macro work areas 795
using DEFWORKA within a nonreentrant program 794
using DEFWORKA within a reentrant program 794

V

validate user authorization, DIAGNOSE code X'88' 105
vestigial status 901, 903, 908
virtual console function, DIAGNOSE code X'08' 19
virtual device class 192
virtual device flag 41, 192
virtual device flags, CP370 972
virtual device status 41, 192
virtual device status, CP370 971
virtual device type 41, 192
virtual device type class 41
virtual disks in storage
accounting record 712
virtual machine
resource usage accounting record 701, 717
virtual machine communication to the spool system service 754

virtual machine group 747
virtual machine information, DIAGNOSE code X'260' 222
virtual machine modes
  DIAGNOSE code restrictions 5
virtual machine storage size, DIAGNOSE code X'60' 62
virtual machines, types of
  DIAGNOSE code restrictions 5
virtual multiprocessor considerations 308
VM architected area 434, 441
VM communication server area
  communication server area, VM 445
VM data spaces
  accessing storage 787
  adding an ALE to an access list 787
  address space identification token (ASIT) 787
  coding macros 793
  CP macro use considerations 786
  created data spaces being shared with other users 783
  creating 786
  creating a remote work area example 796
  creating with ADRSPACE CREATE 786
  definition 783
  DIAGNOSE code use considerations 786
  forcing unique work areas example 795
  instance 787
  nonreentrant program example 794
  overview 783
  reentrant program example 794
  summary of operations 785
  use in applications 786
  uses 784
VMBAT mode name 419
VMCF (Virtual Machine Communication Facility)
  control functions 995
  data transfer functions 997
  DIAGNOSE code X'68'
    data transfer error codes 1009
    return codes 1007
  external interrupt, X'4001' 1004
  functions
    AUTHORIZE 996, 1001
    CANCEL 996, 1001
    IDENTIFY 997, 1001
    QUIESCE 996, 1001
    RECEIVE 999, 1001
    REJECT 997, 1001
    REPLY 999, 1001
    RESUME 997, 1001
    SEND 998, 1001
    SEND/RECV 998, 1001
    SENDX 999, 1001
    SETLIMIT 1000, 1001
    table of functions 989
    UNAUTHORIZE 996, 1001
  introduction 989
  invoking functions 1000
  protocol
    IDENTIFY 995
    SEND 993
    SEND/RECV 993
    SENDX 994
  receiving messages from the special message facility 1011
  required VMCPARM fields for VMCF functions 1003
VMCF (Virtual Machine Communication Facility) *(continued)*
  return codes 1007
  special message facility 989
  table of functions 989
  user doubleword 1006
  using
    applications 990
    general considerations 992
    performance considerations 992
    security 991
VMCMFUNC subcodes 1005
VMCMHDR, VMCF external interrupt message header
  VMCMFUNC subcodes 1005
VMCPARM, VMCF parameter list
  required fields for VMCF functions 1003
VMDUMP and symptom record service, DIAGNOSE code X'94' 113
VMINT mode name 419
VMUDQ macro
  LSTMDISK function
    parameter list 876
volume serial support, DIAGNOSE code X'D0' 161
voluntary time slice end, DIAGNOSE code X'44' 46
voluntary time slice, DIAGNOSE code X'9C' 133
VRDCBLOK DSECT fields 190

W

what are data spaces? 783
WHAT_RECEIVED parameter (APPC)
  of RECEIVE_AND_WAIT 583
work areas, macro 823
writing the virtual printer XAB 151

X

X'4001' external interrupt in VMCF 1004
XAB (external attribute buffer)
  format 979
XAUTOLOG command
  specifying invalid password 700
XCONFIG ACCESSLIST directory control statement 786
XCONFIG ADDRSPACE directory control statement 786

Z

z/VM
  directory
    authorization for IUCV 301
    entries in IUCV 301
    VMUDQ macro 875

Product Number: 5741-A09

Printed in USA - Product Number: 5741-A09

SC24-6272-04