z/OS

IBM Health Checker for z/OS V2R1 User's Guide

Version 2 Release 1

SC23-6843-02

Note
Before using this information and the product it supports, read the information in "Notices" on page 635.

This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures ix

Tables xi

About This Information xiii
Who should use this document xiii
Where to find more information xiii

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

Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated September 2014 xvii
Summary of changes for z/OS Version 2 Release 1 as updated March 2014 xvii
z/OS Version 2 Release 1 summary of changes . . xix
Exploitation of the Flash Express feature . . . xix

Part 1. Using IBM Health Checker for z/OS 1

Chapter 1. Introduction 3
What is a check? 4
For more information, see our fabulous Redpaper!. . 5
Background for IBM's checks 5

Chapter 2. Setting up IBM Health Checker for z/OS 7
Stopping and Starting IBM Health Checker for z/OS Manually 7
Sharing critical IBM Health Checker for z/OS information between systems at different levels. . . 8
Sharing IEASYSxx 8
Sharing the HZSPROC procedure 8
Sharing HZSPRMxx 8
Steps for optimizing IBM Health Checker for z/OS . 9
In a rush? Use these basic steps 9
Software requirements. 10
Software requirement for running REXX checks 10
Customizing the IBM Health Checker for z/OS procedure 10
Allocate the HZSPDATA data set to save check data between restarts 10
Monitoring and sizing the HZSPDATA data set 11
Optionally set up the HZSPRINT utility 12
Optionally define log streams to keep a record of the check output. 12
Create security definitions 14
Setting up security for the IBM Health Checker for z/OS started task 14
Setting up security for the HZSPRINT utility . . 16
Setting up security for IBM Health Checker for SDSF support. 19
Create multilevel security definitions 19
Create HZSPRMxx parmlib members. 20
Tell the system which HZSPRMxx members you want to use 21
Assign IBM Health Checker for z/OS to a WLM service class 23
Obtain checks for IBM Health Checker for z/OS . . 23

Chapter 3. Working with check output 25
Hey! My system has been configured like this for years, and now I'm receiving exceptions! 27
Understanding system data issued with the check messages 27
Understanding exception messages 28
Evaluating check output and resolving exceptions 30
Customizing check exceptions with dynamically varying severity 31
Approaches to automation with IBM Health Checker for z/OS 32
More automation ideas 32
Using HZS exception messages for automation 34
Understanding check state and status. 34
User controlled states 35
IBM Health Checker for z/OS controlled states 36
ACTIVE(DISABLED) and INACTIVE(ENABLED) - understanding check state combinations . . . 36
Check status 37
Using the HZSPRINT utility. 37
Example of HZSPRINT output 39
HZSPRINT utility completion codes 39
Finding check message documentation with LookAt 41

Chapter 4. Managing checks 43
Making dynamic, temporary changes to checks . . 44
Using SDSF to manage checks 45
Managing checks with the MODIFY *hzsproc* command 47
Making persistent changes to checks 52
Creating and maintaining IBM Health Checker for z/OS policies. 53
How IBM Health Checker for z/OS builds policies from policy statements 54
Can I put non-policy statements in my HZSPRMxx member? 59
Using SYNCVAL in a policy to specify the time of day that a check runs 59
Policy statement examples 63
Can I create policy statements using the MODIFY command?. 63
Specifying the HZSPRMxx members you want the system to use 64
Using HZSPRMxx and MODIFY *hzsproc* command 65
Guidelines for HZSPRMxx parmlib members . . 66

Syntax and parameters for HZSPRMxx and MODIFY *hzsproc* 68
Examples of DISPLAY output 89

Part 2. Developing Checks for IBM Health Checker for z/OS 93

Chapter 5. Planning checks 95
Identifying potential checks 96
The life-cycle of a check - check terminology . . . 96
What kind of check do you want to write? 97
Local checks 97
Remote checks 98
Writing local and remote checks in Metal C . . 99
REXX checks 100
Summary of checks - differences and similarities 100
Issuing messages for your check - message table checks versus DIRECTMSG checks 102
Where to next? A road map for developing your check 103

Chapter 6. Writing local check routines 105
Metal C or assembler? 105
Sample local checks 106
Local check routine basics 106
Defining a local check to IBM Health Checker for z/OS 108
Programming considerations 109
Environment 109
Requirements 109
Restrictions 109
Gotchas 109
Input Registers 110
Output Registers 110
Establishing a recovery routine for a check . . 110
Assembler reentrant entry and exit linkage . . 110
Using the check parameter parsing service (HZSCPARS) 111
Using the HZSPQE data area in your local check routine 111
Function codes for local check routines 113
Creating and using data saved between restarts 115
Using ENF event code 67 to listen for check status changes 116
Issuing messages in your local check routine with the HZSFMSG macro 117
Reporting check exceptions 118
Defining the variables for your messages 120
Using default HZSMGB data area format (MGBFORMAT=0). 121
Using HZSMGB data area format MGBFORMAT=1 123
Writing a check with dynamic severity levels . . . 124
Controlling check exception message WTOs and their automation consequences 126
The well-behaved local check routine - recommendations and recovery considerations . . 127
Building Metal C checks. 130
Debugging checks 132

Chapter 7. Writing remote check routines 135
Metal C or assembler? 135
Sample checks 136
Remote check routine basics 137
Programming considerations 138
Environment 138
Requirements 138
Restrictions 138
Establishing a recovery routine for a check . . 139
Preparing for check definition - making sure IBM Health Checker for z/OS is up and running . . . 139
Using ENF event code 67 to listen for IBM Health Checker for z/OS availability 139
Allocate a pause element token using IEAVAPE 140
Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS. 140
Example of an HZSADDCK macro call for a remote check 142
Pause the remote check routine with IEAVPSE . . 143
Using HZSCHECK REQUEST=OPSTART and REQUEST=OPCOMPLETE to communicate check start and stop to IBM Health Checker for z/OS . . 143
Using the check parameter parsing service (HZSCPARS) 143
Using the HZSPQE data area in your remote check routine 144
Release codes for remote check routines 145
Ending a check that is coupled with an application 147
Creating and using data saved between restarts 148
Issuing messages in your remote check routine with the HZSFMSG macro 150
Reporting check exceptions. 151
Defining the variables for your messages 153
Using default HZSMGB data area format (MGBFORMAT=0). 154
Using HZSMGB data area format MGBFORMAT=1 157
Writing a check with dynamic severity levels. . . 158
Controlling check exception message WTOs and their automation consequences 160
Recommendations and recovery considerations for remote checks 161
Building Metal C checks. 163
Debugging checks 166

Chapter 8. Writing REXX checks . . . 169
Sample REXX checks 169
REXX check basics 170
Using input data sets in a TSO-environment REXX check 174
Using REXXIN data sets. 174
REXXIN data set naming conventions 174
Using REXXOUT data sets 175
REXXOUT data set naming conventions . . . 175
Examples: Capturing error data in REXXOUT 175
Defining a REXX check to IBM Health Checker for z/OS 177

Using ENF event code 67 to listen for check status changes 179
Issuing messages from your REXX check with the HZSLFMSG function 180
Reporting check exceptions 182
Writing a check with dynamic severity levels . . . 184
Controlling check exception message WTOs and their automation consequences 186
The well-behaved REXX check - recommendations and recovery considerations 187
Debugging REXX checks 188

Chapter 9. Writing an HZSADDCHECK exit routine 191
Programming considerations for the HZSADDCHECK exit routine 194
Environment 194
Input Registers 194
Output Registers 195
Defining multiple local or REXX checks in a single HZSADDCHECK exit routine 195
Dynamically adding local or REXX exec checks to IBM Health Checker for z/OS 196
Using operator commands to add checks to the system dynamically 196
Using a routine to add checks to the system dynamically 197
Debugging HZSADDCHECK exit routine abends 197
Creating product code that automatically registers checks at initialization 197
Creating product code that deletes checks as it goes down 198

Chapter 10. Creating the message input for your check 199
How messages and message variables are issued at check runtime 200
Planning your check messages 201
Planning your exception messages 202
Planning your information messages 202
Planning your report messages 203
Planning your debug messages 203
Decide what release your check will run on . . 203
Decide whether to translate your check exception messages into other national languages 204
Rely on IBM Health Checker for z/OS to issue basic check information for you 204
Creating the message table 205
Examples of message input 205
Syntax of message input 214
Message input tags 214
Special formatting tags for the message table 223
How messages are formatted in the message buffer 225
Using symbols in the message table 227
Generating the compilable assembler CSECT for the message table 231
Support for translating messages to other languages 232
Guidelines for coding translatable exception message text lines 232

Chapter 11. IBM Health Checker for z/OS System REXX Functions 235
HZSLSTRT function 236
Input variables 236
Output variables 237
HZSLSTRT return codes 239
HZSLFMSG function 240
Input variables 240
HZSLFMSG Output variables 251
HZSLFMSG return codes 253
HZSLSTOP function 255
Input variables 255
Output variables 256
HZSLSTOP return codes 256

Chapter 12. IBM Health Checker for z/OS HZS macros 259
HZSADDCK macro — HZS add a check 260
Description 260
HZSCHECK macro — HZS Check command request 279
Description 279
HZSCPARS macro — HZS Check Parameter Parsing 294
Description 294
HZSFMSG macro — Issue a formatted check message 307
Description 307
Example 1: 337
Example 2: 338
HZSPREAD macro — Read Check Persistent Data 339
Description 339
HZSPWRIT macro — Write Check Persistent Data 349
Description 349
HZSQUERY macro — HZS Query 357
Description 357

Part 3. Check Descriptions 379

Chapter 13. IBM Health Checker for z/OS checks 381
Where are the migration checks? 381
Allocation checks (IBMALLOC) 381
ALLOC_ALLC_OFFLN_POLICY 381
ALLOC_SPEC_WAIT_POLICY 383
ALLOC_TIOT_SIZE 384
ASM checks (IBMASM) 386
ASM_NUMBER_LOCAL_DATASETS 386
ASM_PAGE_ADD 387
ASM_PLPA_COMMON_SIZE 388
ASM_PLPA_COMMON_USAGE 389
ASM_LOCAL_SLOT_USAGE 390
Catalog checks (IBMCATALOG) 392
CATALOG_IMBED_REPLICATE 392
CATALOG_RNLS 393
Communications Server checks (IBMCS) 394

CSRES_AUTOQ_GLOBALTCPIPDATA 394
CSRES_AUTOQ_RESOLVEVIA 395
CSRES_AUTOQ_TIMEOUT 396
CSTCP_CINET_PORTRNG_RSV_*tcpipstackname* 397
CSTCP_IPMAXRT4_*tcpipstackname* 398
CSTCP_IPMAXRT6_*tcpipstackname* 399
CSTCP_SYSTCPIP_CTRACE_*tcpipstackname* . . 400
CSTCP_SYSPLEXMON_RECOV_*tcpipstackname* 401
CSTCP_TCPMAXRCVBUFRSIZE_*tcpipstackname* 402
CSVTAM_CSM_STG_LIMIT 403
CSVTAM_T1BUF_T2BUF_EE 404
CSVTAM_T1BUF_T2BUF_NOEE 405
CSVTAM_VIT_DSPSIZE 405
CSVTAM_VIT_OPT_ALL 406
CSVTAM_VIT_OPT_PSSSMS 407
CSVTAM_VIT_SIZE 408
ZOSMIGV2R1_CS_GATEWAY 409
ZOSMIGV2R1_CS_LEGACYDEVICE 410
Consoles checks (IBMCNZ) 411
CNZ_AMRF_Eventual_Action_Msgs. 411
CNZ_Console_MasterAuth_Cmdsys 412
CNZ_Console_Mscope_And_Routcode 413
CNZ_Console_Operating_Mode 413
CNZ_Console_Routcode_11 414
CNZ_EMCS_Hardcopy_Mscope 415
CNZ_EMCS_Inactive_Consoles 415
CNZ_OBSOLETE_MSGFLD_AUTOMATION 416
CNZ_Syscons_Allowcmd 418
CNZ_Syscons_Mscope 418
CNZ_Syscons_PD_Mode 419
CNZ_Syscons_Routcode. 420
CNZ_Task_Table 420
ZOSMIGV1R13_CNZ_Cons_Oper_Mode . . . 421
Contents supervision checks (IBMCSV) 422
CSV_APF_EXISTS 422
CSV_LNKLST_NEWEXTENTS 423
CSV_LNKLST_SPACE 425
CSV_LPA_CHANGES 426
DAE checks (IBMDAE) 428
DAE_SHAREDSN 428
DAE_SUPPRESSING 429
Device Manager checks (IBMDMO) 430
DMO_TAPE_LIBRARY_INIT_ERRORS 430
DFSMS OPEN/CLOSE/EOV checks (IBMOCE) . . 431
OCE_XTIOT_CHECK. 431
Global Resource Serialization checks (IBMGRS) . . 432
GRS_AUTHQLVL_SETTING 432
GRS_Mode 433
GRS_SYNCHRES 434
GRS_CONVERT_RESERVES 434
GRS_EXIT_PERFORMANCE 435
GRS_GRSQ_SETTING 436
GRS_RNL_IGNORED_CONV 436
HSM checks (IBMHSM) 437
HSM_CDSB_BACKUP_COPIES 437
HSM_CDSB_DASD_BACKUPS 438
HSM_CDSB_VALID_BACKUPS 439
ICSF checks (IBMICSF) 441
ICSF_COPROCESSOR_STATE_NEGCHANGE 441
ICSF_MASTER_KEY_CONSISTENCY 441
ICSFMIG_DEPRECATED_SERV_WARNINGS 442
ICSFMIG7731_ICSF_RETAINED_RSAKEY . . . 444
ICSFMIG7731_ICSF_PKDS_TO_4096BIT . . . 445
ICSFMIG77A1_COPROCESSOR_ACTIVE . . . 446
ICSFMIG77A1_TKDS_OBJECT. 447
ICSFMIG77A1_UNSUPPORTED_HW 448
Infoprint Server checks (IBMINFOPRINT) 449
INFOPRINT_PRINTWAY_MODE. 449
INFOPRINT_V2DB_CHECK 450
ZOSMIGV1R12_INFOPRINT_INVSIZE 452
IOS checks (IBMIOS) 454
IOS_CAPTUCB_PROTECT 454
IOS_CMRTIME_MONITOR. 455
IOS_FABRIC_MONITOR 457
IOS_IORATE_MONITOR 459
IOS_MIDAW 461
IOS_STORAGE_IOSBLKS 462
JES2 checks (IBMJES2) 463
JES2_Z11_Upgrade_CK_JES2 463
Loadwait/Restart checks (IBMSVA) 464
SVA_AUTOIPL_DEFINED 464
SVA_AUTOIPL_DEV_VALIDATION. 465
PDSE checks (IBMPDSE) 466
PDSE_SMSPDSE1 466
Predictive failure analysis checks (IBMPFA) . . . 467
RACF checks (IBMRACF) 467
Write your own RACF resource checks! . . . 467
RACF_AIM_STAGE 471
RACF_CERTIFICATE_EXPIRATION. 473
RACF_GRS_RNL 475
RACF_ICHAUTAB_NONLPA 481
RACF_SENSITIVE_RESOURCES 482
RACF_*classname*_ACTIVE 488
RACF_IBMUSER_REVOKED 490
RACF_UNIX_ID 491
ZOSMIGV1R13_DEFAULT_UNIX_ID 496
Reconfiguration checks (IBMRCF) 500
RCF_PCCA_ABOVE_16M 500
ZOSMIGV1R12_RCF_PCCA_ABOVE_16M . . 501
RMM checks (IBMRMM) 502
ZOSMIGV1R10_RMM_REJECTS_DEFINED . . 502
ZOSMIGV1R10_RMM_VOL_REPLACE_LIM 503
ZOSMIGV1R10_RMM_VRS_DELETED 504
ZOSMIGV1R11_RMM_DUPLICATE_GDG. . . 505
ZOSMIGV1R11_RMM_REXX_STEM 506
ZOSMIGV1R11_RMM_VRSEL_OLD 507
RRS checks (IBMRRS) 508
RRS_ArchiveCFStructure 508
RRS_RMDataLogDuplexMode. 509
RRS_RMDOffloadSize 510
RRS_DUROffloadSize. 511
RRS_MUROffloadSize 511
RRS_RSTOffloadSize 512
RRS_Storage_NumLargeLOGBlks. 513
RRS_Storage_NumLargeMSGBlks 514
RRS_Storage_NumServerReqs 515
RRS_Storage_NumTransBlks 516
RSM checks (IBMRSM) 517
RSM_HVSHARE 517
RSM_MEMLIMIT 518
RSM_MAXCADS 519
RSM_AFQ 520

RSM_REAL 521
RSM_RSU 522
RTM checks (IBMRTM) 523
RTM_IEAVTRML 523
SDSF checks (IBMSDSF) 524
SDSF_CLASS_SDSF_ACTIVE 524
SDSF_ISFPARMS_IN_USE 524
SDUMP checks (IBMSDUMP) 526
SDUMP_AVAILABLE. 526
SDUMP_AUTO_ALLOCATION 526
Serviceability checks (IBMSLIP) 527
| SLIP_PER 527
SMB checks (IBMSMB) 528
SMB_NO_ZFS_SYSPLEX_AWARE 528
ZOSMIGREC_SMB_RPC. 529
SMS checks (IBMSMS) 530
SMS_CDS_REUSE_OPTION 530
SMS_CDS_SEPARATE_VOLUMES 530
Supervisor checks (IBMSUP) 531
IEA_ASIDS 531
IEA_LXS 532
SUP_HIPERDISPATCH 533
SUP_HiperDispatchCPUConfig 536
SUP_LCCA_ABOVE_16M 537
| SUP_SYSTEM_SYMBOL_TABLE_SIZE 538
ZOSMIGV1R12_SUP_LCCA_ABOVE_16M. . . 539
System logger checks (IBMIXGLOGR) 541
IXGLOGR_STAGINGDSFULL 541
IXGLOGR_ENTRYTHRESHOLD 543
IXGLOGR_STRUCTUREFULL. 546
| System trace checks (IBMSYSTRACE) 549
| SYSTRACE_BRANCH 549
| SYSTRACE_MODE 550
Timer supervisor checks (IBMTIMER) 551
ZOSMIGREC_SUP_TIMER_INUSE 551
TSO/E (IBMTSOE) 552
TSOE_USERLOGS. 552
TSOE_PARMLIB_ERROR 553
z/OS UNIX System Services checks (IBMUSS) . . 554
USS_AUTOMOUNT_DELAY 554
USS_CLIENT_MOUNTS. 554
USS_FILESYS_CONFIG 556
USS_HFS_DETECTED 558
USS_KERNEL_PVTSTG_THRESHOLD 559
USS_KERNEL_STACKS_THRESHOLD 560
USS_MAXSOCKETS_MAXFILEPROC 561
USS_PARMLIB 562
USS_PARMLIB_MOUNTS 565
ZOSMIGREC_ROOT_FS_SIZE. 567
ZOSMIGV1R13_RO_SYMLINKS 569
| VLF checks (IBMVLF) 569
| VLF_MAXVIRT. 569
VSAM checks (IBMVSAM) 571
VSAM_INDEX_TRAP 571
VSAM RLS checks (IBMVSAMRLS) 572
VSAMRLS_CFCACHE_MINIMUM_SIZE . . . 572
VSAMRLS_CFLS_FALSE_CONTENTION . . . 573
VSAMRLS_DIAG_CONTENTION 574
VSAMRLS_QUIESCE_STATUS 575
VSAMRLS_SHCDS_CONSISTENCY. 576
VSAMRLS_SHCDS_MINIMUM_SIZE 577
VSAMRLS_SINGLE_POINT_FAILURE 577
VSAMRLS_TVS_ENABLED 578
VSM checks (IBMVSM) 579
| VSM_ALLOWUSERKEYCSA 579
| VSM_CSA_LARGEST_FREE 580
VSM_CSA_LIMIT 582
VSM_SQA_LIMIT 583
VSM_PVT_LIMIT 584
VSM_CSA_THRESHOLD 584
VSM_SQA_THRESHOLD 587
VSM_CSA_CHANGE. 589
XCF checks (IBMXCF) 590
XCF_CDS_MAXSYSTEM 590
XCF_CDS_SEPARATION 590
XCF_CDS_SPOF 591
XCF_CF_ALLOCATION_PERMITTED 592
XCF_CF_CONNECTIVITY 593
XCF_CF_MEMORY_UTILIZATION 594
XCF_CF_PROCESSORS 595
| XCF_CF_SCM_UTILIZATION 596
XCF_CF_STR_AVAILABILITY 597
XCF_CF_STR_DUPLEX 598
XCF_CF_STR_EXCLLIST 599
| XCF_CF_STR_MAXSCM. 600
| XCF_CF_STR_MAXSPACE 600
XCF_CF_STR_NONVOLATILE 601
XCF_CF_STR_POLICYSIZE. 602
XCF_CF_STR_PREFLIST. 603
| XCF_CF_STR_SCM_AUGMENTED 604
| XCF_CF_STR_SCMMAXSIZE 605
| XCF_CF_STR_SCM_MINCOUNTS 605
| XCF_CR_STR_SCM_UTILIZATION 606
XCF_CF_SYSPLEX_CONNECTIVITY 608
XCF_CFRM_MSGBASED 609
XCF_CLEANUP_VALUE 610
XCF_DEFAULT_MAXMSG 610
XCF_FDI 611
XCF_MAXMSG_NUMBUF_RATIO 612
XCF_SFM_ACTIVE 612
XCF_SFM_CFSTRHANGTIME. 613
XCF_SFM_CONNFAIL 614
XCF_SFM_SSUMLIMIT 615
XCF_SFM_SUM_ACTION 616
XCF_SIG_CONNECTIVITY. 616
XCF_SIG_PATH_SEPARATION 617
XCF_SIG_STR_SIZE 618
XCF_SYSPLEX_CDS_CAPACITY 619
XCF_SYSSTATDET_PARTITIONING. 620
XCF_TCLASS_CLASSLEN 621
XCF_TCLASS_CONNECTIVITY 621
XCF_TCLASS_HAS_UNDESIG 622
z/OS File System checks (IBMZFS) 623
ZOSMIGV1R11_ZFS_INTERFACELEVEL . . . 623
ZOSMIGREC_ZFS_RM_MULTIFS 623
ZOSMIGV1R11_ZFS_RM_MULTIFS 624
ZOSMIGV1R13_ZFS_FILESYS 625
ZOSMIGV2R1_ZFS_VERIFY_CACHESIZE. . . 626
ZFS_VERIFY_CACHESIZE 627

Part 4. Appendixes 629

Appendix. Accessibility 631
Accessibility features 631
Using assistive technologies 631
Keyboard navigation of the user interface 631
Dotted decimal syntax diagrams 631

Notices 635
Policy for unsupported hardware. 636
Minimum supported hardware 637

Trademarks 639

Index 641

Figures

1. IBM Health Checker for z/OS with a local check 4
2. Using LookAt to find check message documentation 42
3. Creating a policy in multiple HZSPRMxx members 55
4. Creating multiple policies in one HZSPRMxx member 56
5. The parts of a local check 98
6. The parts of a remote check 99
7. The parts of a REXX check 100
8. Example of issuing a message with variables in an assembler check. 122
9. Example of issuing a message with variables using MGBFORMAT=1 124
10. Example of issuing a message with variables in an assembler check. 156
11. Example of issuing a message with variables using MGBFORMAT=1 158
12. Assembler example: Defining the HZSADDCHECK exit routine r and adding checks 197
13. Inputs and outputs for creating a complete message table 200
14. Message and variable resolution at runtime 201
15. Example of a setup data set that defines symbols used in the message table 230

Tables

1. Access required for printing check output from the message buffer using HZSPRINT 18
2. Interaction of HZSPRMxx settings specified in HZSPROC and IEASYSxx 22
3. User controlled states 35
4. States controlled by IBM Health Checker for z/OS. 36
5. Check state combinations 37
6. When do I use which interface to manage checks? 44
7. F hzsproc command examples 47
8. Summary of local, remote, and REXX checks 100
9. Correlation between IBM Health Checker for z/OS mapping macros and Metal C header files 106
10. Important fields in the HZSPQE data area for a local check routine 112
11. Summary of function codes for local checks 113
12. Important MGBFORMAT=0 fields in the HZSMGB data area for check message variables 123
13. Important MGBFORMAT=1 fields in the HZSMGB data area for check message variables 124
14. Correlation between IBM Health Checker for z/OS mapping macros and Metal C header files 136
15. Important fields in the HZSPQE data area for a remote check routine 144
16. Summary of release codes for remote checks 146
17. Important MGBFORMAT=0 fields in the HZSMGB data area for check message variables 156
18. Important MGBFORMAT=1 fields in the HZSMGB data area for check message variables 158
19. Important HZSPQE information used in a REXX check from HZSLSTRT variables . . . 172
20. A summary of message types for IBM Health Checker for z/OS 216
21. Variable input and output lengths and alignment: 220
22. Which variables allow maxlen? 221
23. Description of <msgitem> classes required for all message explanations 222
24. How messages are formatted in the message buffer 226
25. A summary of pre-defined symbols that resolve when the check runs 227
26. A summary of pre-defined symbols that resolve when you generate the CSECT for the message table 228
27. HZSLSTRT input variable 236
28. HZSLSTRT output variables 237
29. HZSLFMSG input variables 240
30. HZSLFMSG_REQUEST='CHECKMSG' input variables 241
31. HZSLFMSG_REQUEST='DIRECTMSG' required input variables 243
32. HZSLFMSG_REQUEST='DIRECTMSG' optional input variables for HZSLFMSG_REASON='CHECKEXCEPTION' . 244
33. HZSLFMSG_REQUEST='HZSMSG' input variables 245
34. HZSLFMSG_REQUEST='STOP' input variables 250
35. HZSLFMSG output varaibles 251
36. HZSLSTOP input variable 255
37. HZSLSTOP output variables 256
38. Return and Reason Codes for the HZSADDCK Macro 274
39. Return and Reason Codes for the HZSCHECK Macro 289
40. Return and Reason Codes for the HZSCPARS Macro 305
41. Return and Reason Codes for the HZSFMSG Macro 336
42. Return and Reason Codes for the HZSPREAD Macro 345
43. Return and Reason Codes for the HZSPWRIT Macro 354
44. Return and Reason Codes for the HZSQUERY Macro 374
45. RACF_CERTIFICATE_EXPIRATION report columns 474
46. RACF_CERTIFICATE_EXPIRED attributes 474
47. **Systems** Level ENQs that RACF_GRS_RNL checks 476
48. **System** Level ENQs that RACF_GRS_RNL checks 477
49. Additional Discrete General Resources for RACF_SENSITIVE_RESOURCES 483
50. Additional “Generic” General Resources for RACF_SENSITIVE_RESOURCES 483
51. RACF_UNIX_ID check actions based on whether the BPX.UNIQUE.USER and BPX.DEFAULT.USER profiles are defined in the FACILITY class 492
52. ZOSMIGV1R13_DEFAULT_UNIX_ID check actions and migration actions 496
53. Synonyms of Parameters 581

About This Information

This document presents the information you need to install, use, and develop checks for IBM® Health Checker for z/OS®. IBM Health Checker for z/OS is a component of MVS™ that identifies potential problems before they impact your availability or, in worst cases, cause outages. It checks the current active z/OS and sysplex settings and definitions for a system and compares the values to those suggested by IBM or defined by you. It is not meant to be a diagnostic or monitoring tool, but rather a continuously running preventative that finds deviations from best practices. IBM Health Checker for z/OS produces output in the form of detailed messages to let you know of both potential problems and suggested actions to take.

IBM Health Checker for z/OS is available for z/OS V1R4 and later users.

Who should use this document

This document is intended for two separate audiences:

- People using IBM Health Checker for z/OS to find potential problems in their installation. Part 1, "Using IBM Health Checker for z/OS," on page 1 describes how to setup IBM Health Checker for z/OS, work with check output and manage checks. Part 3, "Check Descriptions," on page 379 also includes information that an IBM Health Checker for z/OS user will need, including check descriptions and IBM Health Checker for z/OS framework HZS messages.
- People developing their own IBM Health Checker for z/OS checks. Part 2, "Developing Checks for IBM Health Checker for z/OS," on page 93 includes information on planning checks, developing a check routine, developing messages for your check, and getting your check into the IBM Health Checker for z/OS framework. Part 3, "Check Descriptions," on page 379 also includes information about IBM Health Checker for z/OS macros for use in developing checks.

Where to find more information

Checks for IBM Health Checker for z/OS will be delivered both as an integrated part of a z/OS release or separately, as PTFs. Many new and updated checks will be distributed as PTFs, so that they are not dependent on z/OS release boundaries and can be added at any time. For the most up-to-date information on checks available, see the following Web site:

`http://www.ibm.com/servers/eserver/zseries/zos/hchecker/check_table.html`

IBM Health Checker for z/OS is also available for z/OS V1.4, V1.5, and V1.6 as a z/OS Web download. Make sure that you review the PSP bucket as described in the Web download program directory. There is required service that you must install. This code is part of z/OS V1.4 and V1.5, which have reached end of service. This code is provided without service for those releases. You can find the z/OS Downloads page at:

`z/OS Internet Library (http://www.ibm.com/systems/z/os/zos/bkserv/)`

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of

the documents for all products that are part of z/OS, see *z/OS Information Roadmap*. The following table lists titles and order numbers for documents related to other products.

How to send your comments to IBM

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

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the Contact z/OS.
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  IBM Health Checker for z/OS User's Guide
  SC23-6843-02
- The topic and page number that is related to your comment.
- The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at IBM support portal.

Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated September 2014

The following changes are made to z/OS Version 2 Release 1 (V2R1) as updated September 2014.

New

The following Health Checks are new in z/OS V2R1:

- "ZOSMIGV2R1_CS_LEGACYDEVICE" on page 410

Changes

The following Health Checks have been updated in z/OS V2R1:

- "ASM_LOCAL_SLOT_USAGE" on page 390
- "ASM_PLPA_COMMON_SIZE" on page 388
- "ASM_PLPA_COMMON_USAGE" on page 389
- "RACF_*classname*_ACTIVE" on page 488
- "RACF_SENSITIVE_RESOURCES" on page 482
- "VSM_CSA_THRESHOLD" on page 584
- "VSAM_INDEX_TRAP" on page 571
- "VSM_SQA_THRESHOLD" on page 587

Summary of changes for z/OS Version 2 Release 1 as updated March 2014

The following changes are made to z/OS Version 2 Release 1 (V2R1) as updated March 2014.

New

The following sections are new in z/OS V2R1.

- "Stopping and Starting IBM Health Checker for z/OS Manually" on page 7
- "Sharing critical IBM Health Checker for z/OS information between systems at different levels" on page 8
- "Customizing the IBM Health Checker for z/OS procedure" on page 10
- "Monitoring and sizing the HZSPDATA data set" on page 11
- "Create HZSPRMxx parmlib members" on page 20
- "Customizing check exceptions with dynamically varying severity" on page 31
- "Specifying the HZSPRMxx members you want the system to use" on page 64
- "Writing a check with dynamic severity levels" on page 124

The following Health Checks are new in z/OS V2R1.

- "CATALOG_RNLS" on page 393
- "ZOSMIGV2R1_CS_GATEWAY" on page 409
- "DFSMS OPEN/CLOSE/EOV checks (IBMOCE)" on page 431
- "IOS_FABRIC_MONITOR" on page 457

- "IOS_IORATE_MONITOR" on page 459
- "RACF_AIM_STAGE" on page 471
- "RACF_CERTIFICATE_EXPIRATION" on page 473
- "RACF_UNIX_ID" on page 491
- "SLIP_PER" on page 527
- "SUP_SYSTEM_SYMBOL_TABLE_SIZE" on page 538
- "SYSTRACE_BRANCH" on page 549
- "SYSTRACE_MODE" on page 550
- "VLF_MAXVIRT" on page 569
- "VSM_ALLOWUSERKEYCSA" on page 579
- "VSM_CSA_LARGEST_FREE" on page 580
- "XCF_CF_SCM_UTILIZATION" on page 596
- "XCF_CF_STR_MAXSCM" on page 600
- "XCF_CF_STR_MAXSPACE" on page 600
- "XCF_CF_STR_SCM_AUGMENTED" on page 604
- "XCF_CF_STR_SCMMAXSIZE" on page 605
- "XCF_CF_STR_SCM_MINCOUNTS" on page 605
- "XCF_CR_STR_SCM_UTILIZATION" on page 606

Changes

The following sections have been updated in z/OS V2R1:

- "Steps for optimizing IBM Health Checker for z/OS" on page 9
- "Allocate the HZSPDATA data set to save check data between restarts" on page 10
- "Setting up security for the IBM Health Checker for z/OS started task" on page 14
- "Understanding system data issued with the check messages" on page 27
- "Using the HZSPRINT utility" on page 37
- "Using SDSF to manage checks" on page 45
- "Making persistent changes to checks" on page 52
- "How IBM Health Checker for z/OS builds policies from policy statements" on page 54
- "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68
- "Examples of DISPLAY output" on page 89

The following Health Checks have been updated in z/OS V2R1:

- "ASM_PLPA_COMMON_SIZE" on page 388
- "ASM_PLPA_COMMON_USAGE" on page 389
- "ASM_LOCAL_SLOT_USAGE" on page 390
- "CATALOG_IMBED_REPLICATE" on page 392
- "CSV_APF_EXISTS" on page 422
- "DAE_SUPPRESSING" on page 429
- "RACF_SENSITIVE_RESOURCES" on page 482
- "USS_PARMLIB" on page 562
- "VSAMRLS_CFLS_FALSE_CONTENTION" on page 573
- "VSM_CSA_THRESHOLD" on page 584

- "VSM_SQA_THRESHOLD" on page 587
- "XCF_CF_PROCESSORS" on page 595

z/OS Version 2 Release 1 summary of changes

See important information about "Exploitation of the Flash Express feature."

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

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Exploitation of the Flash Express feature

IBM intends to provide exploitation of the Flash Express® feature on IBM zEnterprise® EC12 (zEC12) and zBC12 servers with CFLEVEL 19 for certain coupling facility list structures in the first half of 2014. This new function is designed to allow list structure data to migrate to Flash Express memory as needed, when the consumers of data do not keep pace with its creators for some reason, and migrate it back to real memory to be processed. When your installation uses WebSphere® MQ for z/OS Version 7 (5655-R36), this new capability is expected to provide significant buffering against enterprise messaging workload spikes and provide support for storing large amounts of data in shared queue structures, potentially allowing several hours' data to be stored without causing interruptions in processing. In addition, z/OS V2R1 Resource Measurement Facility™ (RMF™) is planned to provide measurement data and reporting capabilities for Flash Express when it is used with coupling facilities. Information about externals and interfaces that are related to this planned capability are being made available in z/OS V2R1 for early planning and development purposes only.

Part 1. Using IBM Health Checker for z/OS

Chapter 1. Introduction

The objective of IBM Health Checker for z/OS is to identify potential problems before they impact your availability or, in worst cases, cause outages. It checks the current active z/OS and sysplex settings and definitions for a system and compares the values to those suggested by IBM or defined by you. It is not meant to be a diagnostic or monitoring tool, but rather a continuously running preventative that finds potential problems. IBM Health Checker for z/OS produces output in the form of detailed messages to let you know of both potential problems and suggested actions to take. Note that these messages do not mean that IBM Health Checker for z/OS has found problems that you need to report to IBM! IBM Health Checker for z/OS output messages simply inform you of potential problems so that you can take action on your installation.

There are several parts to IBM Health Checker for z/OS:

- The **framework** of the IBM Health Checker for z/OS is the interface that allows you to run and manage checks. The framework is a common and open architecture, supporting check development by IBM, independent software vendors (ISVs), and users.
- Individual **checks** look for component, element, or product specific z/OS settings and definitions, checking for potential problems. The specific component or element owns, delivers, and supports the checks.

  Checks can be either **local**, and run in the IBM Health Checker for z/OS address space, or **remote**, and run in the caller's address space. So far, most IBM checks are local.

Figure 1 on page 4 shows the various parts of IBM Health Checker for z/OS:

- The IBM Health Checker for z/OS address space, where the framework, currently running local check routines, and other elements reside.
- The HZSPQE data area which contains all the information a check routine needs, including the defaults defined for the check and any installation overrides to those defaults.
- Installation overrides, which are changes the installation can make to check default values, such as interval, parameters, or other values.
- The message table, which contains message data for the check output messages that convey check results.

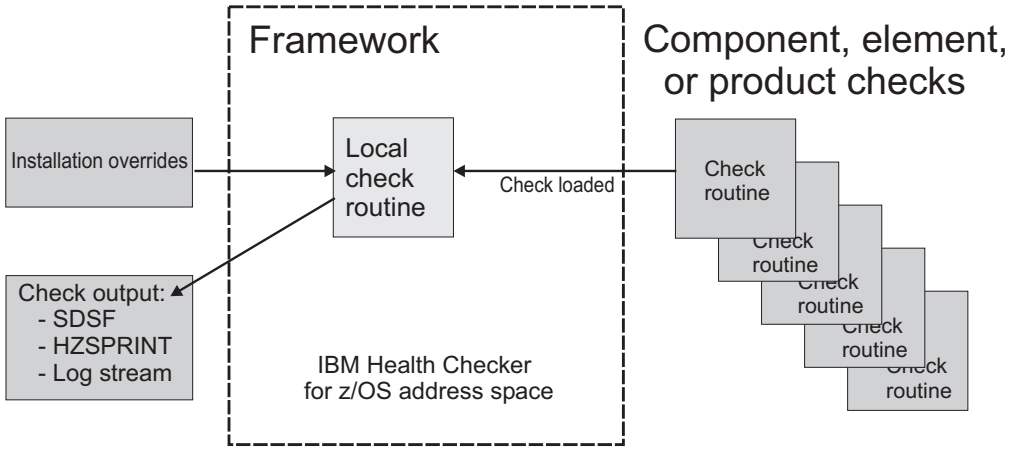


Figure 1. IBM Health Checker for z/OS with a local check

What is a check?

A check is actually a program or routine that identifies potential problems before they impact your availability or, in worst cases, cause outages. A check is owned, delivered, and supported by the component, element, or product that writes it. Checks are separate from the IBM Health Checker for z/OS framework. A check might analyze a configuration in the following ways:

- Changes in settings or configuration values that occur dynamically over the life of an IPL. Checks that look for changes in these values should run periodically to keep the installation aware of changes.
- Threshold levels approaching the upper limits, especially those that might occur gradually or insidiously.
- Single points of failure in a configuration.
- Unhealthy combinations of configurations or values that an installation might not think to check.

This document discusses the following IBM Health Checker for z/OS concepts:

Check owner and check name: Each check has a check owner and check name.

- The check owner is the owning element or component. For IBM checks, checks, these will all start with IBM. For example, IBMASM and IBMUSS are two IBM check owners.
- The check name is the name of the check itself, such as ASM_NUMBER_LOCAL_DATASETS.

You can find check owner and check name for each check in the list of checks in the front of Chapter 13, "IBM Health Checker for z/OS checks," on page 381.

For the most up-to-date list of IBM-owned check owners and names, see the following Web site:

`http://www.ibm.com/servers/eserver/zseries/zos/hchecker/check_table.html`

Check values: Each check includes a set of pre-defined values, such as:

- Interval, or how often the check will run
- Severity of the check, which influences how check output is issued

- Routing and descriptor codes for the check

You can update or override some check values using either SDSF or statements in the HZSPRMxx parmlib member or the MODIFY command. These are called **installation updates**. You might do this if some check values are not suitable for your environment or configuration.

Check output: A check issues its output as messages, which you can view using SDSF, the HZSPRINT utility, or a log stream that collects a history of check output. If a check finds a potential problem, it issues a WTO message. We will call these messages **exceptions**. The check exception messages are issued both as WTOs and also to the message buffer. The WTO version contains only the message text, while the exception message in the message buffer includes both the text and explanation of the potential problem found, including the severity, as well as information on what to do to fix the potential problem.

Resolving check exceptions: To get the best results from IBM Health Checker for z/OS, you should let it run continuously on your system so that you will know when your system has changed. When you get an exception, you should resolve it using the information in the check exception message or overriding check values, so that you do not receive the same exceptions over and over.

Managing checks: You can use either SDSF, the HZSPRMxx parmlib member, or the IBM Health Checker for z/OS MODIFY (F *hzsproc*) command to manage checks. Managing checks includes:

- Printing check output from either SDSF, or using the HZSPRINT utility - see Chapter 3, “Working with check output,” on page 25.
- Displaying check information
- Taking one time actions against checks, such as:
  - Activating or deactivating checks
  - Add new checks
  - Refresh checks - Refresh processing first deletes a check from the IBM Health Checker for z/OS and then adds it back to the system.
  - Run checks

  See “Cheat sheet: examples of MODIFY ***hzsproc*** commands” on page 47.
- Updating check values temporarily using SDSF or the MODIFY *hzsproc* command. See “Making dynamic, temporary changes to checks” on page 44.
- Updating check values permanently using HZSPRMxx. See “Making persistent changes to checks” on page 52.

For more information, see our fabulous Redpaper!

For lots of details and experience based information about using and writing checks for IBM Health Checker for z/OS, see *Exploiting the Health Checker for z/OS infrastructure* (REDP-4590).

Background for IBM's checks

IBM Health Checker for z/OS check routines look at an installation's configuration or environment to look for potential problems. The values used by checks come from a variety of sources including product documentation and web sites, such as:

- z/OS system test
- z/OS Service

- Parallel Sysplex® Availability Checklist at: http://www.ibm.com/servers/eserver/zseries/pso/
- ITSO Redbooks® at: http://www.redbooks.ibm.com/
- Washington System Center Flashes at: http://www.ibm.com/support/techdocs/.

  For migration to a 64–bit environment, see whitepaper WP100269 "z/OS Performance: Managing Processor Storage in a 64–bit environment", and the Washington System Center Flash 10086, "Software Capacity Planning: Migration to 64 bit Mode".
- Parallel Sysplex and z/OS publications:
  - *z/OS MVS Initialization and Tuning Reference*
  - *z/OS MVS Planning: Global Resource Serialization*
  - *z/OS MVS Planning: Operations*
  - *z/OS MVS Setting Up a Sysplex*
  - *z/OS Security Server RACF Command Language Reference*
  - *z/OS Security Server RACF Security Administrator's Guide*
  - *z/OS UNIX System Services Planning*

The description of each individual check contains the rationale behind the values used by the check for comparison against your installation settings. See Chapter 13, "IBM Health Checker for z/OS checks," on page 381.

You might find that the values that the check uses for comparison are not appropriate for your installation or for a particular system. If that is the case, you can either specify overrides to default values or suppress individual checks. See Chapter 4, "Managing checks," on page 43.

Chapter 2. Setting up IBM Health Checker for z/OS

The IBM Health Checker for z/OS framework provides a structure for checks to gather system information and mechanisms to report their findings. The checks compare the system environment and parameters to established settings to uncover potential problems. When z/OS IPLs, it automatically starts IBM Health Checker for z/OS, but for optimal use, you'll want to do the set-up steps described in this chapter.

Checks are provided separately from the IBM Health Checker for z/OS framework (see "Obtain checks for IBM Health Checker for z/OS" on page 23 for more information on obtaining checks) and individual checks should be assessed for their relevance to your installation. You can override parameters for some checks, and you can override values or deactivate any individual check. See Chapter 4, "Managing checks," on page 43.

This chapter covers the following topics:

- "Stopping and Starting IBM Health Checker for z/OS Manually"
- "Sharing critical IBM Health Checker for z/OS information between systems at different levels" on page 8
- "Steps for optimizing IBM Health Checker for z/OS" on page 9

Note that all the setup steps in "Steps for optimizing IBM Health Checker for z/OS" on page 9 for IBM Health Checker for z/OS are the same no matter what types of checks you run, whether local, remote, or REXX. This includes security definitions. Any check-specific setup steps required are documented with the check description in Chapter 13, "IBM Health Checker for z/OS checks," on page 381.

Stopping and Starting IBM Health Checker for z/OS Manually

Although IBM Health Checker for z/OS starts automatically when you start z/OS, you can still stop and start IBM Health Checker for z/OS manually, to apply service, for example. To stop and start IBM Health Checker for z/OS, use the HZSPROC started procedure in the following commands:

```
STOP hzsproc

START hzsproc,HZSPRM=PREV
```

Specifying HZSPRM=PREV, which is the default in the HZSPROC procedure, ensures that you use the same set of HZSPRMxx parmlib members that were in effect in the previous instance of IBM Health Checker for z/OS.

Note: You can prevent an automatic start of Health Checker at IPL time. You do this by assigning a special value of *NONE to the system parameter HZSPROC in for example the IEASYSxx parmlib member: HZSPROC=*NONE. Installations might choose to use this feature to have better control on when certain address spaces, in this case Health Checker, start and use an automation product to issue an explicit START HZSPROC command at a desired time.

Sharing critical IBM Health Checker for z/OS information between systems at different levels

You can share critical IBM Health Checker for z/OS information between systems at different levels, as follows:

- "Sharing IEASYSxx"
- "Sharing the HZSPROC procedure"
- "Sharing HZSPRMxx"

Sharing IEASYSxx

Prior to z/OS V2R1, there were no IBM Health Checker for z/OS related system parameters in IEASYSxx. Now, with z/OS V2R1, for example, there are two system parameters, HZS and HZSPROC. You'll encounter these as you do the "Steps for optimizing IBM Health Checker for z/OS" on page 9.

So, what happens if you share IEASYSxx parmlib members in a sysplex with systems that are at different levels? You can make this work using the WARNUND system parameter, which tells the system to issue message IEA660I when it encounters unsupported IEASYSxx statements are encountered, rather than stopping an IPL to prompt for a correct statement. That means that a system can use an IEASYSxx member that includes statements that are not supported on the system's release level, simply ignoring the ones the system does not support. The system issues message IEA660I for IEASYSxx statements that the system does not support.

To use the WARNUND statement in IEASYSxx see the Overview of IEASYSxx parameters in *z/OS MVS Initialization and Tuning Reference*.

Sharing the HZSPROC procedure

You can share the IBM Health Checker for z/OS startup procedure, HZSPROC, between systems as long as you make specific parameters specific to the system. A typical system specific parameter is the HZSPDATA dataset. If an HZSPDATA DD statement is specified in the HZSPROC procedure, it should use unique system symbols for the system since the HZSPDATA dataset can not be shared between systems. If HZSPDATA is referenced in one of the HZSPRM proc parm specified HZSPRMxx suffixes, that HZSPRMxx parmlib member should use system symbols in the HZSPDATA statement.

You can share HZSPROC between systems at different release levels. In particular, after V2R1 and later, assuming you use the HZSPDATA DD to specify the HZSPDATA dataset. Prior to V2R1 systems only support the DD, not the HZSPRMxx HZSPDATA statement.

Sharing HZSPRMxx

Often a new release of z/OS includes new statements or parameters for the HZSPRMxx parmlib member. For example, the HZSPDATA statement in HZSPRMxx is new for z/OS V2R1. How does this work if you share HZSPRMxx between systems at different levels? The answer is, it should work fine. Even if you exploit all the new parameters and statements that come down the pike, IBM Health Checker for z/OS simply issues a syntax error message for unsupported statements but processes all the other statements in the HZSPRMxx member.

See “Tell the system which HZSPRMxx members you want to use” on page 21 for more information.

Steps for optimizing IBM Health Checker for z/OS

To optimize your IBM Health Checker for z/OS set-up, you'll want to do the following set-up steps before IPLing z/OS:

- If you want to get IBM Health Checker for z/OS started quickly, see “In a rush? Use these basic steps.”
- Otherwise, use the following steps to set up and start IBM Health Checker for z/OS:
  1. “Customizing the IBM Health Checker for z/OS procedure” on page 10
  2. “Software requirements” on page 10
  3. “Allocate the HZSPDATA data set to save check data between restarts” on page 10
  4. “Optionally set up the HZSPRINT utility” on page 12
  5. “Optionally define log streams to keep a record of the check output” on page 12
  6. “Create security definitions” on page 14
  7. Optionally set up SDSF for IBM Health Checker for z/OS support using the IBM Health Checker for z/OS Small Programming Enhancement sections in *z/OS SDSF Operation and Customization*.
  8. “Create multilevel security definitions” on page 19
  9. “Create HZSPRMxx parmlib members” on page 20 and “Tell the system which HZSPRMxx members you want to use” on page 21
  10. “Assign IBM Health Checker for z/OS to a WLM service class” on page 23
  11. “Obtain checks for IBM Health Checker for z/OS” on page 23

In a rush? Use these basic steps

If you're looking to optimize IBM Health Checker for z/OS quickly and without frills, you can follow the map below for minimum steps. These steps do not include the optional steps for defining log streams to record check output, creating multilevel security definitions, or creating your own HZSPRMxx parmlib member from our sample.

1. “Allocate the HZSPDATA data set to save check data between restarts” on page 10
2. “Create security definitions” on page 14
3. “Obtain checks for IBM Health Checker for z/OS” on page 23

Once you've optimized IBM Health Checker for z/OS, use the HZSPRINT utility, SDSF, or operator commands to view and work with check output. See the following for information:

- “Using the HZSPRINT utility” on page 37
- To set up and use SDSF, see the following:
  - Set up security and customization for SDSF support for IBM Health Checker for z/OS using information in IBM Health Checker for z/OS Small Programming Enhancement in *z/OS SDSF Operation and Customization*.
  - “Using SDSF to manage checks” on page 45

Software requirements

IBM Health Checker for z/OS is shipped as part of z/OS .

IBM Health Checker for z/OS can run on a parallel sysplex, monoplex, or XCF local mode environment running z/OS V1R4 or later. Note that different checks have different system level requirements - see Chapter 13, "IBM Health Checker for z/OS checks," on page 381 for check specific information.

Software requirement for running REXX checks

More and more checks are being developed in System REXX. Because check providers use both REXX and compiled REXX for checks, you must ensure that either the SEAGALT or SEAGLPA library is available in the system search order.

- SEAGALT is provided in z/OS V1R9 and higher
- SEAGLPA is provided in the IBM Library for REXX on System z product

For more information about the SEAGALT and SEAGLPA, see *IBM Compiler and Library for REXX on System z: User's Guide and Reference*.

Customizing the IBM Health Checker for z/OS procedure

IBM Health Checker for z/OS starts automatically when you start z/OS. The system uses the following HZSPROC started procedure to start IBM Health Checker for z/OS:

```
//HZSPROC JOB JESLOG=SUPPRESS
//HZSPROC  PROC HZSPRM='PREV'
//HZSSTEP  EXEC   PGM=HZSINIT,REGION=0K,TIME=NOLIMIT,
//      PARM='SET PARMLIB=&HZSPRM'
//*HZSPDATA DD   DSN=SYS1.&SYSNAME..HZSPDATA,DISP=OLD
//        PEND
//        EXEC HZSPROC
```

Note that although this looks like a batch job it **is a started task**. IBM Health Checker for z/OS is set up this way in order to suppress messages to the JESLOG, which might otherwise overflow your JESLOG data set.

If you rename the HZSPROC procedure, make sure the system knows the new name: By default, the system uses the name HZSPROC for the IBM Health Checker for z/OS procedure. If you rename the procedure, you must specify the name of your *hzsproc* procedure in the HZSPROC system parameter of the IEASYSxx parmlib member.

Optionally customize the HZSPROC procedure: Later during this set-up procedure, you will come back to the HZSPROC procedure to:

- Optionally update the procedure to include the name of the HZSPDATA data set defined in "Allocate the HZSPDATA data set to save check data between restarts."
- "Tell the system which HZSPRMxx members you want to use" on page 21.

Allocate the HZSPDATA data set to save check data between restarts

Some checks use the HZSPDATA data set to save data required as part of their processing between restarts of the system or IBM Health Checker for z/OS. To allocate this data set, do the following:

1. Get the HZSALLCP sample JCL from SYS1.SAMPLIB.

2. Update the HZSALLCP JCL for the HZSPDATA data set. The data set must:
   - Be fixed block
   - Be sequential
   - Have a logical record length of 4096

   You must also specify a high level qualifier for the data set. In the following example, we're using the system name as part of the HZSPDATA data set name:

```
//HZSALLCP JOB
//*
//HZSALLCP EXEC PGM=HZSAIEOF,REGION=4096K,TIME=1440
//HZSPDATA DD  DSN=SYS1.sysname.HZSPDATA,DISP=(NEW,CATLG),
//         SPACE=(4096,(100,400)),UNIT=SYSDA,
//         DCB=(DSORG=PS,RECFM=FB,LRECL=4096)
//SYSPRINT DD DUMMY
```

   Within the HZSPDATA DD statement, you can use any UNIT and VOLSER values supported on your system to indicate where you want the system to allocate the data set.

3. Retain the name of the HZSPDATA data set so you can specify it in either:
   - The HZSPDATA statement in the HZSPRMxx parmlib member. See "Create HZSPRMxx parmlib members" on page 20.
   - The IBM Health Checker for z/OS start up procedure, HZSPROC, as follows:

```
//HZSPROC  PROC HZSPRM='PREV'
//HZSSTEP  EXEC   PGM=HZSINIT,REGION=0K,TIME=NOLIMIT,
//      PARM='SET PARMLIB=&HZSPRM'
//HZSPDATA DD   DSN=SYS1.&SYSNAME..HZSPDATA,DISP=OLD
//        PEND
//        EXEC HZSPROC
```

The very first time IBM Health Checker for z/OS starts, you might see a message such as the following:

```
HZS0010I THE HZSPDATA DATA SET CONTAINS NO RECORDS
```

In this case, the message can be ignored since we know the dataset is empty.

Monitoring and sizing the HZSPDATA data set

The default HZSPDATA data set sizes defined in the HZSALLCP JCL should work well, even if you run lots of checks. (Not all checks use persistent data, so the number of checks is not a great way to predict how much space you need for persistent data.) If you write your own checks or obtain non-IBM checks, you should be able to get information about HZSPDATA data set space requirements from the check writers.

If you are concerned about persistent data usage, rather than blindly experiment with larger HZSPDATA data set sizes, IBM recommends that you monitor persistent data set usage with the `F HZSPROC,DISPLAY` command. The display command output looks as follows:

```
HZS0203I  11.27.40 HZS INFORMATION   FRAME LAST   F      E   SYS=SY39
...
HZSPDATA DSN: dsname VOL: volser
HZSPDATA RECORDS: 18
```

For your data set calculations, note that 1 HZSPDATA record equals 4K of storage. In this example, the DISPLAY command output shows that the system is using 18 records of HZSPDATA space, or 72K, which is well within the default sizes of SPACE=(4096,(100,400)).

If the HZSPDATA data set actually becomes full, IBM Health Checker for z/OS does the following:

- Continues to collect persistent data in memory until a new persistent dataset is available.
- Issues system message HZS0012E to indicate that the data set is full and recommend the size needed to hold your persistent data:

  ```
  HZS0012E HZSPDATA DATA SET IS FULL. DATA SET NEEDS ROOM FOR n 4096-BYTE RECORDS
  ```

- When a new persistent dataset is provided, IBM Health Checker for z/OS writes the current relevant persistent data into the new persistent dataset.

Optionally set up the HZSPRINT utility

The HZSPRINT utility allows you to see check output in the message buffer or a IBM Health Checker for z/OS log stream. HZSPRINT writes the current message buffer for the target checks to SYSOUT. If you wish to use the HZSPRINT utility, do the following set up steps:

1. Get the JCL for the HZSPRINT utility from member HZSPRINT in SYS1.SAMPLIB .
2. "Setting up security for the HZSPRINT utility" on page 16.
3. See "Using the HZSPRINT utility" on page 37.

Optionally define log streams to keep a record of the check output

IBM Health Checker for z/OS retains only the check results from the last iteration of a check in the message buffer. If you want to retain a historical record of check results, which is optional, but a good idea, you can define and connect to a log stream. When you have a log stream connected, the system writes check results to the log stream every time a check completes.

Note that most of our instructions are for coupling facility log streams, which are suggested. For information about setting up a DASD-only logstream, see the "Planning for system logger applications" section of *z/OS MVS Setting Up a Sysplex*

Do the following to define log streams:

1. Plan for setting up log streams, including allocation of coupling facility and DASD space. Careful planning of DASD and coupling facility space is important because if the log stream fills up, no additional data will be written to it and data will be lost. See the "Planning for system logger applications" section of *z/OS MVS Setting Up a Sysplex*. Keep in mind the following:
   - Define either:
     - One log stream for each system.
     - One log stream for multiple systems to use.
   - HZS must be the first letters of log stream names and structures you define. For example, you might define a log stream name of HZSLOG1.
   - System logger requires at least a base sysplex configuration in your installation.

- System logger requires SMS to be active, in at least a null configuration, even if you do not use SMS to manage your volumes and data sets. See the "Plan DASD space for system logger" section of *z/OS MVS Setting Up a Sysplex*.

2. Set up security for log streams:
   a. You will accomplish most of the security set up needed for the log stream when you set up security for the IBM Health Checker for z/OS super User ID in “Setting up security for the IBM Health Checker for z/OS started task” on page 14.
   b. The user who will be setting up log stream and structure definitions for the IBM Health Checker for z/OS log stream using the IXCMIAPU Administrative Data Utility program must have authorization to a number of resources. See the Define Authorization for system logger resources section of *z/OS MVS Setting Up a Sysplex*.
   c. See “Security for printing check output from a log stream” on page 19 to set up security access for users to the HZSPRINT output, if you will be using HZSPRINT to print log stream data.
3. Enable log streams in one of the following ways:
   - Use the MODIFY command:

     ```
     f hzsproc,logger=on,logstreamname=logstreamname
     ```
   - Use the LOGGER parameter in the HZSPRMxx parmlib member:

     ```
     LOGGER(ON) LOGSTREAMNAME(logstreamname)
     ```

     To disable a log stream, issue the following MODIFY command:

     ```
     f hzsproc,logger=off
     ```
4. To display check output from a log stream, use either:
   - The HZSPRINT utility - See “Optionally set up the HZSPRINT utility” on page 12.
   - SDSF - See “Using SDSF to manage checks” on page 45.

The following examples show our log stream definitions:

- **CFRM policy definition:** The following example shows a log stream and structure definition defined in the CFRM policy using the administrative data utility, IXCMIAPU:

  ```
  STRUCTURE NAME(HZS_HEALTHCHKLOG) SIZE(9000)
     PREFLIST(CF25, CF01C, CF1)
  ```

  The value defined for SIZE should be no less than 8000 to ensure adequate space for check data. For more information, see:
  - "Define the coupling facility structures attributes in the CFRM policy couple data set" in*z/OS MVS Setting Up a Sysplex*
  - IBM recommends that you use the following web-based CFSizer tool to estimate an appropriate structure size

    ```
    http://www.ibm.com/systems/support/z/cfsizer/
    ```

    .
- **LOGR policy definitions:**
  - The following example shows a coupling facility and log stream structure definition in the LOGR policy using the administrative data utility, IXCMIAPU:

    ```
    DEFINE STRUCTURE NAME(HZS_HEALTHCHKLOG)
           LOGSNUM(1)
           MAXBUFSIZE(65532)
    ```

```
        AVGBUFSIZE(1024)

DEFINE LOGSTREAM NAME(HZS.HEALTH.CHECKER.HISTORY)
          DESCRIPTION(HEALTH_CHECK_RPT)
          STRUCTNAME(HZS_HEALTHCHKLOG)
          STG_DUPLEX(NO)
          LS_DATACLAS(NO_LS_DATACLAS)
          LS_MGMTCLAS(NO_LS_MGMTCLAS)
          LS_STORCLAS(NO_LS_STORCLAS)
          LS_SIZE(4096)
         AUTODELETE(YES)
         RETPD(14)
         HIGHOFFLOAD(80)
         LOWOFFLOAD(0)
         DIAG(NO)
```

Note that the IBM Health Checker for z/OS structure and log stream names must begin with HZS.

– The following example shows a DASD-only log stream definition in the CFRM policy using the administrative data utility, IXCMIAPU. Note that the values you define for a DASD-only log stream in your installation may be different.

```
DEFINE LOGSTREAM NAME (HZS.HEALTH.CHECKER.HISTORY)
     DASDONLY(YES)
     MAXBUFSIZE(65532)
     HIGHOFFLOAD(80)
     LOWOFFLOAD(20)
     STG_SIZE(2000)
     LS_SIZE(1000)
     LS_DATACLAS(lsdataclas)
     LS_STORCLAS(lsstorclas)
     STG_DATACLAS(stgdataclas)
     STG_STORCLAS(stgstorclas)
```

For more information on the LOGR couple data set, see "Add information about log streams and coupling facility structures to the LOGR policy" section of *z/OS MVS Setting Up a Sysplex*.

Create security definitions

Both IBM Health Checker for z/OS and users looking at check output need access to resources. You must create security definitions to control access and maintain security for these resources.

You must do the following types of security setup:

- "Setting up security for the IBM Health Checker for z/OS started task"
- "Setting up security for the HZSPRINT utility" on page 16
- "Setting up security for IBM Health Checker for SDSF support" on page 19

Setting up security for the IBM Health Checker for z/OS started task

You must set up security for IBM Health Checker for z/OS the same way you would for any other started task. To do this task with RACF®, do the following steps:

1. Create a user ID for IBM Health Checker for z/OS and connect the superuser user ID to a group. Define the user ID with:
   - Superuser authority using either:
     – UID(0) explicitly assigned to the user ID.

– Access to the BPX.SUPERUSER resource. The advantage of this method is that it might be more audit friendly, because you avoid having a user profile with UID(0) explicitly assigned to it.

  At runtime, IBM Health Checker for z/OS dynamically switches to (and stays in) an effective UID(0) superuser authority using the defined BPX.SUPERUSER access.

- A home directory of HOME('/')
- A program of PROGRAM('/bin/sh')

Examples:

- Using UID(0), you might use the following commands to define the user ID as follows:

```
ADDUSER hcsuperid
        OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
        NOPASSWORD
ADDGROUP OMVSGRP OMVS(GID(xx))
CONNECT hcsuperid GROUP(OMVSGRP)
```

- Using access to the BPX.SUPERUSER resource, you might use the following commands to define the user ID as follows:

```
ADDUSER hcsuperid OMVS(UID(yy) HOME('/') PROGRAM('/bin/sh')) NOPASSWORD
ADDGROUP OMVSGRP OMVS(GID(xx))
CONNECT hcsuperid GROUP(OMVSGRP)
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(hcsuperid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

For more information, see:

- Assigning superuser attributes in *z/OS UNIX System Services Planning*
- *z/OS Security Server RACF Security Administrator's Guide*
- The ADDGROUP and ADDUSER sections of *z/OS Security Server RACF Command Language Reference*

Note: Once you start IBM Health Checker for z/OS with its associated User ID, changes you make to the UID for the User ID won't usually take effect until the IBM Health Checker for z/OS address space is stopped and restarted.

2. Associate the superuser User ID, *hcsuperid*, with the IBM Health Checker for z/OS started task, HZSPROC. For example:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED HZSPROC.* STDATA(USER(hcsuperid) GROUP(OMVSGRP))
SETROPTS CLASSACT(STARTED)
SETROPTS RACLIST(STARTED)
```

   If you had already RACLISTed the STARTED class, the last statement would have to be `SETROPTS RACLIST(STARTED) REFRESH`. For more information, see:

   - *z/OS Security Server RACF Security Administrator's Guide* and *z/OS UNIX System Services Planning*.
   - The RDEFINE and SETROPTS sections of *z/OS Security Server RACF Command Language Reference*.

3. Give the IBM Health Checker for z/OS started task super User ID access to the HZSPDATA data set on each system where you'll run IBM Health Checker for z/OS. For example, you might specify the following:

```
ADDSD 'SYS1.PRODSYS.HZSPDATA' UACC(NONE)
PERMIT SYS1.PRODSYS.HZSPDATA CLASS(DATASET) ID(hcsuperid) ACCESS(UPDATE)
```

4. Give IBM Health Checker for z/OS started task super User ID READ access to the HZSPRMxx parmlib member(s). For example, you might specify the following:

   ```
   ADDSD 'SYS1.PARMLIB' UACC(NONE)
   PERMIT 'SYS1.PARMLIB' CLASS(DATASET) ID(hcsuperid) ACCESS(READ)
   ```

5. If you will be using a log stream, you must define UPDATE access for the IBM Health Checker for z/OS started task super User ID to each RESOURCE(*logstreamname*) CLASS(LOGSTRM). IBM Health Checker for z/OS connects directly to the defined log stream or streams. For example, you might specify the following:

   ```
   RDEFINE LOGSTRM logstreamname UACC(NONE)
   PERMIT logstreamname CLASS(LOGSTRM) ID(hcsuperid) ACCESS(UPDATE)
   SETROPTS CLASSACT(LOGSTRM) RACLIST(LOGSTRM)
   SETROPTS RACLIST(LOGSTRM)
   ```

   If you had already RACLISTed the LOGSTRM class, the last statement would have to be `SETROPTS RACLIST(LOGSTRM) REFRESH`.

   See the "LOGR parameters for administrative data utility section of *z/OS MVS Setting Up a Sysplex*.

6. REXX health checks support input and output datasets and the checks have a REXXHLQ (REXX dataset high level qualifier) attribute. Be prepared to grant the appropriate access rights for REXX datasets to the user ID that is associated with the Health Checker address space.
7. If the SERVAUTH class is activated and a profile is defined for the EZB.STACKACCESS.*sysname.tcpprocname* resource, you must grant the user ID that is associated with the Health Checker address space READ access to the profile.

   ```
   PERMIT   EZB.STACKACCESS.sysname.tcpprocname CLASS(SERVAUTH) ID(hcsuperid) ACCESS(READ)
   SETROPTS GENERIC(SERVAUTH) REFRESH
   SETROPTS RACLIST(SERVAUTH) REFRESH
   ```

8. To let health check (IBMUSS,ZOSMIGREC_ROOT_FS_SIZE) run successfully, give the Health Checker user ID READ access to the OPERCMDS MVS.DISPLAY.OMVS resource.

Setting up security for the HZSPRINT utility

IBM Health Checker for z/OS users can view check output in the message buffer or log stream using HZSPRINT. HZSPRINT writes the check output for the target checks to SYSOUT. If users in your installation will be using HZSPRINT to print check output, you must authorize HZSPRINT users to the following resources:

- To access check output from the **message buffer**, you must authorize users to the following service resources:
  - QUERY, which returns a list of checks and check status from the message buffer.
  - MESSAGES, which returns the output messages for a check or checks from the message buffer.

  See “Security for printing check output from the message buffer” on page 17.
- To access check output in IBM Health Checker for z/OS **log stream** or streams, you must authorize users to the log stream names. See “Security for printing check output from a log stream” on page 19.

To authorize HZSPRINT users to these service resources with RACF, you must define profiles for them, as shown in the topics below.

Security for printing check output from the message buffer

Users accessing check output from the message buffer, must have authorization to the QUERY and MESSAGES service resources using RACF profiles. The way you define RACF profiles depends on:

- The way users specify the check name and check owner in the HZSPRINT EXEC PARM= statement.
- The level of access you wish to give to the user.

Specifying check name and owner in the HZSPRINT EXEC PARM= statement: Depending on what access level they have and what check output they want, users can specify the exact check name and check owner in the EXEC statement to get output from one check or they can use wildcard characters to get output for multiple checks.

The syntax for the HZSPRINT EXEC statement for printing check output from the message buffer is as follows:

```
//  EXEC PGM=HZSPRNT,PARM='CHECK(check_owner,check_name)'
```

See "What is a check?" on page 4 for how to find the check owner and check name for checks.

The following HZSPRINT EXEC statement examples show different ways users can specify the check name and the check owner to get different output:

- To get check output for all active checks, use the following EXEC statement:

  ```
  //  EXEC PGM=HZSPRNT,PARM='CHECK(*,*)'
  ```

- To get check output for all the checks owned by IBMGRS, use the following EXEC statement:

  ```
  //  EXEC PGM=HZSPRNT,PARM='CHECK(IBMGRS,*)'
  ```

- To get check output for just one check, IBMGRS check GRS_Mode, use the following EXEC statement:

  ```
  //  EXEC PGM=HZSPRNT,PARM='CHECK(IBMGRS,GRS_Mode)'
  ```

- To get check output for all the checks named TRY_ME by any check owner, use the following EXEC statement:

  ```
  //  EXEC PGM=HZSPRNT,PARM='CHECK(*,TRY_ME)'
  ```

See Chapter 3, "Working with check output," on page 25 for complete information about using HZPRINT.

Determining the access level required for check name and owner specification on the HZSPRINT EXEC statement: The table below shows the access required for different user specifications of the check name and owner in the HZSPRINT EXEC PARM= statement, including the resource name or names that must be defined in the XFACILIT class for that particular specification. You must also RACLIST the XFACILIT class in order for HZSPRINT to work, as shown in the examples below the table.

Where we show two possible resource names you can define for a service resource, the system accepts a match on either.

Table 1. Access required for printing check output from the message buffer using HZSPRINT

Check specification	Access required for service resource	Resource names
CHECK(*,*checkname*) CHECK(*,*)	QUERY: Read access to all checks	• HZS.*sysname*.QUERY
	MESSAGES: Read access to individual check	• HZS.*sysname.check_owner*.MESSAGES • HZS.*sysname.check_owner.check_name*.MESSAGES
CHECK(*checkowner*,*)	QUERY: Read access to all checks for a specific owner	• HZS.*sysname.check_owner*.QUERY
	MESSAGES: Read access to individual check	• HZS.*sysname.check_owner*.MESSAGES • HZS.*sysname.check_owner.check_name*.MESSAGES
CHECK(*checkowner*, *checkname*)	QUERY: Read access to individual check	• HZS.*sysname.check_owner*.QUERY • HZS.*sysname.check_owner.check_name*.QUERY
	MESSAGES: Read access to individual check	• HZS.*sysname.check_owner*.MESSAGES • HZS.*sysname.check_owner.check_name*.MESSAGES

Defining RACF profiles for QUERY and MESSAGE service resources: For each resource name identified in the first table, issue:

```
RDEFINE XFACILIT resourcename UACC(NONE)
PERMIT resourcename CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
```

Then, issue the following for the XFACILIT class:

```
SETROPTS CLASSACT(XFACILIT)
SETROPTS RACLIST(XFACILIT)
```

If you already RACLISTed the XFACILIT or FACILITY class, the very last statement in the example above would have to be:

```
SETROPTS RACLIST(XFACILIT) REFRESH
```

Profile definition examples:

The following table shows examples of defining access profiles for the QUERY and MESSAGES service resources in the XFACILIT class to allow a user ID to access check output in HZSPRINT.

In these examples, *hcprintid* is the user ID of either a user or group you're giving access to.

- **Access to output from all checks:**

  ```
  RDEFINE XFACILIT  HZS.sysname.QUERY UACC(NONE)
  PERMIT  HZS.sysname.QUERY CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
  RDEFINE XFACILIT HZS.sysname.check_owner.MESSAGES UACC(NONE)
  PERMIT  HZS.sysname.check_owner.MESSAGES CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
  SETROPTS CLASSACT(XFACILIT)
  SETROPTS RACLIST(XFACILIT)
  ```

- **Access to output from a specified check owner:**

  ```
  RDEFINE XFACILIT HZS.sysname.check_owner.QUERY UACC(NONE)
  PERMIT HZS.sysname.check_owner.QUERY CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
  RDEFINE XFACILIT HZS.sysname.check_owner.check_name.MESSAGES UACC(NONE)
  PERMIT HZS.sysname.check_owner.check_name.MESSAGES CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
  SETROPTS CLASSACT(XFACILIT)
  SETROPTS RACLIST(XFACILIT)
  ```

- **Access to output from a particular check:**

```
RDEFINE XFACILIT HZS.sysname.check_owner.check_name.QUERY UACC(NONE)
PERMIT  HZS.sysname.check_owner.check_name.QUERY CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
RDEFINE XFACILIT HZS.sysname.check_owner.check_name.MESSAGES UACC(NONE)
PERMIT  HZS.sysname.check_owner.check_name.MESSAGES CLASS(XFACILIT) ID(hcprintid) ACCESS(READ)
SETROPTS CLASSACT(XFACILIT)
SETROPTS RACLIST(XFACILIT)
```

For more information, see:

- *z/OS Security Server RACF Security Administrator's Guide* and *z/OS UNIX System Services Planning* .
- The PERMIT, RDEFINE and SETROPTS sections of *z/OS Security Server RACF Command Language Reference*.
- "Using the HZSPRINT utility" on page 37 for information on using HZSPRINT.

Security for printing check output from a log stream

If you use an IBM Health Checker for z/OS log stream to collect check output, you can use HZSPRINT to print the log stream data using one of the following HZSPRINT EXEC statement examples:

```
//  EXEC PGM=HZSPRINT,PARM='LOGSTREAM(logstreamname)'
                    OR
//  EXEC PGM=HZSPRINT,PARM='LOGSTREAM(logstreamname),CHECK(owner,name)'
                    OR
//  EXEC PGM=HZSPRINT,PARM='LOGSTREAM(logstreamname),CHECK(owner,name),EXCEPTIONS'
```

To authorize HZSPRINT users to log stream check output, you must define a profile in the LOGSTRM class for the log stream and assign READ access to users. When you assign access to the log stream for an HZSPRINT user, you give the user access to all check output in the log stream. HZSPRINT access to log streams is all or nothing - you cannot restrict HZSPRINT access to particular check owners or checks in log streams, as you can for check output in the message buffer.

The following profile example shows how you might define HZSPRINT access for a user ID to check output in a log stream:

```
RDEFINE FACILITY log_stream_data_set_name UACC(NONE)
PERMIT log_stream_data_set_name CLASS(LOGSTRM) ID(hcprint) ACCESS(READ)
SETROPTS CLASSACT(LOGSTRM)
SETROPTS RACLIST(LOGSTRM) REFRESH
```

Setting up security for IBM Health Checker for SDSF support

If your installation uses SDSF, set up customization and security for SDSF support for IBM Health Checker for z/OS using Protecting checks in *z/OS SDSF Operation and Customization*.

Create multilevel security definitions

If your system is a multilevel system environment and you are using multilevel security labels to control access to resources, you must assign SECLABELs to the IBM Health Checker for z/OS superuser User ID (*hcsuperid*), to each profile protecting a check, and to the IBM Health Checker for z/OS log stream RACF profile. For complete information on multilevel security, see *z/OS Planning for Multilevel Security and the Common Criteria* and *z/OS Security Server RACF Security Administrator's Guide*.

Do the following:

- Assign a multilevel security label to the IBM Health Checker for z/OS superuser User ID, *hcsuperid*, which you defined in"Setting up security for the IBM Health Checker for z/OS started task" on page 14. Use the following to decide on a SECLABEL setting for the log stream:

- If all your checks are assigned a SECLABEL of SYSLOW, assign a SECLABEL of SYSLOW to the IBM Health Checker for z/OS superuser User ID, *hcsuperid*. Assigning a SECLABEL of SYSLOW to the *hcsuperid* means that any data object that the check touches must have a SECLABEL that would pass the mandatory access check for the type of operation that is being performed.
- If all the checks are above SYSLOW, you must assign a SECLABEL that will dominate all the check SECLABELs to the *hcsuperid*.
- You can also assign a SECLABEL of SYSHIGH to the *hcsuperid*, which will dominate all the check SECLABELs.

The following example enables the SECLABEL class and assigns a multilevel security label of SYSLOW:

```
SETROPTS CLASSACT(SECLABEL) RACLIST(SECLABEL)
ALTUSER hcsuperid SECLABEL(SYSLOW)
```

- Assign a SECLABEL to each profile that protects a check. See Chapter 13, "IBM Health Checker for z/OS checks," on page 381 for the SECLABEL recommended for each check. You'll need to define access to one of the following set of resources:

  - HZS.*sysname*.*check_owner*.QUERY
    HZS.*sysname*.*check_owner*.MESSAGES
    or
  - HZS.*sysname*.*check_owner*.*check_name*.QUERY
    HZS.*sysname*.*check_owner*.*check_name*.MESSAGES

  For example, you might define the following:

```
RALTER XFACILIT HZS.SYS1.IBMRACF.RACF_GRS_RNL.QUERY UACC(NONE) SECLABEL(SYSLOW)
RALTER XFACILIT HZS.SYS1.IBMRACF.RACF_GRS_RNL.MESSAGES UACC(NONE) SECLABEL(SYSLOW)
```

- Assign a SECLABEL to the IBM Health Checker for z/OS log stream RACF profile. Use the following to decide on a SECLABEL setting for the log stream:
  - If all your checks writing to the log stream are SYSLOW, assign a SECLABEL of SYSLOW to the log stream RACF profile.
  - If all the checks are above SYSLOW, you must assign a SECLABEL that will dominate all the check SECLABELs to the log stream RACF profile.
  - You can also assign a SECLABEL of SYSHIGH to the log stream RACF profile, a SECLABEL which will dominate all the check SECLABELs.

  For example, you might define the following:

```
RALTER FACILITY HZS.HEALTH.CHECKER.HISTORY UACC(NONE) SECLABEL(SYSLOW)
```

Create HZSPRMxx parmlib members

You do not have to set up an HZSPRMxx parmlib member to get IBM Health Checker up and running. And at first, you might want to run IBM Health Checker for z/OS without modifying the HZSPRMxx member to see what check output you get on your installation. Later, as you evaluate your check output, you can use it to make permanent updates in policy statements.

The HZSPRMxx parmlib member lets you make permanent updates in policy statements to check values and parameters or to keep a check from running (deactivating the check). Your HZSPRMxx parmlib member should include **only**:

- Policy statements, to make changes that are applied to checks that are added or refreshed.
- The LOGGER parameter, if you want to use a log stream:

LOGGER(ON) LOGSTREAMNAME(*logstreamname*)

- An HZSPDATA parameter identifying the HZSPDATA data set used to save data required as part of their processing between restarts of the system or IBM Health Checker for z/OS.

Including other non-policy statements in your HZSPRMxx member is ineffective, because the system processes the parmlib member specified in the *hzsproc* procedure or IEASYSxx parmlib member before any checks are added or begin running.

You can use sample parmlib member HZSPRM00 to create your own HZSPRMxx parmlib member. To create the policy statements for your HZSPRMxx parmlib member, you can also use input from:

- The **Parameters accepted** portion of each check description in Chapter 13, "IBM Health Checker for z/OS checks," on page 381.
- "Making persistent changes to checks" on page 52

Then, specify the HZSPRMxx parmlib member or members you want the system to use on the HZS parameter of IEASYSxx, as described in "Tell the system which HZSPRMxx members you want to use."

Tell the system which HZSPRMxx members you want to use

Once you have created one or more HZSPRMxx members, IBM recommends that you define them for use by IBM Health Checker for z/OS when it starts automatically at IPL time by doing the following:

1. Specify the desired HZSPRMxx suffixes on the HZS system parameter in the IEASYSxx parmlib member as follows:

   HZS={*xx*|(*xx*,...,*zz*)}

2. In your *hzsproc* procedure, default to or define HZSPRM=PREV to specify that IBM Health Checker for z/OS use the same HZSPRMxx parmlib members as it did for the previous instance of IBM Health Checker for z/OS:

   ```
   //HZSPROC  PROC HZSPRM='PREV'
   //HZSSTEP  EXEC   PGM=HZSINIT,REGION=0K,TIME=NOLIMIT,
   //      PARM='SET PARMLIB=&HZSPRM'
   //*HZSPDATA DD   DSN=SYS1.&SYSNAME..HZSPDATA,DISP=OLD
   //        PEND
   //        EXEC HZSPROC
   ```

   IBM recommends specifying 'HZSPRM=PREV' to make occasional manual restarts (after applying service, for example) easy and consistent.

Having explained the recommendation, we show the full list of possible HZSPRM parameter values for your *hzsproc* procedure below:

HZSPRM='PREV'
Specifies that your *hzsproc* procedure use the HZSPRMxx suffixes, if any, used by the previous instance of IBM Health Checker for z/OS within the current IPL. HZSPRM='PREV' is used as the default in the standard HZSPROC procedure.

HZSPRM='PREV'behaves like HZPRM='SYSPARM'when the system encounters it at initial IPL time (the first use of the *hzsproc*), because there is no previous instance of IBM Health Checker for z/OS to use at that time.

HZSPRM='SYSPARM'
Specifies that your *hzsproc* procedure use the HZSPRMxx suffixes specified on the HZS system parameter in IEASYSxx.

HZSPRM='NONE'
Specifies that the *hzsproc* use no HZSPRMxx parmlib members

HZSPRM={'*xx***|(***xx***,...,***zz***)'}**
Specify the specific suffixes for the HZSPRMxx parmlib member or members that you wish the *hzsproc* procedure to use.

Note: The HZSPRMxx suffixes you specify, explicitly or implicitly, in the HZSPRM parameter of your *hzsproc* procedure override any suffixes defined in the HZS system parameter in IEASYSxx.

If you should happen to leave a manual `START HZSPROC,HZSPRM=` command (in COMMNDxx, for example), the HZSPRMxx parmlib members activated by the IBM Health Checker for z/OS automatic startup will still win out. Any subsequent manual `START HZSPROC` is ignored and rejected, as long as the current Health Instance has not been stopped.

How do the HZSPRMxx settings specified in HZSPROC and IEASYSxx interact?

As mentioned above, we recommend that you specify the HZSPRMxx suffixes you want to use on the HZS system parameter of IEASYSxx and then specify or default to HZSPRM=PREV in your *hzsproc* procedure. However, for those of you who like either to live creatively or to see the fine details of how this all works, we include the following table of how the HZSPRMxx suffixes specified on the HZS system parameter of IEASYSxx and in your *hzsproc* procedure interact when IBM Health Checker for z/OS starts automatically or is manually restarted:

Table 2. Interaction of HZSPRMxx settings specified in HZSPROC and IEASYSxx

IEASYSxx setting	HZSPROC procedure setting	HZSPRMxx parmlib members used
HZS - no suffixes specified	HZSPRM=PREV	No HZSPRMxx parmlib members used
	HZSPRM=SYSPARM	No HZSPRMxx parmlib members used
	HZSPRM=NONE	No HZSPRMxx parmlib members used
	HZSPRM={*xx*|(*xx*,...,*zz*)}	HZSPRMxx parmlib members specified in HZSPROC used
HZS={*xx*|(*xx*,...,*zz*)}	HZSPRM=PREV	Either the HZSPRMxx parmlib members used in the last instance of IBM Health Checker for z/OS, or the ones specified on system parameter HZS=, when IBM Health Checker for z/OS starts at IPL time.
	HZSPRM=SYSPARM	HZSPRMxx parmlib members specified in IEASYSxx used
	HZSPRM=NONE	No HZSPRMxx parmlib members used
	HZSPRM={*xx*|(*xx*,...,*zz*)}	HZSPRMxx parmlib members specified in HZSPROC used

See also “Sharing critical IBM Health Checker for z/OS information between systems at different levels” on page 8.

Assign IBM Health Checker for z/OS to a WLM service class

Assign IBM Health Checker for z/OS to a WLM service class and make sure that it has a priority no lower than the one your installation uses for performance monitoring products like RMF. See the section on defining service classes and performance goals in *z/OS MVS Planning: Workload Management*.

Obtain checks for IBM Health Checker for z/OS

Checks for the IBM Health Checker for z/OS are delivered both as an integrated part of a z/OS release or separately, as PTFs. Many new and updated checks will be distributed as PTFs, so that they are not dependent on z/OS release boundaries and can be added at any time.

- For an up-to-date list of checks available, see

  `http://www.ibm.com/servers/eserver/zseries/zos/hchecker/check_table.html`
- To obtain checks that have been provided in PTFs, use the Enhanced Preventive Service Planning Tool, available at:

  `http://techsupport.services.ibm.com/390/psp_main.html`

  You can identify checks by selecting type **Function** and category **Health Checker**.

For more information on PSP Buckets, see *z/OS Planning for Installation*.

See also "Dynamically adding local or REXX exec checks to IBM Health Checker for z/OS" on page 196.

Current checks at the time this document was published are described in Chapter 13, "IBM Health Checker for z/OS checks," on page 381.

Chapter 3. Working with check output

Once you've set up IBM Health Checker for z/OS, started it, and obtained some checks, you'll want to look at your check output. Output from checks is in the form of messages issued by check routines, as either:

- **Exception messages** issued when a check detects a potential problem or a deviation from a suggested setting. See "Understanding exception messages" on page 28.
- **Information messages** issued to the message buffer to indicate either a clean check run (no exceptions found) or that a check is inappropriate in the current environment and will not run.
- **Reports** issued to the message buffer, often as supplementary information for an exception message.

You can view complete check output messages in the message buffer using the following:

- **The HZSPRINT utility** to write the current message buffer for the target checks to the specified SYSOUT data set. See "Optionally set up the HZSPRINT utility" on page 12 and "Using the HZSPRINT utility" on page 37.
- **SDSF** - see "Using SDSF to manage checks" on page 45
- **A log stream** - see "Optionally define log streams to keep a record of the check output" on page 12

A check can issue a number of different messages, usually issuing at least one:

- **When a check runs without finding an exception**, it should issue an informational message with that information to the message buffer. The following example shows a clean check run case, viewed in the message buffer:

```
CHECK(IBMRSM,RSM_MAXCADS)
START TIME: 06/07/2005 10:55:38.139127
CHECK DATE: 20041006  CHECK SEVERITY: MEDIUM
CHECK PARM: THRESHOLD(80%)


IARH108I The current number of in use CADS entries is 17, which
represents 34% of the total allowed CADS entries of 50. The highest
usage of CADS entries during this IPL is 34%, or 17 total entries. This
is below the current IBMRSM supplied threshold of 80%.

END TIME: 06/07/2005 10:55:38.139653  STATUS: SUCCESSFUL
```

  Note that the status of the check - **STATUS: SUCCESSFUL**.

- **When a check is not appropriate for the current environment**, it should issue an informational message with that information to the message buffer:

```
CHECK(IBMCNZ,CNZ_SYSCONS_MSCOPE)
START TIME: 02/07/2008 11:27:08.812840
CHECK DATE: 20040816  CHECK SEVERITY: MEDIUM

HZS1003E CHECK(IBMCNZ,CNZ_SYSCONS_MSCOPE):
THE CHECK IS NOT APPLICABLE IN THE CURRENT SYSTEM ENVIRONMENT.

CNZHF1004I The system console is not present. The check is not
applicable in this environment.

END TIME: 02/07/2008 11:27:08.813496  STATUS: ENV N/A
```

- **When a check finds an exception** to a suggested value, or another potential problem, the check issues an exception message. The exception message might be accompanied by supporting information in report format. For an exception message, the system issues a WTO with just the message text by default. The system issues both the message text **and** details buffer. The example below shows an exception message in the message buffer:

```
CHECK(IBMCNZ,CNZ_CONSOLE_MSCOPE_AND_ROUTCODE)
START TIME: 01/31/2008 08:57:01.163404
CHECK DATE: 20040816  CHECK SEVERITY: LOW

* Low Severity Exception *

CNZHF0003I One or more consoles are configured with a combination of
message scope and routing code values that are not reasonable.

  Explanation:  One or more consoles have been configured to have a
    multi-system message scope and either all routing codes or all
    routing codes except routing code 11. Note: For MCS and SMCS
    consoles, only the consoles which are defined on this system are
    checked. All EMCS consoles are checked.

  System Action:  The system continues processing.

  Operator Response:  Report this problem to the system programmer.

  System Programmer Response:  To view the attributes of all consoles,
    issue the following commands:
            DISPLAY CONSOLES,L,FULL
            DISPLAY EMCS,FULL,STATUS=L
    Update the MSCOPE or ROUTCODE parameters of MCS and SMCS consoles on
    the CONSOLE statement in the CONSOLxx parmlib member before the next
    IPL. For EMCS consoles (or to have the updates to MCS/SMCS consoles
    in effect immediately), you may update the message scope and routing
    code parameters by issuing the VARY CN system command with either
    the MSCOPE, DMSCOPE, ROUT or DROUT parameters. Note: The VARY CN
    system command can only be used to set the attributes of an active
    console. If an EMCS console is not active, find out which product
    activated it and contact the product owner.  Effective with z/OS
    V1R7, you can use the EMCS console removal service (IEARELEC in
    SYS1.SAMPLIB) to remove any EMCS console definition that is no
    longer needed.

  Problem Determination:  n/a

  Source:  Consoles (SC1CK)

  Reference Documentation:
            z/OS MVS Initialization and Tuning Reference
            z/OS MVS System Commands
            z/OS MVS Planning: Operations

  Automation:  n/a

  Check Reason:  Reduces the number of messages sent to a console in the
    sysplex

END TIME: 01/31/2008 08:57:01.197807  STATUS: EXCEPTION-LOW
```

In this section, we'll cover the following:

- "Hey! My system has been configured like this for years, and now I'm receiving exceptions!" on page 27
- "Understanding system data issued with the check messages" on page 27
- "Understanding exception messages" on page 28

- "Evaluating check output and resolving exceptions" on page 30
- "Approaches to automation with IBM Health Checker for z/OS" on page 32
- "Understanding check state and status" on page 34
- "Using the HZSPRINT utility" on page 37
- "Finding check message documentation with LookAt" on page 41

Hey! My system has been configured like this for years, and now I'm receiving exceptions!

Some customers may be startled by the exception messages that IBM Health Checker for z/OS issues on systems that have been running just fine the way they were. But it's really worth your time and attention to look over the exceptions and evaluate your system, because IBM Health Checker for z/OS reflects suggestions to improve your system's availability and avoid problems. The checks reflect generally accepted recommendations, but you will need to evaluate whether each suggestion is appropriate for your system.

One important thing to note is that **an exception does not imply that there is a problem to report to IBM**. Exceptions are a means for you to evaluate potential availability impacts and take action. See "Evaluating check output and resolving exceptions" on page 30 for how to resolve check exceptions.

Understanding system data issued with the check messages

In the examples of check messages in other topics, you probably noticed data above and below the check messages - IBM Health Checker for z/OS issues this system data to accompany each check message. Fields such as **START TIME:**, **CHECK DATE:**, and **END TIME:** are not part of the message input specified by the check developer. The system issues this data automatically, as appropriate.

The example below shows a subset of some system data you might see with a check message. The system data is highlighted in bold:

```
CHECK(IBMRSM,RSM_MAXCADS)
SYSPLEX:    PLEX1     SYSTEM: SY39
START TIME: 05/15/2013 13:23:58.867122
CHECK DATE: 20041006  CHECK SEVERITY: MEDIUM
CHECK PARM: THRESHOLD(80%)


IARH108I The current number of in use CADS entries is 17, which
represents 34% of the total allowed CADS entries of 50. The highest
usage of CADS entries during this IPL is 34%, or 17 total entries. This
is below the current IBMRSM supplied threshold of 80%.

END TIME: 05/15/2013 13:23:58.867225  STATUS: SUCCESSFUL
```

Most of the system data fields you might see, such as `START TIME:` and `END TIME:` are self-explanatory. However, the list below includes fields that might need a little explanation:

CHECK(*check_owner*,*check_name)*
: The CHECK field displays the owning component or product for the check, as well as the check name. In this example, IBMRSM or RSM is the owner of the check that issued this message. The SYSPLEX and SYSTEM fields further qualify the check name and indicate where exactly this check ran. This can be helpful for example when viewing the message buffer via the SDSF CK panel while it shows checks from multiple systems in one panel.

CHECK SEVERITY: *severity*
This field displays the severity defined for the check that issued the message.

CHECK PARM: *parameter*
This field displays the parameters that are passed to the check routine when it runs.

STATUS: *status*
The STATUS field shows the status of the check when it completed running. There are many status values possible for a check, as shown in display output or the message buffer. See the status field in message HZS0200I in *z/OS MVS System Messages, Vol 6 (GOS-IEA)* for a list of all the possible values for the check status.

ABENDED. TIME: *time* **DIAG:** *sdwaabcc_sdwacrc*
In the event that a check abends, the system will issue this line along with the check message. In this line of data:

TIME: *time*
The time the check abended.

DIAG: *sdwaabcc_sdwacrc*
sdwaabcc is the abend code, and *sdwacrc* is the abend reason code. *sdwacrc* will display zeros if there is no abend reason code for the abend. See *z/OS MVS System Codes* for information on abends.

Understanding exception messages

Exception messages are the most important check output, because they identify potential problems and suggest a solution.

- The complete explanation and details for exception messages are issued to the message buffer, where you can view it with either SDSF, HZSPRINT, or in the log stream.
- By default, the exception message text is also issued as a WTO, prefaced by an HZS WTO message. The HZS message issued reflects the "SEVERITY={HIGH | MEDIUM | LOW | NONE}" on page 79 and "WTOTYPE={CRITICAL|EVENTUAL|INFORMATIONAL|HARDCOPY|NONE}" on page 80 parameters defined for the check. (You can update these parameters to control the severity and descriptor code for the check.)

The following examples show how exception messages and exception message WTOs will look on a system:

Exception message example 1 - An exception message as it appears in the message buffer: The following example shows an exception message in the message buffer. Note that IBM Health Checker for z/OS issues information both before and after the exception message with data including the check owner and name, the severity of the check, and the check parameter in use.

```
CHECK(IBMGRS,GRS_MODE)
START TIME: 06/12/2007 18:44:00.421390
CHECK DATE: 20050105  CHECK SEVERITY: LOW
CHECK PARM: STAR


ISGH0301E Global Resource Serialization is in RING mode.  Global Resource
          Serialization STAR mode was expected.

Explanation: The check found an unexpected mode when global resource serialization
          star mode was expected. Use star mode for best performance in a parallel sysplex.

System Action: The system might perform significantly worse than if it was in star mode.
```

```
Operator Response: Contact your system programmer.

System Programmer Response: See z/OS MVS Planning: Global Resource Serialization for
          more information on converting to global resource serialization star mode.

Problem Determination: N/A

Source: Global resource serialization

Reference Documentation: z/OS MVS Planning: Global Resource Serialization

Automation: N/A

Detecting Module: ISGHCGRS, ISGHCMSG

END TIME: 06/12/2007 18:44:02.003761  STATUS: EXCEPTION-LOW
CHECK(IBMXCF,XCF_SFM_ACTIVE)
START TIME: 06/07/2005 10:40:38.132396
CHECK DATE: 20050130  CHECK SEVERITY: MEDIUM
CHECK PARM: ACTIVE


* Medium Severity Exception *

IXCH0514E The state of Sysplex Failure Management is NOT consistent
with the IBMXCF recommendation.

  Explanation:  Sysplex Failure Management (SFM) is INACTIVE on this
    system. The IBMXCF specification requires that SFM be ACTIVE.

  System Action:  The system continues processing normally.

  Operator Response:  Report the problem to the system programmer.

  System Programmer Response:  Define an SFM policy with the
    administrative data utility IXCMIAPU.

    Start an SFM policy by issuing 'SETXCF
    START,POLICY,TYPE=SFM,POLNAME=xx' at the operating system console.

    Stop an active SFM policy by issuing 'SETXCF STOP,POLICY,TYPE=SFM'
    at the operating system console.

    IBM suggests that SFM should be ACTIVE.

  Problem Determination:  N/A

  Source:  Parallel Sysplex (XCF)

  Reference Documentation:
    z/OS MVS Setting Up a Sysplex

  Automation:  N/A
  Check Reason:  An SFM policy provides better failure management.

END TIME: 06/07/2005 10:40:39.091924  STATUS: EXCEPTION-MED
```

Exception message example 2 - An exception WTO message on the system console: The example below shows how the same check exception message WTO looks on the system console. Note that IBM Health Checker for z/OS issues an HZS message, HZS0002E, and then the check exception WTO appears as part of that message:

```
B7VBID47  HZS0002E CHECK(IBMXCF,XCF_SFM_ACTIVE):
IXCH0514E The state of Sysplex Failure Management is NOT consistent
with the IBMXCF recommendation.
```

Exception message example 3 - An exception WTO message in the system log: The example below shows the same check exception message WTO again, this time on the system console. Note that IBM Health Checker for z/OS issues an HZS message, HZS0002E, and then the check exception WTO appears as part of that message:

```
031 01000000  HZS0002E CHECK(IBMXCF,XCF_SFM_ACTIVE): 882
882 01000000  IXCH0514E The state of Sysplex Failure Management is NOT consistent
882 01000000  with the IBMXCF recommendation.
```

Evaluating check output and resolving exceptions

The best way to use IBM Health Checker for z/OS is to run it continuously on your system. But you must also evaluate check output, and resolve check exceptions. The check exceptions will give you both the reason for the exception and the steps to take to correct it. In the course of evaluating exceptions, you may need to review the exception with a number of different people in your installation with the expertise in the appropriate field. Resolving check exceptions will be an installation-specific process, and you'll need to develop efficient ways to respond. See also “Approaches to automation with IBM Health Checker for z/OS” on page 32.

Once you have evaluated a check exception, you can resolve it in one of the following ways:

- Update your system as suggested by the check exception message, which is the recommended approach. Then you will no longer receive the exception message when the check runs again. You can verify that you have resolved the exception by running the check again (R action character in SDSF or F *hzsproc*,RUN,CHECK=(*checkowner, checkname*) and then looking at the output in the message buffer. The check exception message will be gone from the output if you have resolved the exception.
- Evaluate the parameters specifying the value or values that the check is looking for. If a parameter is not appropriate for your system, update it so that you will no longer receive an inappropriate exception message when the check runs. You will also want to evaluate and possibly update the severity of the check to make sure it is appropriate for your installation. See Chapter 4, “Managing checks,” on page 43.
- Ensure that the check will not run and produce exceptions by either:
  - Putting the check into the Inactive state
  - Deleting the check

  See “Understanding check state and status” on page 34

It is very important that you resolve exception messages, so that when checks run at their specified intervals, they will report only exceptions that require attention. Otherwise, your IBM Health Checker for z/OS output may contain a mixture of messages that you regularly ignore and those reflecting a new potential problem. This might make it more likely that you could miss a key exception message.

Messages for individual checks will be documented in the component or product owning the message. For information about checks, including the name of the document where a check's messages are documented, see Chapter 13, “IBM Health Checker for z/OS checks,” on page 381.

Customizing check exceptions with dynamically varying severity

Some checks provide the capability of issuing check exception messages with a dynamically varying severity level, which gives you more control over how exception messages are issued and handled. For example, you might use the dynamic severity function for checks that inspect a system setting value and compare it against a threshold. As the value approaches the high threshold, the check can vary the severity of the exception, depending on how close to the threshold the value is. Some checks that support dynamic severity are:

- CHECK(IBMASM,ASM_PLPA_COMMON_SIZE)
- CHECK(IBMASM,ASM_LOCAL_SLOT_USAGE)
- CHECK(IBMASM,ASM_PLPA_COMMON_USAGE)
- CHECK(IBMVSM,VSM_CSA_THRESHOLD)
- CHECK(IBMVSM,VSM_SQA_THRESHOLD)

Example of using dynamic severity for a check: In this example, check VSM_CSA_THRESHOLD looks at the systems's level of common service area (CSA) storage level. If the ultimate emergency high level threshold for CSA usage is 95%, you might establish criteria for low, medium and high severity check exceptions based on CSA usage as follows:

- CSA usage at 60% is LOW severity
- CSA usage at 80% is MED severity
- CSA usage at 95% is HI severity

The advantage of staging the severity like this is that you get a little more flexibility and react-time than simply having one threshold established at CSA usage of 95%.

How does the check know what severity exception to issue when using dynamic severity? The check knows what severity exception to issue because you'll tell it ahead of time, using the check parameters. For example, using our example above, we might define the following check parameters for the VSM_CSA_THRESHOLD check:

- CSA_LOW(60%) - to send a low severity exception message for a CSA usage between 60% and 79%
- CSA_MED(80%) - to send a medium severity exception message for a CSA usage between 80% to 94%
- CSA_HIGH(95%) - to send a high severity exception message for CSA usage of 95% and above.

The check looks at the CSA usage on the system, and then uses the check parameters defined to determine what check severity to use when issuing an exception message.

It can be confusing to figure out whether a larger/higher parameter value corresponds to a higher severity or not. To make things even more confusing, checks such as the VSM_CSA_THRESHOLD check have it both ways! If you specify the parameter as a percent, then a bigger percentage corresponds to a higher severity. If you specify the parameter as a number, this applies to the amount of storage remaining and a lower number corresponds to a higher percentage. Read the parameter descriptions carefully.

For information about writing a check that exploits dynamic severity, see “Writing a check with dynamic severity levels” on page 124.

Approaches to automation with IBM Health Checker for z/OS

Why automate with IBM Health Checker for z/OS? Because even with all our planning and coding efforts, IBM Health Checker for z/OS is really only as good as the quality and speed of the installation's response to the check exceptions it finds. So what really matters is how quickly exception information gets routed to the right person to resolve the exception. In most cases, the person who manages IBM Health Checker for z/OS and sees check output first hand is not going to be the right person to resolve all the exceptions that pop up. And since checks are spread across components and products, you'll be routing the information to many different people (no one person can handle the whole variety of check exceptions effectively). In other words, you'll need an effective exception resolution process to go with IBM Health Checker for z/OS, and automation can be an integral part of that process.

There are numerous ways you can automate IBM Health Checker for z/OS and its exception messages, depending on the products installed in your shop and a million other variables. Here we'll describe our initial, simple approach to automating responses to check messages on our test systems. See also "More automation ideas."

Our approach to automation on a test sysplex:

1. **Automate HZSPRINT to keep a record of check messages on each system:** We use System Automation running under NetView® to automate HZSPRINT. We code the HZSPRINT JCL so that it automatically prints the messages from checks that found an exception. You can code the JCL for HZSPRINT so that it prints the message buffer to a sequential data set or simply to SYSOUT. Our JCL prints the message buffer data to a sequential data set for any check that finds an exception, as shown in the following example:

```
//HZSPRINT JOB 'ACCOUNTING INFORMATION','HZSPRINT JOB',
//         CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
//HZSPRINT EXEC PGM=HZSPRNT,TIME=1440,REGION=0M,
//    PARM=('CHECK(*,*)',
//    'EXCEPTIONS')
//SYSOUT   DD DSN=HCHECKER.PET.CHKEXCPT.SEQ.REPORT,DISP=MOD
```

2. **Automate HZSPRINT on each system to send e-mail messages:** You can add a step to the HZSPRINT JCL for each system that uses the Simple Mail Transfer Protocol (SMTP) FTP command to send e-mail messages. To do this, you must have SMTP set up - see *z/OS Communications Server: IP User's Guide and Commands*. We're using SMTP to send an e-mail alert whenever a check finds an exception. To do this, we key off of the HZS exception messages - see "Using HZS exception messages for automation" on page 34. This is only one simple approach to automating responses to check exceptions - see also "More automation ideas."

More automation ideas

There are many ways to use IBM and vendor products to automate responses to check output, including sending e-mail messages or setting off beepers. You've seen one approach we're using on a test sysplex. But there are a lot of ways to approach automation to help make sure you get the exception information to the people who can quickly resolve exceptions. Here are some automation ideas to kick around:

- **Key automation off check severity:** You can key your automation off check severity, tailoring the response to different severities. Because checks are classified as HIGH, MEDIUM, or LOW severity, you can tailor check response

based on the severity. For example, for HIGH severity check exceptions, you might want to set off a beeper call, while for LOW and MEDIUM severity check exceptions an email message would suffice.

Tailoring exception response depending on severity also means that each installation will need to evaluate the severity setting of each check, to see if that setting is appropriate for their environment. You can update the severity for a check using either the MODIFY command, HZSPRMxx parmlib member, or SDSF.

- **Route exception alerts to either a generic on-call address or a product expert:** When you set up your automation to make beeper calls or send emails, you can route the alerts either to a generic on-call address or to the expert for the specific product or component getting the check exception.
  - Routing to a generic on-call address makes automation setup faster, but could perhaps slow down response, since the person on call might have to re-route the information to an expert. To make responding easier, you can supply the person on call with a list of product / component experts. To make this approach work even better, you could create a run book for IBM Health Checker for z/OS, with the procedures for responding to check exceptions.
  - Routing alerts to specific product experts might make for faster responses to check exceptions, but could make the automaton set up more time consuming.

  Each installation will have to carefully calculate the trade-offs in this equation to make a decision about routing exception alerts.
- **Automate by message using MPF exits:** You can use an MPF installation exit to key off message identifiers and do message-specific processing. For example, using MPF exits you can:
  - Modify the presentation of messages, such as color and intensity.
  - Modify message routing, such as updating routing codes, changing the console that messages are routed to, or redirecting message traffic.
  - Suppress or automate message responses, such as filtering messages, performing error thresholding, or deleting messages.

  See IEAVMXIT -- Installation-Specified MPF Exits in *z/OS MVS Installation Exits*.
- **Automate on a check basis using routing codes:** You can update the routing codes assigned for all the messages for a particular check to to modify message routing. To update the routing codes, specify the ROUTCODE parameter on either the MODIFY *hzsproc* command or in the HZSPRMxx parmlib member. See Routing Codes in *z/OS MVS System Messages, Vol 1 (ABA-AOM)*.
- **Put check output in a central place for responders:** Whether they're routing check output to the lucky on-call person or a product / component expert, installations have to get the right information to the responder in order to take the appropriate action. Emailing the check output is problematic because the volume of check output can be very high. Instead, we're using HZSPRINT to write the data to a data set. That way we'll be able to email the name of the data set to the responder.
- **Keep it simple:** The goal of whatever automation method you pick is to get the right information to the right person so that the exception can be corrected as quickly as possible. To that end, keeping automation simple will make it easier to set up, maintain, and respond to exceptions quickly.

Using HZS exception messages for automation

A check exception message WTO consists of an HZS header message, followed by the check-specific exception message text, as shown in the system console example below:

```
HZS0001I CHECK(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)
BPXH033E MAXSOCKETS value for AF_INET is too low.
```

The HZS header messages issued with exceptions are:

- **HZS0001I**: Exception information message: low severity or WTOTYPE(INFORMATIONAL). Indicates that the check found a problem that will not impact the system immediately, but that should be investigated.
- **HZS0002E**: Exception eventual action message: medium severity or WTOTYPE(EVENTUAL). Indicates that the check found a medium severity problem in an installation.
- **HZS0003E**: Exception critical eventual action message: high severity or WTOTYPE(CRITICAL). Indicates that the check routine found a high-severity problem in an installation.
- **HZS0004I**: Exception hardcopy message: hardcopy only informational severity or WTOTYPE(HARDCOPY).

See *z/OS MVS System Messages, Vol 6 (GOS-IEA)* for a complete list of IBM Health Checker for z/OS HZS messages.

To find the documentation for the check-specific messages (which are on the second line of the WTO), such as BPXH033E shown above, look in either:

- Using LookAt - see "Finding check message documentation with LookAt" on page 41.
- The check owner product or element documentation. For a list of documents where you can find the messages documented for each check, see Chapter 13, "IBM Health Checker for z/OS checks," on page 381.

Understanding check state and status

Part of managing checks is understanding the check state and status shown for checks in the check messages in the message buffer, SDSF or the MODIFY*hzsproc*,DISPLAY output:

- **State:** Indicates whether a check will run at the next specified interval.
- **Status:** Describes the output of the check when it last ran.

For example:

- In the message buffer, system information displayed with the check message includes check status:

```
CHECK(IBMCNZ,CNZ_CONSOLE_MSCOPE_AND_ROUTCODE)
START TIME: 06/08/2005 09:49:17.410704
CHECK DATE: 20040816  CHECK SEVERITY: LOW

* Low Severity Exception *

CNZHF0003I One or more consoles are configured with a combination of
message scope and routing code values that are not reasonable.
                          .
                          .
                          .
 Check Reason:  Reduces the number of messages sent to a console in
   the sysplex
END TIME: 06/08/2005 09:49:17.451937  STATUS: EXCEPTION-LOW
```

- In SDSF, information displayed about checks includes state and status:

```
NAME                            CheckOwner      State              Status
CNZ_AMRF_EVENTUAL_ACTION_MSGS   IBMCNZ          ACTIVE(ENABLED)    SUCCESSFUL
CNZ_CONSOLE_MSCOPE_AND_ROUTCODE IBMCNZ          INACTIVE(ENABLED)  INACTIVE
CNZ_SYSCONS_ROUTCODE            IBMCNZ          ACTIVE(ENABLED)    EXCEPTION-LOW
GRS_CONVERT_RESERVES            IBMGRS          ACTIVE(DISABLED)   GLOBAL
```

- If you enter the `f hzsproc,display,checks` command to display check information, you'll receive output like the following. (Note that the check states are explained at the bottom of the output.)

```
HZS0200I 10.56.19 CHECK SUMMARY     134
CHECK OWNER      CHECK NAME                      STATE STATUS
IBMVSM           VSM_CSA_CHANGE                   AE   SUCCESSFUL
IBMRRS           RRS_RSTOFFLOADSIZE               AE   SUCCESSFUL
IBMRRS           RRS_DUROFFLOADSIZE               AE   SUCCESSFUL
IBMRRS           RRS_MUROFFLOADSIZE               AE   SUCCESSFUL
IBMRRS           RRS_RMDOFFLOADSIZE               AE   SUCCESSFUL
IBMRRS           RRS_RMDATALOGDUPLEXMODE          AE   SUCCESSFUL
IBMCNZ           CNZ_SYSCONS_PD_MODE              AE   SUCCESSFUL
IBMCNZ           CNZ_EMCS_INACTIVE_CONSOLES       ADG  SYS=J80
IBMCNZ           CNZ_SYSCONS_ROUTCODE             AE   EXCEPTION-LOW
IBMCNZ           CNZ_SYSCONS_MSCOPE               AE   SUCCESSFUL
IBMCNZ           CNZ_EMCS_HARDCOPY_MSCOPE         AE   EXCEPTION-MED
                                      .
                                      .
                                      .
  A - ACTIVE          I - INACTIVE
  E - ENABLED         D - DISABLED
  G - GLOBAL CHECK    + - ADDITIONAL WARNING MESSAGES ISSUED
```

Both of these examples show that the state and status for the highlighted check, CNZ_SYSCONS_ROUTECODE, are as follows:

- The check state is **AE** or **ACTIVE(ENABLED)**, which means that it will run at its next scheduled interval.
- The check status is **EXCEPTION-LOW**, indicating that the check found a low severity exception.

Check states: Each check state has two parts:

1. "User controlled states"
2. "IBM Health Checker for z/OS controlled states" on page 36

Check status: For check status, see "Check status" on page 37.

User controlled states

Table 3. User controlled states

Check state	Description
Active or **A**	An active check is one that has been added to IBM Health Checker for z/OS. An active check will run at whatever interval was specified for the check in the HZSADDCHECK exit routine or HZSPRMxx parmlib member, unless the system disables it. The life of an active check lasts until it gets refreshed or deleted. A check becomes active when: • It has been added to IBM Health Checker for z/OS in the active state. • You specify ACTIVATE or UPDATE ACTIVE on the HZSPRMxx parmlib member or the MODIFY command (F *hzsproc*).

Table 3. User controlled states (continued)

Check state	Description
Inactive or **I**	An inactive check is not eligible to run. The check becomes inactive when either: • It has been added to IBM Health Checker for z/OS in the inactive state. • You specify DEACTIVATE or UPDATE INACTIVE on the HZSPRMxx parmlib member for the MODIFY command (F *hzsproc*).

IBM Health Checker for z/OS controlled states

Table 4. States controlled by IBM Health Checker for z/OS

Check state	Description
Enabled or **E**	All checks are added to the system as enabled, and checks stay enabled unless IBM Health Checker for z/OS. An enabled check can be either active or inactive. An check will run if it is both enabled and active, or **eligible**.
Disabled or **D**	A disabled check is one that IBM Health Checker for z/OS has disabled because of check routine or environmental problems such as: • The check routine encounters multiple errors, such as 3 consecutive abends. • The Init function processing does not complete successfully. • The installation environment is not appropriate for the check. For example, the check might be looking for sysplex values when the installation is not a sysplex environment or the check may require UNIX System Services at a time when UNIX System Services is down. • The parameters passed to the check are not valid. • The check is a global one running on a different system. A disabled check is **not eligible** to run. You can get IBM Health Checker for z/OS to enable your check by fixing the error causing the check to be disabled and then **refreshing** the check. If you update the parameters passed to a check, you do not need to refresh the check, because the system will re-enable the check automatically in order to let it see if the parameters are now correct. Some conditions causing a disabled check may resolve themselves. For example, if a check is disabled because it is a global check that is already running on a system in the sysplex, it will show up as disabled on other systems. Then, when it is no longer running on the original system, the system will enable the check on another system in the sysplex. Or, if a check requires UNIX System Services to run, but UNIX System Services is down, that check will be disabled until UNIX System Services comes up again. At that point, the system will enable the check.
Global or **G**	A global check is one which runs on one system but reports on sysplex-wide values and practices. A global check shows up as disabled for all systems in the sysplex, except for the one where it is actually running.

ACTIVE(DISABLED) and INACTIVE(ENABLED) - understanding check state combinations

Checks have a two part state, which can sometimes seem contradictory. Basically, however, it all boils down to whether a check is eligible to run or not. If a check is **eligible**, it is both active and enabled, and running at its established interval. An **ineligible** check will not run because it was either:

- Disabled by IBM Health Checker for z/OS because of errors or environmental problems
- Deactivated by a user
- Both disabled **and** deactivated

Table 5. Check state combinations

Eligibility	State
Eligible states	• **ACTIVE(ENABLED)** or **AE**: Check is ready and able to run.
Ineligible states	• **ACTIVE(DISABLED)** or **AD**: Check has been defined to IBM Health Checker for z/OS and was running, but IBM Health Checker for z/OS found errors and disabled the check (see "IBM Health Checker for z/OS controlled states" on page 36). The check will not run. • **INACTIVE(ENABLED)** or **IE**: A user has deactivated the check (see "User controlled states" on page 35). From IBM Health Checker for z/OS's point of view, this check is in good standing and can run whenever the user re-activates it. However, the check will not run. • **INACTIVE(DISABLED)** or **ID**: The system disabled the check because of system or environment errors and a user deactivated it (see "IBM Health Checker for z/OS controlled states" on page 36). The check will not run.

Check status

There are many status values possible for a check, as shown in display output or the message buffer. See the status field in message HZS0200I in HZS messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*

Using the HZSPRINT utility

The HZSPRINT utility allows you to look at check output. HZSPRINT writes the message buffer for the target checks to SYSOUT for one check, multiple checks, or all checks.

The following information assumes that you have already set up security for HZSPRINT - see "Setting up security for the HZSPRINT utility" on page 16.

The SYS1.SAMPLIB JCL for the HZSPRINT utility is as follows:

```
//HZSPRINT JOB
//*...                                                          */
//HZSPRINT EXEC PGM=HZSPRNT,TIME=1440,REGION=0M,PARMDD=SYSIN
//SYSIN DD *,DLM='@@'
CHECK(*,*)
,EXCEPTIONS
@@
//SYSOUT   DD SYSOUT=A,DCB=(LRECL=256)
```

HZSPRINT parameters can be passed:

- via the JCL PARM string which is limited to 100 characters, or
- via a JCL PARMDD, which is limited to 256 "effective" characters, for HZSPRINT, at this time. Trailing blanks per line do not count though. Do not include any other extra blanks, in particular at the beginning of any line.

Parameters should be separated from each other by a comma. The following parameters are supported by HZSPRINT:

CHECK(*check_owner*,*check_name*)

check_owner must be between 1-16 characters and *check_name* must be between 1-32 characters. To find the check owner and check name, use either the SDSF CK option or use the following MODIFY command:

```
F hzsproc,DISPLAY,CHECKS
```

You can also use wildcard characters '*' and '?' in both the check owner and check name fields to get output from multiple checks. For example, to see the output of all the checks on the system, you could use the following:

```
//      PARM='CHECK(*,*)'
```

An asterisk (*) represents any string having a length of zero or more characters. A question mark (?) represents a position which contains any single character. The system converts any lowercase letters to uppercase.

CHECK(*,*) is the default setting for HZSPRINT. If you do not specify CHECK, you will get CHECK(*,*) to see the output of all checks. Note that using CHECK(*,*) will only work if you have access to all the checks. See "Setting up security for the HZSPRINT utility" on page 16.

EXCEPTIONS

Optional parameter EXCEPTIONS lets you limit the output in SYSOUT to messages from checks that wrote at least one check exception message. For example, to see the output of all checks that found exceptions, use the following:

```
//      PARM='CHECK(*,*),EXCEPTIONS'
```

Deleted checks will not be reported on.

LOGSTREAM(*log_stream_name*)

Optional parameter LOGSTREAM specifies that you want to print the specified log stream, instead of querying the current Health Checker instance. The *log_stream_name* has to start with HZS.

SYSNAME(*system_name*)

Optional parameter SYSNAME lets you limit the output in SYSOUT to output from checks running on the specified system, *sysname*. You can specify the SYSNAME parameter only with LOGSTREAM.

You can use wildcard characters '*' and '?' in the *system_name* field to specify that you want check output from multiple systems.

The default for SYSNAME is SYSNAME(*), which will give you output for specified checks from all the systems in the sysplex.

TIMERANGE(*12-char-start*,*12-char-stop*)

Optional parameter TIMERANGE lets you limit the data in SYSOUT to entries in the specified time range. Data will only reported for check iterations with a start time within the TIMERANGE. Specifying TIMERANGE will let HZSPRINT look beyond the most current iteration of a check as far as data for previous check iterations is available. While this is the default behavior for when LOGSTREAM is specified, without TIMERANGE and without LOGSTREAM, only the most current check iteration will be reported on for a check. Specify the 12-char-start and 12-char-stop as YYYYMMDDHHMM. All 12 characters must be valid decimal digits and must represent a valid year, month (01-12), day (01-31 depending on the month), hour (00-23), and minute (00-59) specification.

When TIMERANGE is specified with a LOGSTREAM, the time range applies to entries in the logstream.

When TIMERANGE is specified without a LOGSTREAM, it applies to entries available in the currently running instance of IBM Health Checker for z/OS.

If you want to allocate a data set for HZSPRINT output:

- The data set must be:
  - Fixed length, blocked records. For example, RECFM=FBA or RECFM=FBM
  - Logical record length of 256
- Add the name of the output data set allocated above to the HZSPRINT JCL. For example:

  ```
  //SYSOUT   DD DISP=SHR,DSNAME=D10.HCHECKER.REPORT.FEB2505,DCB=(LRECL=256)
  ```
- Note that the first character of each line of HZSPRINT output is a carriage control character.

Example of HZSPRINT output

The following shows a portion of the HZSPRINT output for a request that includes output for all checks with exceptions:

```
********************************************************************************
*                                                                              *
* HZSU001I IBM Health Checker for z/OS Check Messages                          *
* Filter: CHECK(*,*)                                                           *
* Filter: Only checks with exception(s)                                        *
*                                                                              *
********************************************************************************


********************************************************************************
*                                                                              *
* Start: CHECK(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)                              *
*                                                                              *
********************************************************************************

CHECK(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)
START TIME: 03/30/2005 11:31:06.593289
CHECK DATE: 20040808  CHECK SEVERITY: LOW
CHECK PARM: 64000,64000

BPXH003I z/OS UNIX System Services is configured using OMVS=(00) which
correspond to the BPXPRMxx suffixes. The IBMUSS specification for IBM
Health Checker for z/OS USS_MAXSOCKETS_MAXFILEPROC is 64000,64000.
                     .
                     .
                     .
END TIME: 03/30/2005 11:31:08.457023  STATUS: EXCEPTION-LOW

********************************************************************************
*                                                                              *
* End:   CHECK(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)                              *
*                                                                              *
********************************************************************************
```

HZSPRINT utility completion codes

The following list shows the completion codes returned by the HZSPRINT utility in system message IEF142I:

Completion code
Description

0 Success

400
No matches - the HZSPRINT utility could not match your request with checks.

401
HZSPRINT could not retrieve all messages from requested checks.

402
Some records were missing.

403
HZSPRINT could not write all the message buffers.

404
The log stream specified was empty.

801
An unknown keyword was encountered in the HZSPRINT parameter string.

802
Incorrect check owner specified. *checkowner* must be between 1-16 characters.

803
Incorrect check name specified. *checkname* must be between 1-32 characters.

804
Comma missing between *checkowner* and *checkname* in the HZSPRINT JCL.

805
Missing the closing parenthesis in `CHECK(checkowner,checkname)`.

806
Incorrect log stream name specified.

811
No log stream was specified in the JCL.

812
The log stream specified was incorrect.

813
The check specified was bad.

814
No system name was specified in the SYSNAME parameter.

815
The system name specified in the SYSNAME parameter was incorrect.

816
The SYSNAME parameter is not allowed as specified - you can only specify SYSNAME with the LOGSTREAM parameter.

817
The syntax of the TIMERANGE value is invalid.

818
The TIMERANGE value is empty.

899
The HZSPRINT parameter string is too long.

1200
The HZSPRINT utility was not authorized to retrieve the requested

information. For example, the XFACILIT class may not have been RACLISTed. Check the security definitions described in “Setting up security for the HZSPRINT utility” on page 16.

1201
The system could not open the specified SYSOUT data set. Check the SYSOUT data set requirements in “Optionally set up the HZSPRINT utility” on page 12.

1202
Unexpected logical record length on the specified SYSOUT data set.

1203
IBM Health Checker for z/OS is not active.

1204
HZSPRINT encountered an error with the log stream.

1205
The SYSOUT data set specified is not allocated.

1206
The specified SYSOUT data set is partitioned.

1601 -1604
Internal error. Contact the IBM Support Center.

Finding check message documentation with LookAt

LookAt is being sunset with the announcement of the new version of z/OS (V2R1). Knowledge Centers will gradually take over the function of all search in technical documentation.

You can continue to access older releases (prior to V2R1) of messages here as-is. Neither the site nor the content will continue to be updated.

To find check message documentation on messages prior to V2R1, use component message documents or use message explanations directly from the **LookAt** Web site at http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/. Because checks, along with their output messages, might be added by PTFs between releases of component message documents, LookAt will contain the most up to date message information. The check message ID is on the second line of the WTO for a check message, as shown for check message BPXH033E below:

```
HZS0001I CHECK(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)
BPXH033E MAXSOCKETS value for AF_INET is too low.
```

From the LookAt Web site, specify the check message ID and select the appropriate z/OS release. (Only releases z/OS V1R7 and higher will contain check message documentation.)

The example below shows how we have selected z/OS V1R7 to search for SDUMP check exception message IEAH701I. LookAt will take directly into the message documentation for IEAH701I.

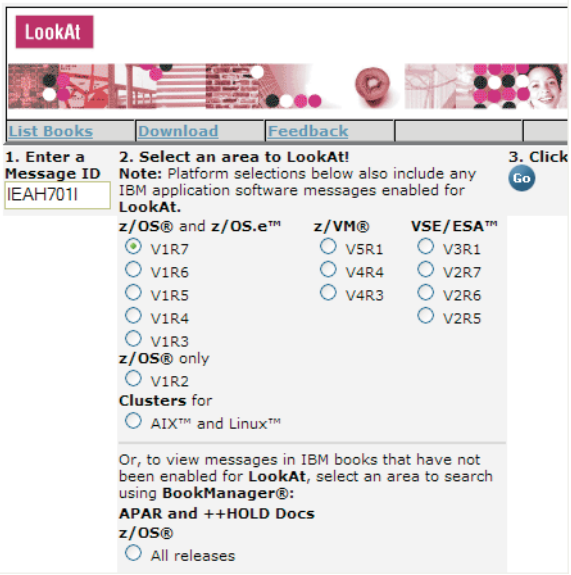


Figure 2. Using LookAt to find check message documentation

If you don't find a particular message in a z/OS release, choose the button on the bottom to search in APARs and ++HOLDs for all releases. You may find the message there because some checks will be released APARs between releases.

Chapter 4. Managing checks

Managing checks includes tasks such as:

- Updating or overriding values defined for checks or check output, such as check interval, check severity, or check message routing code or WTO type
- Making checks active or inactive
- Requesting that the system process HZSPRMxx parmlib members
- Adding checks
- Deleting checks
- Refreshing checks (deleting then adding) checks
- Displaying check information

You can manage checks with the following interfaces:

- **Make dynamic, temporary changes** to checks such as deactivating, adding, running, or temporarily updating check values, using:

  SDSF. See "Using SDSF to manage checks" on page 45.

  MODIFY command. See "Making dynamic, temporary changes to checks" on page 44.
- **Make persistent changes** to checks that persist across check refreshes and restart of IBM Health Checker for z/OS using policies. You can define policies by specifying policy statements to be in your **HZSPRMxx** parmlib member or members, specifying the parmlib member is in the list of parmlib members being used at the start IBM Health Checker for z/OS, and activating the policy. See "Making persistent changes to checks" on page 52.

Table 6. When do I use which interface to manage checks?

How long will the change be in effect?	Task	Recommended interface
Dynamic, temporary changes See "Making dynamic, temporary changes to checks."	**I want to look at check output**	**SDSF or HZSPRINT**
	I want to issue one-time actions against checks, including: • Adding, and deleting checks • Refreshing checks (deleting one or more checks, then adding all eligible checks) • Displaying checks • Running checks	**SDSF or MODIFY** *hzsproc*
	I want to experiment with temporary updates to check values, such as: • Interval • Severity • Category • Check message routing codes These changes will last until the check is refreshed (deleted and then add all eligible checks).	**SDSF or MODIFY *hzsproc*,UPDATE**
Persistent changes See "Making persistent changes to checks" on page 52.	**I want to make changes that will persist across IBM Health Checker for z/OS restarts**, such as permanent updates to check values on policy statements, adding local checks from the HZSPRMxx parmlib member, or turning on log stream support for IBM Health Checker for z/OS	**HZSPRMxx statements**, and then make sure that parmlib member(s) are in the list of parmlib members to be applied at IBM Health Checker for z/OS restart.

Making dynamic, temporary changes to checks

If you want to make dynamic, temporary check updates, use either:

- SDSF- see "Using SDSF to manage checks" on page 45.
- The MODIFY command - See "Cheat sheet: examples of MODIFY ***hzsproc*** commands" on page 47.

You can :

- Take one time only actions against checks, such as:
  - Adding and deleting checks
  - Refreshing checks (deleting one or more checks and then adding all eligible checks)
  - Displaying checks and check history
  - Running checks
- Update check values with changes that last until the next refresh (add) of the check or checks, including:
  - Activating and deactivating checks
  - Updating check values. For example, using SDSF and the MODIFY commands, you can update check values, such as interval, severity, category, or check message routing codes.

  Check values changed this way will last just until the check is refreshed (deleted and added again). The system will **not** apply the changed values to any new checks that you add later. SDSF or MODIFY commands are great for testing check value updates, but to make permanent changes you should create a policy

statement to apply the changes to all refreshed and added checks and persist across IBM Health Checker for z/OS restarts. See “Creating and maintaining IBM Health Checker for z/OS policies” on page 53.

Using SDSF to manage checks

For IBM Health Checker for z/OS, SDSF provides the CK command to display and manage checks. Using CK, you can issue one-time or temporary actions against active checks, including:

- Displaying check information - action character D
- Displaying check output - action character S

  Shows check status and the message buffer for the check iteration.
- Displaying check history in the IBM Health Checker for z/OS log stream- action character L

  The check history panel (CKH) displays iterations of the check run during the lifetime of the IBM Health Checker for z/OS address space. Checks that ran before the last IBM Health Checker for z/OS restart are not accessible to SDSF and are not displayed, even if they are resident in the log stream.

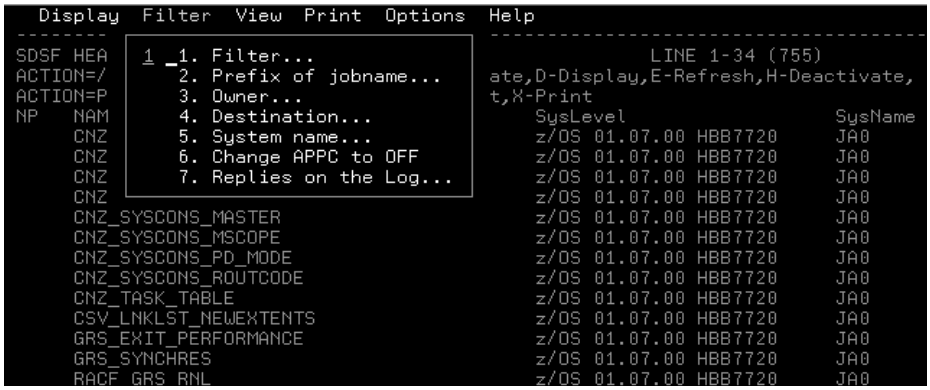
  This panel is only applicable if you have a log stream defined and setup for IBM Health Checker for z/OS.
- Refreshing checks (deleting one or more checks, then adding all eligible checks) - action character E
- Deleting checks - action character P
- Running checks - action character R
- Making temporary updates to check values in use, such as check interval, category, severity, or check message routing code. These updates will last until the check is refreshed (deleted and added again).- action character U
- Deactivating checks - action character H
- Changing check states

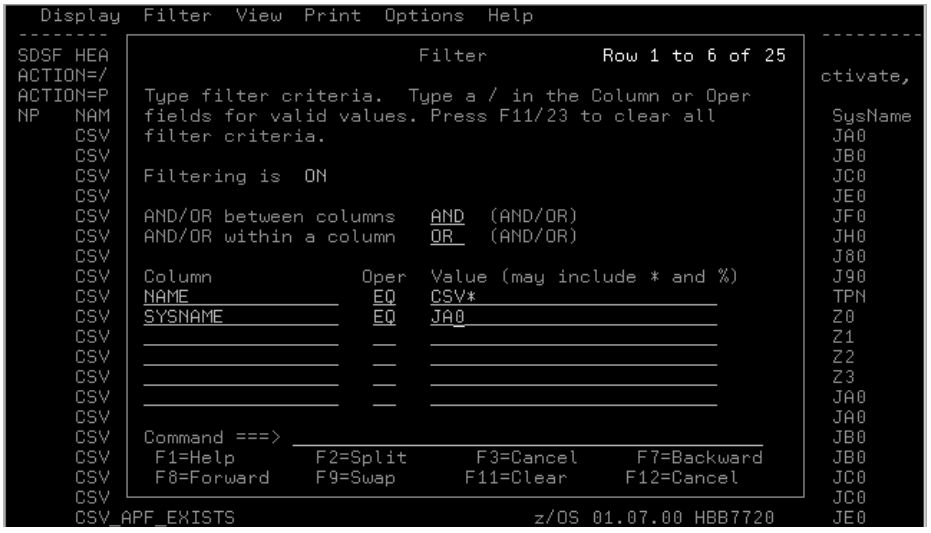
Like all SDSF primary displays CK can also be accessed from a pull-down when SDSF is running as an ISPF dialog. To display the action characters for the CK panel, use the **SET ACTION SHORT** or **SET ACTION LONG** command.

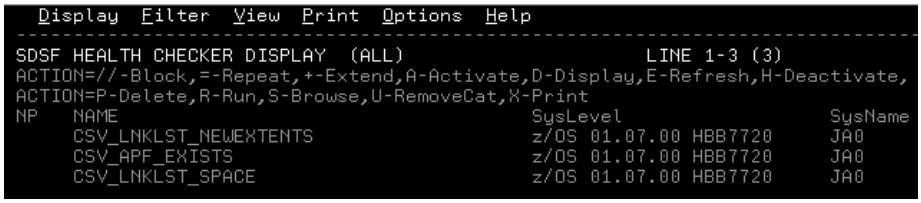
You can get sysplex-wide information about checks using SDSF's server and WebSphere MQ. Starting with V1R13, SDSF uses XCF instead of Websphere MQ (if all systems are at V1R13 or higher) to display sysplex-wide information and no configuration is required.

You can also do the following with CK:

- **See just exceptions**, using the CK E command instead of CK.
- **Browse** a check using the S action character: When you are running SDSF under ISPF, you can also use the SB or SE action characters to browse the output with ISPF browse or edit.
- **Limit** the checks shown with the S command. For example, S ABC* would show all checks that start with ABC. Reset the checks shown by typing S without parameters.
- **Filter** checks shown using the filter option on the left hand side at the top of the screen. In this example, we filter for checks with names starting with CSV on system JA0:







You can turn filtering off by using the FILTER OFF command on the command line.

- **Sort** the checks. For example, you can sort checks reporting exceptions in descending order using SDSF command `SORT RESULT D`.
- **Display** checks using the D or DL action characters.

For complete information on the SDSF CK and CKH panels, see the following:

- To setup security, see Checks on the CK and CKH panel in *z/OS SDSF Operation and Customization*.
- **SDSF online help** for information about the columns and functions, such as action characters, overtypeable columns, and commands.
- To customize columns on the CK and CKH panels in ISFPARMS and review customized field lists for the CK panel for changes related to the new LogStream column, see Variable field lists (ISFFLD or FLD) in *z/OS SDSF Operation and Customization*.

Managing checks with the MODIFY *hzsproc* command

MODIFY (F *hzsproc*) commands are useful for making dynamic, temporary changes to checks. See "Using HZSPRMxx and MODIFY *hzsproc* command" on page 65 for complete syntax information. In this section, we'll cover the following:

- "Cheat sheet: examples of MODIFY ***hzsproc*** commands"
- "Why you should not add checks using the MODIFY *hzsproc* command" on page 48
- "Why does my check reappear after I delete it? Understanding delete processing" on page 48
- "But my check doesn't reappear after ADDNEW - what happened to it?" on page 49
- "How can I delete checks while IBM Health Checker for z/OS is terminating?" on page 50
- "Using the category filter to manage checks" on page 50

Cheat sheet: examples of MODIFY *hzsproc* commands

The following examples of MODIFY (F *hzsproc*) commands are useful for making dynamic, temporary changes to checks. See "Using HZSPRMxx and MODIFY *hzsproc* command" on page 65 for complete syntax information.

Table 7. F hzsproc command examples

Action	Command example
Run checks	Run all checks that have an owner that is 6 characters long beginning with IBM: `F hzsproc,RUN,CHECK=(ibm???,*)` This is a one time action issued against the checks involved.
Activate checks	Activate checks that belong to any of the categories A or B: `F hzsproc,ACTIVATE,CHECK=(*,*),CATEGORY=(ANY,A,B)` See "Using the category filter to manage checks" on page 50. This is a one time action issued against the checks involved.
Deactivate checks	Deactivate checks that belong both to categories B and C: `F hzsproc,DEACTIVATE,CHECK=(*,*),CATEGORY=(ALL,B,C)` This is a one time action issued against the checks involved.
Disable checks	You cannot disable a check, the system will disable a check in response to check routine or environmental problems. See "IBM Health Checker for z/OS controlled states" on page 36.
Enable checks	You cannot enable a check, the system enables a check after you solve whatever problem led the system to disable it in the first place. See "IBM Health Checker for z/OS controlled states" on page 36.
Delete a check	Delete a check: `F hzsproc,DELETE,CHECK=(IBMRACF,RACF_GRS_RNL)` This is a one time action issued against the check or checks involved. When you delete a check using the MODIFY command, your check will come back to run again whenever ADDNEW processing occurs. (ADDNEW processing refreshes **all** checks.) If you want a check to be deleted and stay deleted, use the DELETE parameter on a policy statement. See "Why does my check reappear after I delete it? Understanding delete processing" on page 48.
Refresh checks	Refresh (delete one or more checks and add all eligible checks): `F hzsproc,REFRESH,CHECK=(*,*)` This command deletes specific checks and then adds ALL checks eligible to run.

Table 7. F hzsproc command examples (continued)

Action	Command example
Undelete a check	Undelete a check: `F hzsproc,ADDNEW` This is a one time action issued against all checks that are eligible to run.

Update a check	• Update a check to high severity:
	`F hzsproc,UPDATE,CHECK=(IBMRACF,RACF_GRS_RNL),SEVERITY=HIGH`
	• Update all checks with a check owner that starts with "a" and a name that starts with "b" to have:
	– WTOTYPE of informational
	– INTERVAL of one hour
	`F hzsproc,UPDATE,CHECK=(a*,b*),WTOTYPE=INFORMATIONAL,INTERVAL=01:00`
	These updates lasts until the check involved is refreshed.

Clearing a check parameter error	There are lots of checks that do not accept parameters (see Chapter 13, "IBM Health Checker for z/OS checks," on page 381). If you do have a check that you have defined with parameters when it does not accept parameters, you can clear the parameter error by issuing the following command to update the check with a null parameter string: `F hzsproc,UPDATE,CHECK=(checkowner,checkname),PARM()`

Add HZSPRMxx parmlib members	• Add HZSPRMxx parmlib members to the list of members that IBM Health Checker for z/OS is using:
	`F hzsproc,ADD,PARMLIB=(suffix1,suffix2,...suffixn)`
	• Replace the list of parmlib members that IBM Health Checker for z/OS is using:
	`F hzsproc,REPLACE,PARMLIB=(suffix1,suffix2,...suffixn)`
	
	REPLACE,PARMLIB does the following:
	– Sets the list of HZSPRMxx parmlib member suffixes to the list specified on the REPLACE parameter.
	– Wipes out any existing policy statements.
	– Processes the statements in the parmlib members in the list, applying them to existing checks.
	– Processes the policy statements and applies the statements to new checks.

Activate a policy or **Switch between policies**	Activate an IBM Health Checker for z/OS: `F hzsproc,ACTIVATE,POLICY=policyname`

Why you should not add checks using the MODIFY *hzsproc* command

We recommend against adding checks using the MODIFY command because the system will not remember changes you made using MODIFY when IBM Health Checker for z/OS is restarted. In addition, the MODIFY command for your entire check definition is limited to 126 characters. Use HZSPRMxx instead, see "Using HZSPRMxx and MODIFY *hzsproc* command" on page 65.

Why does my check reappear after I delete it? Understanding delete processing

The `F hzsproc,DELETE` command is a onetime action issued against a check. That means that if you issue the `F hzsproc,DELETE` command to delete a check, it will probably reappear to run the very next time something kicks off ADDNEW processing. No matter how it's kicked off, ADDNEW processing tries to refresh **all** checks, bringing any temporarily deleted check back in the process. In this section,

we'll explain a bit about how delete processing works. But the bottom line is this: If you really want to delete a check permanently, do it in a policy statement.

We'll use a scenario to explain why your check keeps coming back. But first, you'll need to understand that all the relevant facts about a check routine are contained in a check definition contained in either:

- An HZSADDCHECK exit routine
- An HZSPRMxx parmlib member, created with the ADD | ADDREPLACE CHECK command

When a command or other request kicks off ADDNEW processing, the system adds or reactivates the check as defined in the check definition.

1. Okay, let's say that:
   - CHECK(A,B) is added to the system by HZSADDCHECK exit routine AEXIT.
   - CHECK(C,D) is added to the system in the HZSPRMxx parmlib member with the ADD | ADDREPLACE CHECK command.
2. Now, let's say that someone issues a MODIFY command or non-policy parmlib statement that deletes CHECK(A,B). When delete processing completes:
   - CHECK(A,B) is in the deleted status
   - CHECK(C,D) is eligible to run
3. Now, something kicks off ADDNEW processing, such as a request to refresh CHECK(C,D). A refresh request consists of a delete of the check, followed by an ADDNEW request.
4. The ADDNEW command reactivates CHECK(C,D) as defined in the HZSPRMxx member. But ADDNEW processing **also** runs the AEXIT HZSADDCHECK exit routine, and AEXIT adds CHECK(A,B) back to the system, or undeletes it. Deleted CHECK(A,B) is back! ADDNEW processing kicked off for one check reactivates all checks as defined in check definitions in either HZSADDCHECK exit routine's or HZSPRMxx parmlib members.

So, if you really want to delete a check permanently, use a policy statement in an HZSPRMxx member, such as:

```
ADDREPLACE POLICY STMT(DEL1) DELETE Check(A,B)
```

Then issue `F hzsproc,ADD,PARMLIB=xx` to add the HZSPRMxx member containing the new policy statement to the list of members containing the IBM Health Checker for z/OS policy.

Now, when something kicks off ADDNEW processing, the system will reactivate all the undeleted check definitions, bringing back CHECK(C,D) but not CHECK(A,B).

Note that ADDNEW processing is staged, so that the system will first process all check definitions to add all the checks, bringing back CHECK(A,B). Then however, the system also applies the policy statements, including the statement that deletes CHECK(A,B). In the end, CHECK(A,B) stays deleted when you put the delete in the policy.

But my check doesn't reappear after ADDNEW - what happened to it?

When ADDNEW processing is kicked off, all checks added by either the HZSADDCHECK exit routines or in the HZSPRMxx parmlib member are candidates for being refreshed as part of the ADDNEW processing. Candidates for

refresh are checks that are not deleted by policy statements and that do not already exist. If ADDNEW does not bring back your check from deletion, the problem is probably one of the following:

- You have a policy statement in your policy that deletes that check.
- The exit routine that added the check the last time has been updated and no longer adds your check.
- The exit routine that added the check the last time has been removed from the HZSADDCHECK exit.

Why can't I re-add my HZSPRMxx parmlib defined check after I delete it? More understanding of the delete processing...

Here is a possible common mistake: Lets say that you defined a System REXX check in the HZSPRMxx parmlib member using the F *hzsproc*,ADD | ADDREPLACE,CHECK command. Then, you deleted it using the DELETE command. But now you want to bring it back again, so you issue the ADD,CHECK command. But the command fails, with a message telling you the check already exists, even though it will not appear in SDSF or display output. That is because you deleted the check, but the **check definition** is still lurking there in the HZSPRMxx parmlib member, and is still loaded in the system. What you need to do to get your check to run again is to put an ADDREPLACE,CHECK statement containing the check definition into a parmlib member, and issue the F *hzsproc*,ADD,PARMLIB,CHECKS. Your check will now be ready to run.

How can I delete checks while IBM Health Checker for z/OS is terminating?

While IBM Health Checker for z/OS is in the process of terminating, you may get a message that the system is waiting for checks to complete before termination itself can complete:

```
HZS0020E WAITING FOR CHECKS TO COMPLETE
```

The wait might be longer if you have System REXX checks running on the system. But if you try to speed up the process of IBM Health Checker for z/OS termination by deleting checks using the `F hzsproc,DELETE` command, you will find that neither that command nor most other `F hzsproc` commands work during the termination process.

However, you **can** use the following command to delete all the checks during termination of IBM Health Checker for z/OS:

```
F hzsproc,DELETE,CHECK=(*,*),FORCE=YES
```

Make sure that the FORCE=YES option is what you want:

- FORCE=YES issued against a remote check will result in a non-retriable abend.
- FORCE=YES will delete checks that are still in the process of running.

The only other `F hzsproc` command that will work during the termination process is the `F hzsproc,DISPLAY,CHECKS` command.

Using the category filter to manage checks

When you have many checks, you can use categories to make it easier to manage or display information.

- Use the ADDCAT, REPCAT, and REMCAT parameters:
  - ADDCAT lets you add the specified check to a category
  - REPCAT lets you replace a category for a check
  - REMCAT lets you remove a check from a category

- Use the CATEGORY filter to filter actions against checks by category. CATEGORY accepts up to 16 named categories, each represented by a 1-16 character string.

For example, you might put checks into categories such as *shift* and *offshift*, *global*, or *exception*. Then you can perform actions such as activate, deactivate or run a group of checks with one command. All categories are user-defined; IBM does not define any categories for checks.

The following examples shows how you can use categories in the HZSPRMxx member and in the MODIFY command to manage checks:

1. First, I add a number of checks to a new category, DEBUG in the HZSPRMxx.

   ```
   /* add some checks to category debug */
   ADD POLICY STMT(POL1) UPDATE CHECK(IBMUSS,USS_FILESYS_CONFIG)
       ADDCAT(DEBUGCAT)
   ADD POLICY STMT(POL2) UPDATE CHECK(IBMCNZ,CNZ_TASK_TABLE)
       ADDCAT(DEBUGCAT)
             .
             .
             .
   ADD POLICY STMT(POL17) UPDATE CHECK(IBMGRS,*)
       ADDCAT(DEBUGCAT)
   ```

   This takes some time, but it will save time in step 2.
2. Now I want to put all the checks from category DEBUG into debug mode. I only want to do this temporarily, so I do this with a MODIFY command. I can do this without specifying all those long check names by using the category filter:

   ```
   F hzsproc,UPDATE,CHECKS=(*,*),CATEGORY=(DEBUGCAT),DEBUG=ON
   ```

But there's more you can do with the CATEGORY filter! The syntax of the filter is:

```
CATEGORY=([{ANY|EVERY|EXCEPT|ONLY},][category1[,...,categoryn]])
```

I can assign checks to multiple categories, and sort them out on the CATEGORY filter using ONLY, ANY, EVERY, and EXCEPT:

ANY
Checks that are in any of the specified categories

EVERY
Checks that are in every specified category

EXCEPT
Checks that are not in any of the specified categories

ONLY
Checks that are in every one of the specified categories and that have only as many categories as are specified. For example, a check assigned to three categories would not match if the CATEGORY=ONLY statement on this MODIFY command specified two categories.

ONLY is the default, but for the sake of clarity, we recommend that you specify the category option that you want.

For example, in the following scenario, I have checks ONE, TWO, THREE, FOUR, and FIVE in the following categories:

Category	SHIFT1	SHIFT2	IMPORTANT	RACF	GRS	CONSOLES
Checks	ONE THREE FOUR FIVE	TWO	ONE TWO FOUR	ONE TWO	THREE	FOUR FIVE

So if I create a policy statement in my HZSPRMxx member to change existing and future checks to LOW severity in every category except category IMPORTANT:

```
ADD POLICY STMT(LOWCAT) UPDATE CHECK(*,*)
     CATEGORY(EXCEPT,IMPORTANT)
     SEVERITY(LOW)
```

This will affect only checks that are not in the IMPORTANT category, which will be checks THREE and FIVE.

Using these categories and checks, the following table shows how a bunch of category filters map to checks affected in our scenario:

CATEGORY filter	checks affected
CATEGORY=(ANY,SHIFT1)	ONE, THREE, FOUR, FIVE
CATEGORY=(ANY,IMPORTANT)	ONE, TWO, FOUR
CATEGORY=(ANY,SHIFT1,SHIFT2)	ONE, TWO, THREE, FOUR, FIVE
CATEGORY=(EVERY,SHIFT1,CONSOLES)	FOUR, FIVE
CATEGORY=(EVERY,SHIFT1,IMPORTANT,CONSOLES)	FOUR
CATEGORY=(EXCEPT,IMPORTANT)	THREE, FIVE
CATEGORY=(ONLY,SHIFT1)	None
CATEGORY=(ONLY,SHIFT1,CONSOLES)	FIVE

Making persistent changes to checks

You can make changes to checks that persist across check refreshes and restart of IBM Health Checker for z/OS using statements in the HZSPRMxx parmlib member. The HZSPRMxx parmlib member should include only the following kinds of changes:

- Defining policies by specifying policy statements in your HZSPRMxx parmlib member or members, specifying the parmlib member is in the list of parmlib members being used at the start IBM Health Checker for z/OS, and activating the policy. See "Creating and maintaining IBM Health Checker for z/OS policies" on page 53.
- Defining local installation-written check defaults to the system and adding them to IBM Health Checker for z/OS using the ADD | ADDREPLACE CHECK statement in an HZSPRMxx parmlib member. See "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.
- Turning log stream support for IBM Health Checker for z/OS on or off using the LOGGER parameter in an HXSPRMxx parmlib member. See "LOGGER" on page 74.
- Defining the name of the HZSPDATA persistent data set using the HZSPDATA parameter in an HZSPRMxx parmlib member. See "HZSPDATA" on page 73.

Note that you can share HZSPRMxx parmlib members across different systems in a sysplex, even if those systems are at different levels, and even if a system at a

lower level does not support the most current HZSPRMxx parameters you specify. See "Sharing critical IBM Health Checker for z/OS information between systems at different levels" on page 8.

Creating and maintaining IBM Health Checker for z/OS policies

An IBM Health Checker for z/OS **policy** lets you manage checks by applying persistent changes to checks. A policy is the place to put any check changes you want to make persistent and to have applied to checks you add in the future. Starting with z/OS V1R8, you can create multiple policies and switch between them. (Systems at the z/OS V1R4 through R7 with IBM Health Checker for z/OS support installed can have only one policy per system.)

An IBM Health Checker for z/OS **policy** simply consists of a set of policy statements in an HZSPRMxx member or members currently in use for a system. The system applies the information in your active IBM Health Checker for z/OS policy to all existing checks and to any new checks you add. IBM Health Checker for z/OS processes information from the active policy every time checks are added or refreshed, every time you activate a new policy, and whenever you restart IBM Health Checker for z/OS.

When we use the term IBM Health Checker for z/OS restart, we mean either:

- Restarting IBM Health Checker for z/OS after it terminates
- Starting of IBM Health Checker for z/OS on a subsequent IPL

To ensure that a policy is remembered and applied at IBM Health Checker for z/OS restarts, specify the HZSPRMxx members containing the policy in the IBM Health Checker for z/OS procedure, *hzsproc* and use the following command to activate the policy you wish to make the **current** policy:

```
F hzsproc,ACTIVATE,POLICY=policyname
```

If you have multiple policies, you can switch between them using the same `F hzsproc,ACTIVATE,POLICY=policyname` command.

On each policy statement, you update check values for a check or set of checks, specifying updates that you wish to apply permanently. You can also use policy statements to permanently delete checks or to remove another policy statement.

- `ADD POLICY` creates a new policy statement.
- `ADDREPLACE POLICY` specifies that the system either add or replace the following policy statement, as appropriate. If the policy statement is new, the system will add it. If the policy statement exists already, the system will replace it with the one specified.
- `REMOVE POLICY` removes an existing policy statement.
- Use the `UPDATE` option on your policy statement to update check values.

If you do not specify a policy name on your policy statement, the system assigns the statement to the default policy name, which is DEFAULT.

For complete syntax, see "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

When you create an IBM Health Checker for z/OS policy and specify that the system use it, the system applies the values immediately to the existing checks. Then, when you add new checks that meet the policy statement criteria, the values will be applied to those checks as well.

Use the following procedure to create an IBM Health Checker for z/OS policy that persists across restarts:

1. Specify the policy statements in an HZSPRMxx member or members. If you do not specify a policy name when you define a policy statement, the system assigns a default policy name of DEFAULT to the statement.
2. To add the HZSPRMxx member(s) immediately to the list of parmlib members that IBM Health Checker for z/OS processes values from, issue the `F hzsproc,ADD PARMLIB` command.
3. Activate the policy that you want as the current active policy using the `F hzsproc,ACTIVATE POLICY=policy` command. If you do not activate a policy, the system uses policy statements assigned to policy DEFAULT, if there are any. Otherwise, if you do not activate a policy and have no policy statements assigned to policy DEFAULT, you will not have a policy in effect.

   The system applies the values in the active policy to the specified active checks immediately, and re-applies them every time the checks are added or refreshed until an IBM Health Checker for z/OS restart.
4. Refresh all the checks, so that only the values for the current active policy are in use. Use the `F hzsproc,REFRESH,CHECK=(*,*)`. See "Some finer points of how policy values are applied" on page 56 for why this is necessary.
5. To make sure your policy persists across IBM Health Checker for z/OS restarts, specify the HZSPRMxx members containing your policy in either:
   - The START *hzsproc* command in the COMMNDxx parmlib member
   - The IBM Health Checker for z/OS procedure, *hzsproc*

   See "Specifying the HZSPRMxx members you want the system to use" on page 64.

We'll cover the following policy topics:

- "How IBM Health Checker for z/OS builds policies from policy statements"
- "Can I put non-policy statements in my HZSPRMxx member?" on page 59
- "Policy statement examples" on page 63
- "Can I create policy statements using the MODIFY command?" on page 63
- "Specifying the HZSPRMxx members you want the system to use" on page 64

How IBM Health Checker for z/OS builds policies from policy statements

Because a policy is really just a collection of policy statements, there is a lot of flexibility in the way you can define your policy or policies. For example, you can:

- "Define one policy in multiple HZSPRMxx parmlib members"
- "Define multiple policies in one HZSPRMxx parmlib member" on page 56
- Use a combination of both approaches

The system applies policy statements from the active policy to checks in exactly the order they occur in the HZSPRMxx members, and in the order in which you specify the HZSPRMxx members you want the system to use.

Define one policy in multiple HZSPRMxx parmlib members

The following picture shows an example of how policy statements for a single policy (DAY) can be spread between two different parmlib members, HZSPRM01 and HZSPRM02:

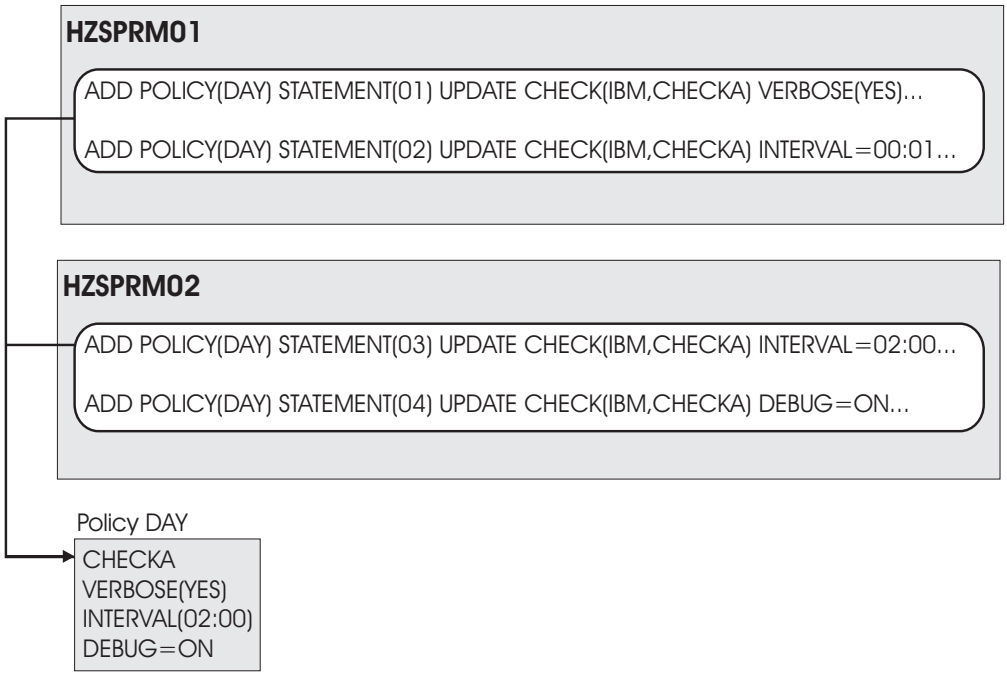


Figure 3. Creating a policy in multiple HZSPRMxx members

Now, if I specify `START hzsproc,HZSPRM=(01,02)`, and activate policy DAY, the system builds the policy from all the policy statements it finds in HZSPRM01 and HZSPRM02, preserving the order in which they were found. When the system applies the policy, it also processes the policy statements in that same order. In this case, statement 03 in HZSPRM02 contradicts the update to the interval made in HZSPRM01. Since HZSPRM02 is specified second on the `START` command, the second interval update is processed and applied last, and so wins out. The final value for interval is 02:00, or once every two hours rather than once a minute.

You can display the complete contents of the DAY policy from any HZSPRMxx parmlib members in use by issuing the following command:

```
F hzsproc,DISPLAY,POLICY=DAY,DETAIL
```

The output might look as follows:

```
HZS202I  11.03.45 POLICY DETAIL    511
POLICY DAY STMT: 01  ORIGIN: HZSPRM01  DATE: 20060501
  UPDATE CHECK(IBM,CHECKA)
  REASON: Change to verbose mode
  VERBOSE: YES

POLICY DAY STMT: 02  ORIGIN: HZSPRM01  DATE: 20060501
  UPDATE CHECK(IBM,CHECKA)
  REASON: Change the interval
  INTERVAL: 00:01

POLICY DAY STMT: 03  ORIGIN: HZSPRM02  DATE: 20060501
  UPDATE CHECK(IBM,CHECKA)
  REASON: Change the interval again
  INTERVAL: 02:00

POLICY DAY STMT: 04  ORIGIN: HZSPRM02  DATE: 20060501
  UPDATE CHECK(IBM,CHECKA)
  REASON: Turn Debug mode on
  DEBUG: ON
```

Note that the output of the detail command display shows the HZSPRMxx parmlib member that a policy statement comes from.

Define multiple policies in one HZSPRMxx parmlib member

The following picture shows an example of how IBM Health Checker for z/OS assembles three different policies from policy statements in a single HZSPRMxx member. Note that the statement that omits a policy name is assigned to policy DEFAULT:

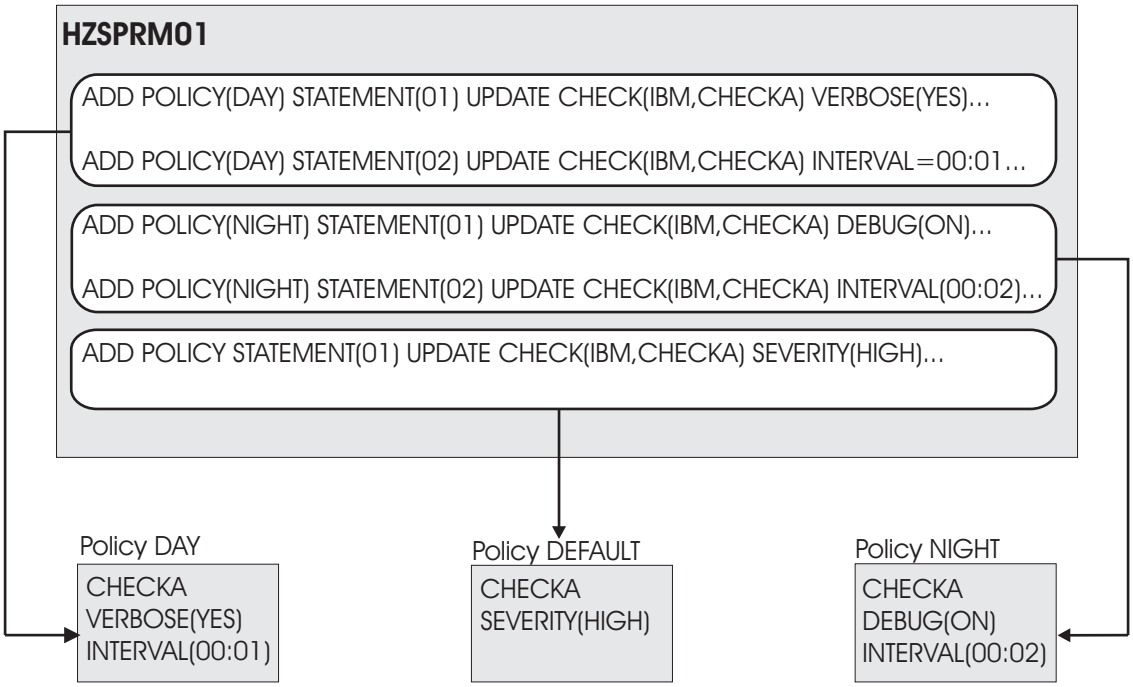


Figure 4. Creating multiple policies in one HZSPRMxx member

Some finer points of how policy values are applied

Generally, all you need to know about the way the system applies policy statement values to checks is that when you activate a policy using the `F hzsproc ACTIVATE POLICY=policy` command, the system applies the values in the active policy to the specified active checks immediately, and re-applies them every time the checks are added or refreshed until IBM Health Checker for z/OS restart. However, there are some nuances to how this works:

REPLACE PARMLIB command nuances: When you issue the following REPLACE PARMLIB commands to replace the existing policy statements, the system starts replace processing by deleting all existing policy statements, and then adding the policy statements in the parmlib members specified in the REPLACE command:

- `REPLACE PARMLIB=xx`
- `REPLACE PARMLIB=xx,POLICY`
- `REPLACE PARMLIB=xx,ALL`

These commands will not reset the active policy, but even the active policy will be affected by this processing.

ACTIVATE POLICY command nuances: Let's look at an example using the policy statements shown in Figure 4.

1. After I've created HZSPRM01 as it appears in Figure 4 on page 56, I issue the `F hzsproc,ADD PARMLIB` command to add HZSPRM01 to the list of parmlib members I want IBM Health Checker to use.
2. Here's where things get interesting. By default, the active IBM Health Checker for z/OS policy is DEFAULT, unless I specify otherwise. That means that if I want the DEFAULT policy to be the active one, I can now either issue the `F hzsproc,ACTIVATE,POLICY=DEFAULT` command or do nothing - I get DEFAULT as the active policy either way. The system applies the values for policy DEFAULT to check CHECKA immediately, upgrading its default severity to HIGH. The system reapplies this values every time CHECKA is refreshed.
3. Now I want to switch to my DAY policy, so I issue the `F hzsproc,ACTIVATE,POLICY=DAY` command. When I do the activate, the system immediately applies the DAY policy values to policy CHECKA. But until I refresh CHECKA, values applied to CHECKA include both DAY values and any DEFAULT values that policy DAY does not contradict. This means that the pre-refresh values currently in use for CHECKA after I activate policy DAY include:
   - SEVERITY(HIGH) from DEFAULT
   - VERBOSE(YES) from DAY
   - INTERVAL(00:01) from DAY
4. But I wanted only the DAY values applied to CHECKA! What do I do? I refresh CHECKA using command `F hzsproc,REFRESH,CHECK=(IBM,CHECKA)`. After refresh, my values for CHECKA include only DAY values:
   - VERBOSE(YES) from DAY
   - INTERVAL(00:01) from DAY

   Refreshing CHECKA resets all the values to their default setting except for the active DAY policy values.
5. Wait, there's more. Now I want to switch to policy NIGHT, so I issue my `F hzsproc ACTIVATE POLICY=NIGHT` command. Until I refresh CHECKA, values applied to CHECKA will also include some DAY and some NIGHT values:
   - VERBOSE(YES) from DAY
   - DEBUG(ON) from NIGHT
   - INTERVAL(00:02) from NIGHT

   Notice that NIGHT value INTERVAL(00:02) overrides the DAY interval of 00:01.
6. When I refresh CHECKA, I get just the NIGHT values:
   - DEBUG(ON) from NIGHT
   - INTERVAL(00:02) from NIGHT

START HZSPROC policy nuances: Some of the accumulating effects of applying and activating two policies in a row, without refreshing checks in between, show a different behavior at Health Checker start-up time.

For example consider having one or more HZSPRMxx parmlib members with essentially the following policy related statements and no ADD CHECK statements:

1. ADDREPLACE POLICY(FIRSTPOLICY) STATEMENT(somestmt11) ...
2. ACTIVATE POLICY(FIRSTPOLICY)
3. ADDREPLACE POLICY(SECONDPOLICY) STATEMENT(somestmt21) ...
4. ACTIVATE POLICY(SECONDPOLICY)

What you might expect is that all checks will have the settings from both FIRSTPOLICY and SECONDPOLICY applied, cumulatively, when Health Checker has completed its start phase. But, no, only settings from the SECONDPOLICY will be applied. This is because the system will read those parmlib members before adding any checks (from the HZSADDCHECK exit...), and therefore the FIRSTPOLICY will only be applied to the still empty set of health checks that the systems knows about at that point. At the point when checks actually get added to Health Checker only the SECONDPOLICY is active anymore and only its settings will be applied to those checks.

Compare this to the situation where you use a MODIFY HZSPROC,ADD,PARMLIB=xx command to apply those policy statements not at start-up time, but after Health Checker is already up and running. Then you get the accumulated effect as described in the previous sections.

Therefore, it is not a good idea to have multiple ACTIVATE POLICY statements, for different policies, in your initial set of HZSPRMxx parmlib members.

How IBM Health Checker for z/OS uses the dates on policy statements

When you specify a policy statement, you must include a date. The system checks this date against the date that the check was added to the system, the add-check date. If the policy statement date is older than the add-check date, then it means that the policy statement might have been written against an older version of the check and thus might no longer be appropriate. For that reason, the system will not apply a policy statement whose date is older than the check date. We call this a policy date exception.

You can display the checks to which an outdated policy statement would apply using the following MODIFY command:

```
F hzsproc,DISPLAY,CHECK(*,*),POLICYEXCEPTIONS
```

You can display the outdated policy statements using the following MODIFY command:

```
F hzsproc,DISPLAY,POLICY=policyname,OUTDATED
```

The system will also issue message HZS0420E if it finds a policy with a date older than a check it applies to:

```
HZS0420E nnn CHECKS HAVE BEEN FOUND FOR WHICH AT LEAST ONE MATCHING
POLICY STATEMENT HAD A DATE OLDER HAN THE CHECK DATE.
THE POLICY STATEMENTS WERE NOT APPLIED TO THOSE CHECKS.
THE FIRST CASE IS
CHECK(checkowner,checkname)
MATCHED BY POLICY STATEMENT stmt.
```

This message tells the installation to reevaluate the policy statement for the updated check.

To display the add-check date, issue the following command for the check or checks identified by the `F hzsproc,DISPLAY,CHECK=(*,*),POLICYEXCEPTIONS` command and find the default date in the output:

```
F hzsproc,DISPLAY,CHECK=(check_owner,check_name),DETAIL
```

If you want to bypass the comparison of dates between the policy statement and the check, use the DATE(*yyyymmdd*,NOCHECK) parameter on the policy statement in HZSPRMxx. You might use the NOCHECK parameter, for example, to bypass

verification so that you do not have to update the policy statement date for minor changes to a check. The following example shows the use of NOCHECK on a policy statement:

```
ADDREPLACE POLICY(policyname) STMT(GLOBAL)
           UPDATE CHECK(IBMGRS,GRS_CONVERT_RESERVES)
           ADDCAT (GLOBAL) REASON('GRS_CONVERT_RESERVES in global category')
           DATE(20050901,NOCHECK)
```

- NOCHECK is ignored for:
  - POLICY UPDATE with PARM, ACTIVE, INACTIVE, SEVERITY, and/or INTERVAL
  - POLICY DELETE
- When NOCHECK is processed, the policy statement is applied to the matching check or checks.
- When NOCHECK is not processed , **and** the date for the matching check is equal to or older than the specified policy statement date, the system applies the policy statement. If a matching check date is newer than the policy statement date, the system does not apply the policy statement.

| As an alternative to NOCHECK to indicate "apply this update, no matter what
| DATE", you can use system symbols to set the DATE to the "day of last Health
| Checker start" via: DATE(&YR4&LMON&LDAY).

Can I put non-policy statements in my HZSPRMxx member?

Your HZSPRMxx member should include only policy statements, the LOGGER parameter, and ADD | ADDREPLACE CHECK statements. Policy statements are appropriate because the system applies them every time IBM Health Checker for z/OS starts up, as well as when checks are added or refreshed. On the other hand, the system applies non-policy statements, such as UPDATE, ADDNEW, or DISPLAY, only to currently active checks, and the statements are applied just once. This means that including non-policy statements in your HZSPRMxx member will be ineffective. Non-policy statements that are in your HZSPRMxx member will not be part of your IBM Health Checker for z/OS policy.

Using SYNCVAL in a policy to specify the time of day that a check runs

You can use the **SYNCVAL** parameter in a policy to give you more control over when a check runs, or at least when it is scheduled to run. Because SYNCVAL, INTERVAL, and EXCEPTINTERVAL all work together, you must coordinate the values. SYNCVAL lets you specify when a check is scheduled to run, while INTERVAL and EXCEPTINTERVAL specify how often it runs, synchronized with the SYNCVAL parameter.

You might find SYNCVAL useful to schedule check workload for a low-stess time like overnight maintenance windows, to schedule checks for specific times in order to balance the overall system workload, or maybe you simply have a check that you want to run very very predictably.

Note that:

- You can only use SYNCVAL in a policy statement - it is **valid only** on an `ADD|ADDREPLACE, POLICY STMT(statement_name) UPDATE,filters,update_options` statement in the *update_options*.
- You can use INTERVAL and EXCEPTINTERVAL on either the :
  - UPDATE or ADD|ADDREPLACE,POLICY STMT with UPDATE

– ADD/ADDREPLACE ,CHECK statement

The syntax for SYNCVAL is:

```
SYNCVAL={SYSTEM|hh:mm|*:mm)
```

For the full syntax, see "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

The way you specify SYNCVAL changes the way the system measures the INTERVAL for a check:

- If you specify or default to **SYSTEM**, nothing changes. The check runs immediately after being added, and subsequently thereafter at the interval defined for the check. The specified interval time starts ticking away when a check finishes running.
- If you specify SYNCVAL=(*hh*:*mm* | *:*mm*, you are specifying the time of day at which the check is scheduled to run. Then the INTERVAL or EINTERVAL specify how often. When you specify SYNCVAL=(*hh*:*mm* | *:*mm*), the interval time starts ticking when the check *starts* running (not when it finishes).

  The INTERVAL | EINTERVAL you specify interacts with SYNCVAL as follows:

  – If the INTERVAL is 24 hours, the check is scheduled to run at the SYNCVAL time of day, every day.
  – If the INTERVAL is less than 24 hours, the check is scheduled to run multiple times per day as specified by the INTERVAL, and one of those times is the SYNCVAL time of day. Thus you are using the SYNCVAL time to synchronize the check interval.
  – If the INTERVAL is greater than 24 hours, the check does not run every day, honoring the INTERVAL, but when it is scheduled to run, it will be at the SYNCVAL time of day.
  –

Examples: Making SYNCVAL work for you

SYNCVAL={*hh*:*mm*|*:*mm*) sounds more complex than it is. This section will go into a bunch of detail about how it works, but don't let all this verbiage scare you - in the end, SYNCVAL just lets you establish a synchronization point so that you can predict and control when a check is scheduled to run.

Note that in the examples below, we talk about coordinating with INTERVAL. But all this information also applies to EXCEPTINTERVAL.

Example 1 - I want exact run times for my check: Lets say we've got a check, CHECKA. Because we like to do things the best and most efficient way, we set up a nice policy statement for CHECKA for our HZSPRMxx parmlib member so that the way we want the check to run is preserved in perpetuity. We need this check to run at synchronized predictable intervals, so we have the following list of assumptions for our CHECKA:

- CHECKA has a runtime of 1 minute.
- We want CHECKA to every 30 minutes.
- We want CHECKA to run at predictable, synchronized times, starting at 12:00 noon.

To make this happen, we will specify a policy statement that combines the INTERVAL we want with the SYNCVAL start point that provides the point of synchronization for predictable run times:

```
ADDREPLACE POLICY(policyname) STMT(GLOBAL)
           UPDATE CHECK(CHECKOWNERA,CHECKA)
           SYNCVAL(12:00) INTERVAL(00:30)
           REASON('Synchronize CHECKA run time')
           DATE(20110112)
```

Once this policy takes effect, the system schedules the check to run at 12:00 and every 30 minutes thereafter, at exactly 12:30, 1:00, 1:30 and so on. If you already have CHECKA defined and running on the system when you set up a policy statement with SYNCVAL for it, you have to refresh your check in order for the initial SYNCVAL start time take effect. However, note that if you add CHECKA in the future, it is synchronized to 15,30,45, and 0 minutes after the hour, even without a REFRESH.

Now, just for fun, we'll add EXCEPTINTERVAL(HALF) to the ADDREPLACE policy:

```
ADDREPLACE POLICY(policyname) STMT(GLOBAL)
           UPDATE CHECK(CHECKOWNERA,CHECKA)
           SYNCVAL(12:00) INTERVAL(00:15)
           EXCEPTINTERVAL(HALF)
           REASON('Synchronize CHECKA run time')
           DATE(20110112)
```

Once this policy takes effect, if the check finds an exception the interval time is halved, so that the system schedules the check to run at 12:00 and every **15** minutes thereafter, at exactly 12:15, 12:30, 12:45 and so on.

Example 2 - I want my check to run at a specific time every day: I want to run CHECKB once a day at midnight (00:00) so it's not interfering with any other applications (not that a check typically makes much performance impact). Here's a policy statement that will make that happen:

```
ADDREPLACE POLICY(policyname) STMT(GLOBAL)
           UPDATE CHECK(CHECKOWNERB,CHECKB)
           SYNCVAL(00:00) INTERVAL(24:00)
           REASON('Make CHECKB run at midnight')
           DATE(20110112)
```

Now, when the policy takes effect, the check is scheduled to run once a day exactly at midnight.

Example 3 - I want my check to run at the same minute of the hour, every time it does run: I just want to be able to predict that CHECKC will only run at 15 minutes past the hour when it runs (for example, because my system has other work scheduled every hour) at the top of the hour. I want CHECKB to run every 6 hours, but at 15 minutes past the hour. Here's my statement:

```
ADDREPLACE POLICY(policyname) STMT(GLOBAL)
           UPDATE CHECK(CHECKOWNERC,CHECKC)
           SYNCVAL(*:15) INTERVAL(06:00)
           REASON('Make CHECKC run at 15 minutes after the hour')
           DATE(20110112)
```

That works. But look out for the gotchas; SYNCVAL and INTERVAL have to synch up, if you will. For example, `SYNCVAL(*:15) INTERVAL(06:00)` will work. But `SYNCVAL(*:15) INTERVAL(00:18)` will not work - you're asking the system to run CHECKC every 18 minutes at 15 minutes after the hour. See“SYNCVAL restrictions” on page 62

Fine points of how SYNCVAL works

What happens if I specify SYNCVAL in a policy statement for a check that's already running? If you specify and activate a policy with a SYNCVAL value for a check that is already running, the SYNCVAL initial start time value is used only as a synchronization point for subsequent check iterations until IBM Health Checker for z/OS is restarted or the check is refreshed.

I want to manually run a check that has a SYNCVAL specified for it. What happens to my synchronization? Nothing! Your synchronization is intact! If you issue a MODIFY *hzsproc*,RUN command against a check that has a SYNCVAL synchronization point, that manual check run will not interfere with the SYNCVAL synchronized check run schedule. The check runs when run manually and then at its next already scheduled run time.

My check missed the initial SYNCVAL time. When will it run? Lets say you specify SYNCVAL={*hh*:*mm*|*:*mm*) value for a check and activate the policy statement, but the check run misses the specified first run time for some reason (for example, if the check is added inactive, gets deactivated, or z/OS UNIX System Services is down). What happens in this case is that the start time is moved out to the next possible start time that matches the SYNCVAL setting. Note that the start time does **not** simply move to the next SYNCVAL-synchronized interval instead.

Need an example? Okay, lets say CHECKA needs z/OS UNIX System Services to run and so is was added with `ADD CHECK ... USS(YES)`. Now you define and activate a policy for the check, as follows:

```
ADDREPLACE POLICY(policyname) STMT(GLOBAL)
           UPDATE CHECK(CHECKOWNERA,CHECKA)
           SYNCVAL(12:00) INTERVAL(00:15)
```

However, z/OS UNIX System Services is down at 12:00 noon, so CHECKA can't run, missing the noon start time. CheckA won't run at all until noon of the next day.

Gotcha - don't make your INTERVAL shorter than the check run time: If you specify an INTERVAL that is shorter than the check run time, note that you might miss a SYNCVAL point. In this case, the system schedules the check on the next possible SYNCVAL point.

If the INTERVAL is too short with respect to the running of the check, you might miss a SYNCVAL point, in which case the system will schedule the check on the next SYNCVAL point (this is simlar tothe "start time is moved out" case. So maybe it doesn't really need saying (it's really a "don't do that").

SYNCVAL restrictions

Make sure that the values for SYNCVAL and INTERVAL / EXCEPTINTERVAL parameters work validly together. These parameters must be coordinated whether you specify SYNCVAL and INTERVAL/EXCEPTINTERVAL on the same policy statement, or just use the currently defined INTERVAL/EXCEPTINTERVAL for the check.

- For SYNCVAL(*hh*:*mm*) and {INTERVAL|EXCEPTINTERVAL}(*hhh*:*mm*), the *hhh*:*mm* value in total minutes (*hhh**60 + *mm*) must be a divisor or multiple of 1440 minutes (24 hours).
- For SYNCVAL(*:*mm*) and {INTERVAL|EXCEPTINTERVAL}(*hhh*:*mm*), the *hhh*:*mm* value in total minutes (*hhh**60 + *mm*) must be a divisor or multiple of 60 minutes (1 hour).

Note that the EXCEPTINTERVAL values of HALF or SYSTEM are valid with any SYNCVAL value specified.

Policy statement examples

- **The basic syntax for a policy statement** that updates a check should look something like this:

  ```
  ADDREPLACE POLICY STMT(statement_name) UPDATE CHECK(check_owner,
  check_name) options REASON('reason_for_change') DATE(yyyymmdd)
  ```

  The ADDREPLACE POLICY statement is identified by the statement name defined in the STMT parameter. If you have already defined a policy statement with the same name, the system replaces it with the new policy statement, as long as the DATE specified is more current than the existing one. For this reason, be careful when you specify ADDREPLACE with a policy statement name that already exists, because you'll most likely be overwriting the old policy statement with your new one.

- **Make all checks low severity except for UNIX System Services checks:**

  ```
  ADDREPLACE POLICY STMT( LOW ) UPDATE CHECK(*,*)
  SEVERITY(LOW) ('Make all checks low severity to start') DATE(20061130)

  ADDREPLACE POLICY STMT( USSMED ) UPDATE CHECK(IBMUSS,*)
  SEVERITY(MEDIUM) REASON('Make all USS checks medium severity') DATE(20061130)
  ```

  – Policy statement **LOW** makes all checks low severity
  – Policy statement **USSMED** then makes the UNIX System Services checks medium severity

- **Update the severity value for all IBMGRS checks**:

  ```
  ADDREPLACE POLICY STMT(POL4) UPDATE CHECK(IBMGRS,*)
      SEVERITY(HIGH) REASON('change policy') DATE(20050901)
  ```

  The system applies the values to all:
  – Existing IBMGRS checks
  – New IBMGRS checks added later

  These same values will be applied to all IBMGRS checks every time they are refreshed or added.

- **Apply the following changes to all checks**:
  – Apply a severity of HIGH
  – Apply a WTO type of IMMEDIATE
  – Use additional descriptor code 16
  – Use routing codes 126,127

  ```
  ADDREPLACE POLICY STMT(POL3) CHECK(*,*) UPDATE SEVERITY(HIGH)
     WTOTYPE(IMMEDIATE) DESCCODE(16) ROUTCODE(126,127)
     REASON(Updating all my checks) DATE(20050920)
  ```

- **Delete a check:**

  ```
  ADDREPLACE POLICY STMT(DEL1) DELETE Check(IBMRACF,RACF_GRS_RNL)
  ```

  We recommend that you delete checks in your policy, see "Why does my check reappear after I delete it? Understanding delete processing" on page 48 for details.

Can I create policy statements using the MODIFY command?

We recommend against creating policy statements using the MODIFY command because the system will not remember changes you made using MODIFY when

IBM Health Checker for z/OS is restarted. The policy is the place to put permanent check changes that you want to have applied to any checks you add in the future. Use HZSPRMxx to create a permanent policy for IBM Health Checker for z/OS.

All the policy statements you create in the HZSPRMxx parmlib member or members you specify that the system is using add up to the single IBM Health Checker for z/OS policy.

Specifying the HZSPRMxx members you want the system to use

- To specify the HZSPRMxx members at **startup time**, specify the two digit suffix of an HZSPRMxx member in one of the following commands:

  START *hzsproc*,HZSPRM=*xx*
  or
  START *hzsproc*,HZSPRM=(*x1,...,xn*)

  In this example, *hzsproc* is the name of the IBM Health Checker for z/OS procedure. Note that if you issue the `START hzsproc` without specifying a parmlib member suffix on the HZSPRM= parameter in either the start command or the IBM Health Checker for z/OS procedure, the system uses the special value HZSPRM=PREV, which will let the system use the HZSPRMxx suffixes as specified on the system parameter HZS (for the first start), or the suffixes which were in use by a previous instance of Health Checker (for secondary starts).

  The IBM Health Checker for z/OS procedure as shipped contains the following:

  ```
  //HZSPROC  JOB  JESLOG=SUPPRESS
  //HZSPROC   PROC  HZSPRM='PREV'
  //HZSSTEP   EXEC    PGM=HZSINIT,REGION=0K,TIME=NOLIMIT,
  //        PARM='SET  PARMLIB=&HZSPRM'
  //*HZSPDATA  DD    DSN=SYS1.&SYSNAME..HZSPDATA,DISP=OLD
  //           PEND
  //           EXEC  HZSPROC
  ```

  The value for PARM= must resolve to `SET PARMLIB=(suffix1,...,suffixn)`.
- To specify the HZSPRMxx members you want IBM Health Checker for z/OS to use each time the system starts it up, see "Tell the system which HZSPRMxx members you want to use" on page 21.
- To specify HZSPRMxx members **dynamically** while IBM Health Checker for z/OS is running, use one of the following modify commands:

  ```
  F hzsproc,ADD PARMLIB=(suffix1,suffix2,...suffixn)
  F hzsproc,REPLACE PARMLIB=(suffix1,suffix2,...suffixn)
  ```

  where *suffixn* is the two digit suffix of an HZSPRMxx member.

Using HZSPRMxx and MODIFY *hzsproc* command

The syntax for both the HZSPRMxx parmlib members and the MODIFY *hzsproc,*parameters command (`F hzsproc,parameters`) follows. You can use the same parameters and syntax for both the HZSPRMxx parmlib member and the `F hzsproc,parameters` command. However, if you want to be consistent with the way commands and parmlib members are specified:

- **Command syntax:** Issue the `F hzsproc,parameters` command as shown in our syntax diagram.
- **HZSPRMxx syntax:** To specify parameters in an HZSPRMxx member:
  - Use parentheses where we show an equal sign. For example:
    - X=Y should be X(Y)
    - X=(Y) should be X(Y)
  - Separate parameters with blanks instead of commas. For example, command

    ```
    UPDATE,CHECK=(IBMAAA,CHECKA)
    ```

    should be as follows in HZSPRMxx:

    ```
    UPDATE CHECK(IBMAAA,CHECKA)
    ```

    See "Guidelines for HZSPRMxx parmlib members" on page 66 for more information.

Parameters take effect for different durations:

- The following parameters are one time actions which are applied immediately. We recommend that you use the **MODIFY** command for these:
  - ADDNEW
  - DELETE
  - DISPLAY
  - REFRESH
  - RUN
  - STOP
  - ADD, or REPLACE,PARMLIB - Note that you can only specify these parameters on the MODIFY command. They are not valid in a HZSPRMxx member.
- The following parameters are applied immediately and remain in effect until the check is refreshed. For any changes you wish to keep, we recommend that you use policy statements in an **HZSPRMxx** parmlib member for these parameters.
  - UPDATE
  - ACTIVATE
  - DEACTIVATE

  If you just want to experiment with temporary check changes, you can use UPDATE, ACTIVATE, and DEACTIVATE in the **MODIFY** command. However, if you want to keep the changes you make with UPDATE, ACTIVATE, and DEACTIVATE past the next refresh, we recommend that you use these parameters in policy statements in an **HZSPRMxx** parmlib member so that they take affect whenever the affected check is added.

  Note that if you put these parameters into HZSPRMxx member(s) but not on policy statements, they act only on checks that have been added at the time that the parmlib member is processed. Particularly during initialization of IBM Health Checker for z/OS, no order is guaranteed among adding of checks and processing of the parmlib member.

- The ADD, ADDREPLACE, or REMOVE POLICY parameters are applied immediately, and are applied again whenever a check is added or refreshed. We recommend that you use policy statements in an **HZSPRMxx** parmlib member for these parameters. See "Creating and maintaining IBM Health Checker for z/OS policies" on page 53.
- The ADD and ADDREPLACE CHECK parameters are applied immediately and remain in effect as long as the parmlib member in which they are defined is in use. See "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Using wildcard characters in MODIFY *hzsproc* and HZSPRMxx: When you have many checks, you can simplify management by using wildcards. You can use wildcard characters * and ?. An asterisk (*) represents any string having a length of zero or more characters. A question mark (?) represents a position which may contain any single character. For example, the following command specifies that all checks with an owner that is 6 characters long beginning with IBM be run:

```
F hzsproc,RUN,CHECK=(ibm???,*)
```

The following HZSPRMxx POLICY statement specifies that a new policy be added that will update all IBMUSS owned checks to a severity of HIGH.

```
ADD POLICY STMT(POL5) UPDATE CHECK(IBMUSS,*) SEVERITY(HIGH)
```

Guidelines for HZSPRMxx parmlib members

The following sections contain guidance for creating an HZSPRMxx parmlib member for IBM Health Checker for z/OS:

HZSPRMxx summary

The following summarizes HZSPRMxx characteristics:

Default member supplied by IBM?
No

Required or optional?
Optional

Directly affects performance?
No

Read at IPL or at command?
S *hzsproc* or F *hzsproc* command.

Allows listing of parameters at IPL or command?
Yes through F *hzsproc*,DISPLAY command

Response to errors:

Syntax error
Error message is issued

Read errors
Error message is issued

Unsupported parameters
Error message is issued

Support for system symbols?
Yes. See What are system symbols? in *z/OS MVS Initialization and Tuning Reference*.

Support for concatenated parmlib?
Yes

Parameter in IEASYSxx (or supplied by the operator)
None.

IBM supplied defaults for HZSPRMxx
The checks provide the default information that you can place in HZSPRMxx to override check defaults. See Chapter 13, “IBM Health Checker for z/OS checks,” on page 381 for check default information.

Syntax rules for HZSPRMxx
Follow the rules in Description and use of the parmlib concatenation in *z/OS MVS Initialization and Tuning Reference*.

The following rules also apply to the creation of HZSPRMxx parmlib members:

- Enter data only in columns 1 through 71. Do not enter data in columns 72 through 80; the system ignores these columns.
- Comments may appear in columns 1-71 and must begin with "/*" and end with "*/".

Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*

Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*

```
MODIFY hzsproc | HZSPRMxx
  ACTIVATE,filters

  ADDNEW

  DEACTIVATE,filters

  DELETE,filters[,FORCE={NO | YES}]

  DISPLAY
    {
    [CHECKS[,filters][,LOCALE=(HZSPROC|REMOTE|REXX|NOTHZSPROC)][,SUMMARY|,DETAIL][,ANY|,NOTDELETED|,DELETED]
  [,POLICYEXCEPTIONS][,EXCEPTIONS][,DIAG]]
    |
    [filters[,LOCALE=(HZSPROC|REMOTE|REXX|NOTHZSPROC)][,SUMMARY|,DETAIL][,ANY|,NOTDELETED|,DELETED][,POLICYEXCEPTIONS]
  [,EXCEPTIONS][,DIAG]]
    |
    [POLICY[=policyname][,STATEMENT=name][,SUMMARY|,DETAIL]}
          [,CHECK=(check_owner,check_name)[,SUMMARY|,DETAIL][,OUTDATED]
    |
    [STATUS]
    |
     POLICIES  }

|  HZSPDATA=datasetname[,VOLUME=volser]

|  SET,OPTION,CHKMSG_NOCONSID=[ON|OFF]

  LOGGER=
   [OFF|ON|ON,LOGSTREAMNAME=logstreamname]

  REFRESH,filters

  RUN,filters

  STOP

  UPDATE,filters
          [,ACTIVE|INACTIVE]
          [,ADDCAT=(cat1,...,cat16)]
          [,DATE={date | (date,NOCHECK)}]
          [,DEBUG={OFF|ON}]
          [,VERBOSE={NO|YES}]
          [,DESCCODE=(desccode1,...,desccoden)]
          [,INTERVAL={ONETIME|hhh:mm}]
          [,EXCEPTINTERVAL={SYSTEM|HALF|hhh:mm}]
          [,SYNCVAL={SYSTEM|hh:mm|*:mm}]
          [,PARM=parameter,REASON=reason,DATE={date | (date,NOCHECK)}]
          [,REASON=reason]
          [,REPCAT=(cat1[,cat2[,...,cat16]])]
          [,REMCAT=(cat1[,cat2[,...,cat16]])]
          [,REXXTIMELIMIT=timelimit]
          [,ROUTCODE=(routcode1,...,routcoden)]
          [,SEVERITY={HIGH|MEDIUM|LOW|NONE}]
          [,WTOTYPE={CRITICAL|EVENTUAL|INFORMATIONAL|HARDCOPY|NONE}]
```

Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*

```
{ADD | ADDREPLACE},CHECK=(check_owner,check_name)
                    ,{CHECKROUTINE=routinename
                     | EXEC=execname
                        ,REXXHLQ=hlq
                       [,REXXTIMELIMIT=timelimitvalue]
                    {  [,REXXTSO=YES]
                    | [,REXXTSO=NO
                        [,REXXIN={NO | YES}
                         ]
                    }
                     }
                     ,MESSAGETABLE={msgtablename | *NONE }
                     ,SEVERITY={HIGH|MEDIUM|LOW}
                     ,INTERVAL={ONETIME|hhh:mm}
                     ,DATE=date
                     ,REASON=reason
                     [,PARM=parameter]
                     [,GLOBAL}
                     [,ACTIVE|INACTIVE]
                     [,EXCEPTINTERVAL={SYSTEM|HALF|hhh:mm}]
                     [,USS={NO|YES}]
                     [,VERBOSE={NO|YES}]
                     [,ENTRYCODE=entrycode | 0]
                     [,ALLOWDYNSEV={NO|YES}]
                     [,DOM={SYSTEM|CHECK}]

ADD,PARMLIB=(suffix1...suffixn)

REPLACE,PARMLIB=(suffix1...suffixn)
     [,{CHECKS|POLICY|ALL}]

ACTIVATE,POLICY=policyname

{ADD | ADDREPLACE}
     ,POLICY[=policyname][,STATEMENT=name],UPDATE,filters[,update_options],REASON=  reason,DATE={date|(date,NOCHECK)}
     |
     ,POLICY[=policyname][,STATEMENT=name],DELETE,filters,REASON=reason,DATE=  {date|(date,NOCHECK)}

REMOVE,POLICY[=policyname],STATEMENT=name
```

The parameters are:

filters

Filters specify which check or checks you wish to take an action against. You can specify wildcard characters * and ? for filters. An asterisk (*) represents any string having a length of zero or more characters. A question mark (?) represents a position which may contain any single character.
The syntax of the filters is as follows:

CHECK=(*check_owner*,*check_name*)
EXITRTN=*exit routine*
CATEGORY=([{ANY|EVERY|EXCEPT|ONLY},][*category1*[,...,*categoryn*]])

CHECK=(*check_owner*,*check_name***)**
check_owner specifies the 1-16 character check owner name. *check_name* specifies the 1-32 character check name. CHECK is a **required** filter, except for the DISPLAY,CHECKS,*filters* command.

EXITRTN=*exit routine*
EXITRTN specifies the HZSADDCHECK exit routine that added the check(s) to IBM Health Checker for z/OS.

CATEGORY=([{ANY|EVERY|EXCEPT|ONLY},][*category1***[,...,***categoryn***]])**
Filter checks by user defined category, see "Using the category filter to manage checks" on page 50.

You can specify one of the following category filters:

ANY
Checks that are in any of the specified categories

EVERY
Checks that are in every specified category

EXCEPT
Checks that are not in any of the specified categories

ONLY
Checks that are in every one of the specified categories and that have only as many categories as are specified. For example, a check assigned to three categories would not match if the CATEGORY=ONLY statement on this MODIFY command specified two categories.

ONLY is the default, but for the sake of clarity, we recommend that you specify the category option that you want.

category1`[,...,`*categoryn*`]`
Specifies the category name or names. You can specify up to 16 named categories, each represented by a 1-16 character string.

ACTIVATE

`ACTIVATE,`*filters*

Sets the specified check or checks to the active state. If the check is eligible to run, ACTIVATE will cause the check to run immediately and reset the interval for the check. You must specify filter CHECK=(*check_owner*,*check_name*) with ACTIVATE. Other filters are optional. See "filters" on page 69.

`ACTIVATE,` *filters* is equivalent to the UPDATE,*filters*,ACTIVE command. See UPDATE ACTIVE and INACTIVE parameters.

ADDNEW

`ADDNEW`

ADDNEW adds checks to IBM Health Checker for z/OS.

- For checks defined and added by a HZSADDCHECK exit routine, ADDNEW calls the HZSADDCHECK exit to add checks to IBM Health Checker for z/OS
- For checks defined and added in an HZSPRMxx parmlib member (using the ADD|ADDREPLACE,CHECK parameters), ADDNEW processes the definitions in parmlib to add checks to IBM Health Checker for z/OS

The system does the following ADDNEW processing for each added check:

- Applies any installation updates in the policy to the default values for the check.
- Loads the check routine, if this is a local check.
- Loads the message table, if it is a local or a REXX exec check.

All checks that are added to IBM Health Checker for z/OS are scheduled to run unless they are not eligible to be run. If a check delete is pending when the ADDNEW parameter is processed, the check will not run until delete processing is complete.

You can use ADDNEW to undelete a check that has been deleted. See "Why does my check reappear after I delete it? Understanding delete processing" on page 48.

DEACTIVATE

`DEACTIVATE,`*filters*

DEACTIVATE disables running of the specified check until ACTIVATE is specified. You must specify filter CHECK=(*check_owner,check_name*) with DEACTIVATE. Other filters are optional. See filters.

DEACTIVATE is the same as the UPDATE,*filters*,INACTIVE command. See UPDATE ACTIVE and INACTIVE parameters.

DELETE

```
DELETE,filters[,FORCE={NO | YES}]
```

Remove the specified check(s) from the IBM Health Checker for z/OS. If specified check or checks are running when the command is issued, the system waits until they are finished running before deleting them. You must specify filter CHECK=(*check_owner,check_name*) with DELETE. Other filters are optional. See filter.

You can undelete a deleted check using the ADDNEW parameter. See "Why does my check reappear after I delete it? Understanding delete processing" on page 48 for more information about DELETE processing.

FORCE={NO | YES}
Specifies whether or not you want to force deletion of a check even if the check is running. FORCE=NO is the default.

You should use FORCE=YES only as a last resort after trying the DELETE parameter with FORCE=NO because:

- FORCE=YES will cause a check to be interrupted in the middle of its processing:
  - FORCE=YES issued against a local check will result in a non-retriable abend on the third try.
  - FORCE=YES issued against a remote or REXX exec check will result in a non-retriable abend.
- FORCE=YES will delete checks that are still in the process of running.

DISPLAY

DISPLAY issues messages with information specified. The different options display the information as follows:

```
DISPLAY
  {
  [CHECKS[,filters][,LOCALE=(HZSPROC|REMOTE|REXX|NOTHZSPROC)][,SUMMARY|,DETAIL][,ANY|,NOTDELETED|,DELETED]
[,POLICYEXCEPTIONS][,EXCEPTIONS][,DIAG]]
  |
  [filters[,LOCALE=(HZSPROC|REMOTE|REXX|NOTHZSPROC)][,SUMMARY|,DETAIL][,ANY|,NOTDELETED|,DELETED]
[,POLICYEXCEPTIONS][,EXCEPTIONS][,DIAG]]
  |
  [POLICY[=policyname][,STATEMENT=name][,SUMMARY|,DETAIL]}
        [,CHECK=(check_owner,check_name)[,SUMMARY|,DETAIL][,OUTDATED]
  |
  [STATUS]
  |
   POLICIES  }
```

CHECKS
CHECKS displays information about checks.

filters
You must specify filter CHECK=(*check_owner,check_name*) with DISPLAY, unless you specify DISPLAY,CHECKS*filters*. Other filters are optional. See filters.

LOCALE=(HZSPROC | REMOTE | REXX | NOHZSPROC)
Specifies whether you want to display local or remote checks:

- **HZSPROC** specifies that you want to display only local checks that run in the *hzsproc* address space.
- **REMOTE** specifies that you want to display only remote checks that run in the caller's address space.
- **REXX** specifies that you want to display only REXX exec checks running under System REXX.
- **NOTHZSPROC** specifies that you want to display both remote and REXX exec checks, but not local checks.

If you do not specify LOCALE, the system displays both local and remote checks.

SUMMARY
IBM Health Checker for z/OS issues message HZS200I with summary information about the specified checks. See the "Example of DISPLAY SUMMARY message output". For each check matching the specified criteria, the following information is returned:

- Check owner
- Check name
- The name of any category the check is a member of
- The current status of the check.

SUMMARY is the default.

DETAIL
IBM Health Checker for z/OS issues message HZS0201I (See "Example of DISPLAY DETAIL message output") with detailed information about the specified check including:

- Check name
- Check owner
- The name of any category the check is a member of
- The Date and time the check was last executed
- The current status of the check
- The check values, such as severity, routing codes, and descriptor codes.

ANY
Displays information about both deleted and non-deleted checks. ANY is the default.

NOTDELETED
Displays information about checks that have not been deleted.

DELETED
Displays information about checks that have been deleted.

POLICYEXCEPTIONS
Display information about checks for which a policy statement matching the check was **not** applied because the date on the policy statement is older than the check date.

EXCEPTIONS
Display information about checks that completed with a non-zero return value.

DIAG
Displays additional diagnostic data such as the check routine address and message table address. See "Example of DISPLAY DIAG message output".

POLICY

Displays the specified policy statements. You can filter DISPLAY POLICY by:

- Policy name
- Policy statement name
- Check owner or name

Output is displayed in message HZS0202I for DETAIL ("Example of DISPLAY POLICY DETAIL message output") or HZS0204I ("Example of DISPLAY POLICY SUMMARY message output") for SUMMARY.

POLICY=*policyname*

Specifies the 1-16 character name of the policy whose policy statements you wish to display. If you do not specify *policyname* , the system displays the current active policy statements. If *policyname* contains wildcards, the system displays all applicable policies and their statements, with a blank line separating each policy.

STATEMENT=*name*

STATEMENT specifies the 1-16 character name of the policy statement whose policy statements you wish to display.

CHECK=(*check_owner***,***check_name***)**

check_owner specifies the 1-16 character check owner name . *check_name* specifies the 1-32 character check name. The system displays the policy statements that apply to the specified checks.

STATUS

Displays status information about IBM Health Checker for z/OS and checks in message HZS0203I (see "Example of DISPLAY STATUS message output"), including the following information:

- Number of checks that are eligible to run
- Number of active checks that are running
- Number of checks that are not eligible to run
- Number of deleted checks
- ASID of the IBM Health Checker for z/OS address space
- The log stream name and its status
- The current HZSPRMxx parmlib suffix list

POLICIES

Displays the names of all policies defined for IBM Health Checker for z/OS.

HZSPDATA

```
HZSPDATA=datasetname[,VOLUME(volser)]
```

Specifies the name and optional volume for the HZSPDATA data set you want IBM Health Checker for z/OS to use to save data required by checks as part of their processing between restarts of the system or IBM Health Checker for z/OS. See “Allocate the HZSPDATA data set to save check data between restarts” on page 10 for more information.

SET,OPTION

CHKMSG_NOCONSID=[ON|OFF]

When turned ON this option can prevent routing problems for check exception messages by omitting the console ID parameter on WTOs sent for health check exception messages.

Once set, the option value only changes when explicitly modified again, until the next IPL, which resets it to OFF. A restart of Health Checker will not modify the option. The current option value can be displayed via operator command

```
MODIFY HZSPROC,DISPLAY,STATUS
```

LOGGER

```
LOGGER=
 [OFF|ON|ON,LOGSTREAMNAME=logstreamname]
```

Use the **LOGGER** parameter to connect to and use a pre-defined log stream whenever a check generates output.

LOGGER=ON,LOGSTREAMNAME=*logstreamname*

The first time you use the LOGGER parameter to connect to the log stream for IBM Health Checker for z/OS, you must specify a log stream name. The log stream name must begin with HZS and must follow system logger naming rules. See *z/OS MVS Setting Up a Sysplex* for information on setting up and managing a log stream.

The system rejects this parameter if the log stream is already in use without any errors. If a log stream is in use with errors when you use the LOGGER parameter, the system disconnects from the current log stream.

After initially specifying LOGGER=ON,LOGSTREAMNAME=*logstreamname*, you can use LOGGER=ON and LOGGER=OFF to toggle use of the log stream on and off.

LOGGER=ON

Connects to and begins using the log stream for check routine messages.

- LOGGER=ON is rejected if a log stream has not already been provided.
- LOGGER=ON has no effect if the log stream is already in use without any errors.

LOGGER=OFF

Stops using the log stream for check routine messages. LOGGER=OFF is the default.

REFRESH

```
REFRESH,filters
```

Refreshes the specified check or checks. Refresh processing first deletes a check from the IBM Health Checker for z/OS and does the ADDNEW function (ADDNEW parameter).

When you issue a command with the REFRESH parameter on it, the system processes the policy statements and applies any changes to check values that the policy statements contain. See “How IBM Health Checker for z/OS builds policies from policy statements” on page 54.

You must specify filter CHECK=(*check_owner*,*check_name*) with REFRESH. Other filters are optional. See "filters".

RUN,*filters*

Run the specified check(s) immediately, one time. Specifying RUN does not reset the check interval. You must specify filter CHECK=(*check_owner*,*check_name*) with RUN. Other filters are optional. See "filters".

STOP

Stop IBM Health Checker for z/OS. Do not use STOP unless absolutely necessary; every time you STOP the IBM Health Checker for z/OS address space, that address space identifier (ASID) becomes unavailable.

To start IBM Health Checker for z/OS, use one of the following commands:

- START *hzsproc*
- START *hzsproc*,HZSPRM=*xx*

UPDATE

```
UPDATE,filters
        [,ACTIVE|INACTIVE]
        [,ADDCAT=(cat1,...,cat16)]
        [,DATE={date | (date,NOCHECK)}]
        [,DEBUG={OFF|ON}]
        [,VERBOSE={NO|YES}]
        [,DESCCODE=(desccode1,...,desccoden)]
        [,INTERVAL={ONETIME|hhh:mm}]
        [,EXCEPTINTERVAL={SYSTEM|HALF|hhh:mm}]
        [,SYNCVAL={SYSTEM|hh:mm|*:mm}]
        [,PARM=parameter,REASON=reason,DATE={date | (date,NOCHECK)}]
        [,REASON=reason]
        [,REPCAT=(cat1[,cat2[,...,cat16]])]
        [,REMCAT=(cat1[,cat2[,...,cat16]])]
        [,REXXTIMELIMIT=timelimit]
        [,ROUTCODE=(routcode1,...,routcoden)]
        [,SEVERITY={HIGH|MEDIUM|LOW|NONE}]
        [,WTOTYPE={CRITICAL|EVENTUAL|INFORMATIONAL|HARDCOPY|NONE}]
```

UPDATE allows you to temporarily update the current default or override values for the specified checks. Values updated are in effect until the next refresh for specified checks. You cannot update checks that are deleted or have a delete pending against. Note that adding or removing the check from a category does not effect the current check implementations.

You must specify filter CHECK=(*check_owner,check_name*) with UPDATE. Other filters are optional. See "filters".

If an UPDATE request does not actually change anything, the system takes no action in response to the request. For example, if you use an UPDATE request to make the severity HIGH for a check whose severity is already HIGH, the system does not process the request, and nothing is done.

ACTIVE | INACTIVE

Use the **ACTIVE** and **INACTIVE** parameters to change the state of the check. See “Understanding check state and status” on page 34. These parameters are equivalent to "ACTIVATE parameter" and "DEACTIVATE parameter".

ADDCAT=(*cat1***,...,***catn)*

ADDCAT adds specified checks into each of the listed categories. You can specify up to 16 categories, each a 1-16 character category name.

DATE={*date* **| (***date***,NOCHECK)}**

DATE specifies when the values for a check were last updated. The date is specified in the format *yyyymmdd*. If the date specified on the UPDATE parameter is earlier than the date for the check, the system does not process the values specified on the UPDATE parameter because the UPDATE statement may no longer be appropriate for the check.

NOCHECK

IBM Health Checker for z/OS verifies the date for the UPDATE statement, making sure that it is equal to or newer than the date for

the check that the statement applies to. Use NOCHECK to bypass date verification so that you do not have to update the UPDATE statement date for minor changes to a check.

If you use the NOCHECK parameter, you must use parentheses on the DATE parameter. For example, you can specify either `DATE=date` or `DATE=(date,NOCHECK)`.

If your POLICY UPDATE statement contains any of the following, NOCHECK is ignored: PARM, ACTIVE, INACTIVE, SEVERITY, INTERVAL. NOCHECK is also ignored for a POLICY DELETE statement.

DEBUG={OFF|ON}

DEBUG specifies the debug mode desired:

- **OFF** specifies that debug mode is off, which is the default.
- **ON** specifies that you want to run with debug mode on. Turning debug mode ON lets you see debug messages, if the check produces any, which are designed to help a product or installation debug a check routine or to display additional check information. Debug messages are only issued when the check is in debug mode. See the individual check descriptions in Chapter 13, "IBM Health Checker for z/OS checks," on page 381 to see if a check issues additional information when you specify DEBUG=ON.

  Using SDSF to view check output in the message buffer when the debug mode is ON allows you to see the message IDs of debug messages.

  Debug mode ON will not take effect until you re-run the check or checks. For example, after you issue a command to turn debug mode ON, you could issue the command with the RUN parameter, which will run the check or checks with debug mode ON.

  For a REXX exec check, DEBUG must be ON for the check to write data or error messages to an output data set.

VERBOSE={NO|YES}

VERBOSE specifies the verbose mode desired:

- **NO** specifies that you do not want to run in verbose mode.
- **YES** specifies that you want to run in verbose mode. Running in verbose mode you see additional messages about non-exception conditions, if the check supports verbose mode. These messages are only issued when the check is in verbose mode. See the individual check descriptions in Chapter 13, "IBM Health Checker for z/OS checks," on page 381 to see if a check supports issues additional messages if you specify VERBOSE=YES.

  Verbose mode does not take effect until you re-run the check or checks. For example, after you issue a command to turn verbose mode on, you could issue the `F hzsproc` command with the RUN parameter to run the check or checks with verbose mode on.

DESCCODE=(*desccode1*[,...,*desccoden*])

DESCCODE specifies descriptor code(s) **in addition** to the system default descriptor code used when an exception message is issued by the specified check. You can specify a list of up to 13 descriptor codes, each with a value between 1 and 13

See"WTOTYPE" for a list of the message types and corresponding default descriptor codes. For example, if you specify DESCODE=(7) for a low

severity check, the system uses descriptor code 7 **in addition** to the default descriptor code of 12 for the check. See Descriptor codes in *z/OS MVS System Messages, Vol 1 (ABA-AOM)*.

If you do not specify DESCCODE or if you specify DESCCODE=0, the system uses just the system-default descriptor code when the exception message is issued.

INTERVAL={ONETIME|*hhh*:*mm*}

INTERVAL specifies how often the check should run:

- **ONETIME** specifies that the check should run only once, ONETIME specifies that the check should run only once, kicked off either by refresh processing or by the SYNCVAL scheduled run time.
- *hhh:mm* provides a specific interval for the check to run. The check will run at refresh time and periodically afterwards at the interval specified or in alignment with the SYNCVAL parameter.

  hhh indicates the number of hours, from 0 to 999. *mm* indicates the number of minutes, from 0 to 59.

The system starts processing the interval depending on whether you are using the SYNCVAL parameter:

- If you are using SYNCVAL={*hh*:*mm*|*:*mm*}, the specified interval time starts ticking away when the check is scheduled to run, to align it with the SYNCVAL start time.
- If you are not using SYNCVAL or using SYNCVAL=SYSTEM, the specified interval time starts ticking away when a check finishes running.

EXCEPTINTERVAL={SYSTEM | HALF | *hhh*:*mm*}

EXCEPTINTERVAL specifies how often the check should run after the check has found an exception. This parameter allows you to specify a shorter check interval for checks that have found an exception.

- **SYSTEM**, which is the default, specifies that the EXCEPTINTERVAL is the same as the INTERVAL.
- **HALF** specifies that the exception interval is defined to be half of the normal interval. For example, the exception interval will be set to 30 seconds, if the normal interval is set to 1 minute.

  Note that if you change the INTERVAL parameter for a check, the system will also recalculate the exception interval.
- *hhh:mm* provides a specific exception interval for the check. After raising an exception, the check will run at the exception interval specified. *hhh* indicates the number of hours, from 0 to 999. *mm* indicates the number of minutes, from 0 to 59.

The system starts processing the interval depending on whether you are using the SYNCVAL parameter:

- If you are not using SYNCVAL or using SYNCVAL=SYSTEM|HALF, the specified interval time starts ticking away when a check finishes running.
- If you are using SYNCVAL={*hh*:*mm*|*:*mm*}, the specified interval time starts ticking away when the check starts running, to align it with the SYNCVAL start time.

SYNCVAL={SYSTEM|*hh*:*mm*|*:*mm*}

SYNCVAL specifies a synchronization value you can use to control when a

check is scheduled to run. SYNCVAL works with both the INTERVAL and EXCEPTINTERVAL parameters, so the values must be coordinated.

You can only use SYNCVAL in a policy statement - it is **valid only** on an `ADD|ADDREPLACE, POLICY STMT(statement_name)` `UPDATE,filters,update_options` statement in the *update_options*.

See "Using SYNCVAL in a policy to specify the time of day that a check runs" on page 59 for examples and how to coordinate SYNCVAL with INTERVAL and EXCEPTINTERVAL.

The values for SYNCVAL include:

SYSTEM
Specifies that the check run immediately after being added, if eligible, and subsequently at the interval defined for the check. The specified interval time starts ticking away when a check finishes running. SYSTEM is the default.

hh:*mm*
Specifies a specific time in hours and minutes in the day to schedule the check to run for the first time, as well as any subsequent iterations synchronized with the current INTERVAL or EXCEPTINTERVAL value.

Check run time is scheduled to aligned with the SYNCVAL start time. The INTERVAL time for the check starts ticking away when the check starts running.

hh Valid values are 0 through 23.

mm Valid values are 0 through 59.

Make sure that the values for SYNCVAL and INTERVAL/EXCEPTINTERVAL parameters work validly together. These parameters must be coordinated whether you specify SYNCVAL and INTERVAL/EXCEPTINTERVAL on the same policy statement, or just use the currently defined INTERVAL/EXCEPTINTERVAL for the check. For SYNCVAL(*hh*:*mm*) and INTERVAL(*hhh*:*mm*), the *hhh*:*mm* value in total minutes (*hhh**60 + *mm*) must be a divisor or multiple of 1440 minutes (24 hours).

*****:*mm*
Specifies the minute in an hour the check is scheduled to run for the first time, as well as for subsequent runs, synchronized with the current INTERVAL or EXCEPTINTERVAL value. If the given minute has already passed for the current hour, the check is scheduled for the specified minute in the next hour.

Check run time is scheduled to aligned with the SYNCVAL start time. The INTERVAL time for the check starts ticking away when the check starts running.

Make sure that the values for SYNCVAL and INTERVAL/EXCEPTINTERVAL parameters work validly together. These parameters must be coordinated whether you specify SYNCVAL and INTERVAL/EXCEPTINTERVAL on the same policy statement, or just use the currently defined INTERVAL/EXCEPTINTERVAL for the check. For SYNCVAL(*:*mm*) and INTERVAL(*hhh*:*mm*), the *hhh*:*mm* value in total minutes (*hhh**60 + *mm*) must be a divisor or multiple of 60 minutes (1 hour).

Valid values for *mm* are 0 to 59.

PARM=*parameter*

PARM specifies the check specific parameters being passed to the check. The value for *parameter* can be 1-256 text characters. You can specify these characters as:

- A single value enclosed in single quotes.
- Multiple values, where either each value is enclosed in single quotes or the group of values is enclosed is single quotes. The system will separate these values from each other by a comma. For example, both PARM='p1,p2,p3' and PARM=('p1','p2','p3') will both resolve to a parameter value of p1,p2,p3.

You can also specify PARM with a null parameter string, PARM=(), to remove all parameters and clear a parameter error for a check that does not accept parameters.You must specify the **REASON** and **DATE** parameters when you specify **PARM**.

When you issue a command with the PARM parameter, the check runs immediately.

REASON=*reason*

REASON specifies the unique reason the check specific parameters are being overridden. The value for *reason* can be 1-126 text characters. You can specify these characters as:

- A single value enclosed in single quotes.
- Multiple values, where either each value is enclosed in single quotes or the group of values is enclosed is single quotes. The system will separate these values from each other by a blank. For example, both REASON='r1 r2 r3' and REASON=('r1','r2','r3') will both resolve to a reason of r1 r2 r3.

REMCAT=(*cat1***,...,***catn***)**

REMCAT removes the specified checks from the categories listed. You can specify up to 16 categories, each a 1-16 character category name.

REPCAT=(*cat1***,...,***catn***)**

REPCAT removes the specified checks from any existing categories they belong to and adds them to the categories listed.

REXXTIMELIMIT=*timelimit*

REXXTIMELIMIT specifies an optional input parameter that is the number of seconds to which the execution of an iteration of the exec is to be limited. You can specify a value between 0 and 21474536 for REXXTIMELIMIT.

A value of 0 is treated the same as no time limit. The default is that there is no time limit.

ROUTCODE=(*routcode1***,...,***routcoden***)**

ROUTCODE specifies the routing codes to be used when an exception message is issued by the specified check. You can specify a list of up to 128 routing codes, each with a value between 1 and 128.

If ROUTCODE is not specified, or if ROUTCODE=0 is specified, the system uses the system-default routing codes for exception messages.

SEVERITY={HIGH | MEDIUM | LOW | NONE}

SEVERITY overrides the default check severity. The severity you pick determines how the exception messages the check routine issues with the HZSFMSG service are written.

- HIGH indicates that the check routine is checking for high-severity problems. All exception messages that the check issues with the HZSFMSG service will be issued to the console as critical eventual action messages. HZS0003E is issued, which includes message text defined by the check owner.
- MEDIUM indicates that the check routine is looking for medium-severity problems. All exception messages the check issues with HZSFMSG will be issued to the console as eventual action messages. HZS0002E is issued which includes message text defined by the check owner.
- LOW indicates that the check is looking for low-severity problems. All exception messages the check issues with HZSFMSG will be issued to the console as informational messages. HZS0001I is issued which includes message text defined by the check owner.
- NONE indicates that you're assigning no severity to this check. Exception messages issued by the check with HZSFMSG are issued to the hardcopy log, rather than the console. HZS0004I is issued which includes message text defined by the check owner.

Note that there are a couple of things that can override the SEVERITY specifications and implicit behaviors derived from SEVERITY:

- The WTOTYPE parameter overrides the implicit WTO type derived from the SEVERITY defined for a check - see"WTOTYPE".
- A check using the dynamic check severity function can override the SEVERITY you define for a check - see “Writing a check with dynamic severity levels” on page 124.

WTOTYPE={CRITICAL|EVENTUAL|INFORMATIONAL|HARDCOPY|NONE}

WTOTYPE specifies the type of WTO you want the system to issue when the check finds an exception. This parameter includes all WTOs issued by a check.

The WTOTYPE specified on the UPDATE CHECK statement overrides the implicit WTO type derived from either:

- The SEVERITY specified for the check on either the ADD or UPDATE CHECK statement.
- The dynamic severity specified by the check. See “Writing a check with dynamic severity levels” on page 124.

Note that the WTOTYPE only determines what type of action is associated with a check exception WTO. Check status (such as EXCEPTION, MEDIUM, or SUCCESSFUL), remains unaffected by WTOTYPE because it is derived solely from the current severity specified in either the check definition or dynamically by the check itself.

CRITICAL
Specifies that the system issue an critical eventual action message (with a descriptor code of 11) when the check finds an exception. This is the default if SEVERITY(HIGH) is specified or defaulted to.

EVENTUAL
Specifies that the system issue an eventual action message (with a descriptor code of 3) when the check finds an exception. This is the default if SEVERITY(MEDIUM) is specified or defaulted to.

INFORMATIONAL
Specifies that an informational message with a descriptor code of 12 should be issued when an exception is found. This is the default if SEVERITY(LOW) is specified or defaulted.

HARDCOPY
Specifies that the system issue the message to the hardcopy log. This is the default if SEVERITY(NONE) is specified.

NONE
Specifies that no WTO be issued when the check finds an exception.

{ADD | ADDREPLACE},CHECK

```
{ADD | ADDREPLACE},CHECK=(check_owner,check_name)
                      ,{CHECKROUTINE=routinename
                       | EXEC=execname
                          ,REXXHLQ=hlq
                         [,REXXTIMELIMIT=timelimitvalue]
                      {  [,REXXTSO=YES]
                      | [,REXXTSO=NO
                          [,REXXIN={NO | YES}
                             ]
                      }
                       }
                       ,MESSAGETABLE={msgtablename | *NONE }
                       ,SEVERITY={HIGH|MEDIUM|LOW}
                       ,INTERVAL={ONETIME|hhh:mm}
                       ,DATE=date
                       ,REASON=reason
                       [,PARM=parameter]
                       [,GLOBAL}
                       [,ACTIVE|INACTIVE]
                       [,EXCEPTINTERVAL={SYSTEM|HALF|hhh:mm}]
                       [,USS={NO|YES}]
                       [,VERBOSE={NO|YES}]
                       [,ENTRYCODE=entrycode | 0]
                       [,ALLOWDYNSEV={NO|YES}]
                       [,DOM={SYSTEM|CHECK}]
```

Allows you to add or replace a check definition in an HZSPRMxx parmlib member. The parameters correspond to the parameters in the HZSADDCK macro.

ADD
Allows you to add a check definition to an HZSPRMxx parmlib member. If you use ADD,CHECK to add a check that has already been defined, the request is rejected. When a check that was added in this way is subsequently deleted, the check definition still remains. ADDNEW or REFRESH command processing will bring the check back exactly as it was defined.

ADDREPLACE
Specifies that the system either add or replace the check definition, as appropriate. If the check defiinition is new, the system will add it. If the check definition exists already, the system will replace it with the one specified.

CHECKROUTINE=*routinename* | **EXEC=***execname*
Use CHECKROUTINE or EXEC to specify the type of check you are adding or replacing.

- CHECKROUTINE=*routinename* specifies that you are defining a local or remote assembler check. Required parameter specifies a module name for the check you are adding or replacing. The system gives control to the entry point of this module to run the check. The check routine module must be in an APF-authorized library and the system must be able to locate the check routine within the joblib or steplib of the IBM Health Checker for z/OS address space, the LPA, or the LNKLST.

- EXEC=*execname* specifies that you are defining a REXX exec check. Required parameter specifies an exec name for the check you are adding or replacing. The exec does not have to be in an APF-authorized library.

REXXHLQ=*hlq*
When EXEC=*execname* is specified, REXXHLQ is a required input parameter specifying the high level qualifier for data sets(s) to be made available to the REXX exec. See "Using REXXOUT data sets" on page 175 for information on how the system determines the name of your input or output data set.

REXXTIMELIMIT=*timelimit*
When EXEC is specified, REXXTIMELIMIT specifies an optional input parameter that is the number of seconds to which the execution of an iteration of the exec is to be limited. You can specify a value between 0 and 21474536 for REXXTIMELIMIT.

A value of 0 is treated the same as no time limit. The default is that there is no time limit.

REXXTSO=YES
Specifies that you are adding or replacing a TSO REXX exec check. A TSO check runs in a TSO environment and can use TSO services. See Chapter 8, "Writing REXX checks," on page 169 for more information.

REXXTSO=YES is the default.

REXXTSO=NO
Specifies that you are adding or replacing a non-TSO REXX exec check. A non-TSO check does not run in a TSO environment, and cannot use TSO services. See Chapter 8, "Writing REXX checks," on page 169 for more information.

REXXIN={NO | YES}
Specifies whether or not a non-TSO check requires a sequential input data set.

You can specify REXXIN(YES) only for a non-TSO REXX exec check defined with REXXTSO(NO). See "Using REXXOUT data sets" on page 175 for information on how the system determines the name of your input set for information for a non-TSO check.

REXXIN=NO is the default.

MESSAGETABLE=*msgtablename* **| *NONE**
Required parameter specifies the module name of the message table that will be used when generating messages for the check you are adding or replacing:

- *msgtablename* specifies a name for the message table. The message table must be built using HZSMSGEN. The message table module must be in an APF-authorized library and the system must be able to locate the check routine within the joblib or steplib of the IBM Health Checker for z/OS address space, the LPA, or the LNKLST.
- *NONE indicates that you are adding or replacing a check that has no associated message table, using instead HZSFMSG REQUEST=DIRECTMSG or HZSLFMSG_REQUEST='DIRECTMSG' to issue messages directly from the check routine. See "Issuing messages for your check - message table checks versus DIRECTMSG checks" on page 102.

SEVERITY={HIGH | MEDIUM | LOW}

Required parameter **SEVERITY** defines default check severity for the check you are adding or replacing. The severity you pick determines how the exception messages the check routine issues with the HZSFMSG service are written.

- HIGH indicates that the check routine is checking for high-severity problems. All exception messages that the check issues with the HZSFMSG service will be issued to the console as critical eventual action messages. HZS0003E is issued, which includes message text defined by the check owner.
- MEDIUM indicates that the check routine is looking for medium-severity problems. All exception messages the check issues with HZSFMSG will be issued to the console as eventual action messages. HZS0002E is issued which includes message text defined by the check owner.
- LOW indicates that the check is looking for low-severity problems. All exception messages the check issues with HZSFMSG will be issued to the console as informational messages. HZS0001I is issued which includes message text defined by the check owner.

Note that there are a couple of things that can override the SEVERITY specifications and implicit behaviors derived from SEVERITY:

- The WTOTYPE parameter overrides the implicit WTO type derived from the SEVERITY defined for a check - see"WTOTYPE".
- A check using the dynamic check severity function can override the SEVERITY you define for a check - see “Writing a check with dynamic severity levels” on page 124.

INTERVAL={ONETIME|*hhh*:*mm*}

Required parameter **INTERVAL** specifies how often the check should run:

- **ONETIME** specifies that the check should run only once, kicked off by refresh processing or in alignment to a SYNCVAL value for the check.
- *hhh:mm* provides a specific interval for the check to run. The check will run at refresh time and periodically afterwards at the interval specified. *hhh* indicates the number of hours, from 0 to 999. *mm* indicates the number of minutes, from 0 to 59.

The system starts processing the interval depending on whether you are using the SYNCVAL parameter:

- If you are not using SYNCVAL or using SYNCVAL=SYSTEM, the specified interval time starts ticking away when a check finishes running.
- If you are using SYNCVAL={*hh*:*mm*|*:*mm*}, the specified interval time starts ticking away when the check starts running, to align it with the SYNCVAL start time.

DATE=*date*

Required parameter **DATE** specifies when you define the check you are adding or replacing. The date is specified in the format *yyyymmdd*. If the date specified on the ADDREPLACE parameter is earlier than the original date for the check, the system does not process the values specified on the ADDREPLACE parameter because it may no longer be appropriate for the check. When the date provided on a matching UPDATE, POLICY UPDATE or POLICY DELETE statement is older than this date, that policy statement is not applied to this check.

REASON=*reason*

Required parameter **REASON** specifies what the check routine validates. The value for *reason* can be 1-126 text characters. You can specify these characters as:

- A single value enclosed in single quotes.
- Multiple values, where either each value is enclosed in single quotes or the group of values is enclosed is single quotes. The system will separate these values from each other by a blank. For example, both REASON='r1 r2 r3' and REASON=('r1','r2','r3') will both resolve to a reason of r1 r2 r3.

PARM=*parameter*

PARM specifies the check specific parameters being passed to the check you are adding or replacing. The value for *parameter* can be 1-256 text characters. You can specify these characters as:

- A single value enclosed in single quotes.
- Multiple values, where either each value is enclosed in single quotes or the group of values is enclosed is single quotes. The system will separate these values from each other by a comma. For example, both PARM='p1,p2,p3' and PARM=('p1','p2','p3') will both resolve to a parameter value of p1,p2,p3.

GLOBAL

Specifies that the check you are adding or replacing is global, which means that it runs on one system but reports on sysplex-wide values and practices. If you do not specify GLOBAL, the systems assumes that the check is local, which means that it will run on each system in the sysplex where it is active and enabled.

ACTIVE | INACTIVE

Use the **ACTIVE** and **INACTIVE** parameters to specify the state of the check you are adding or replacing. See "Understanding check state and status" on page 34. These parameters are equivalent to "ACTIVATE parameter" and "DEACTIVATE parameter".

EXCEPTINTERVAL={SYSTEM | HALF | *hhh*:*mm*}

EXCEPTINTERVAL specifies how often the check you are adding or replacing should run after the check has found an exception. This parameter allows you to specify a shorter check interval for checks that have found an exception.

- **SYSTEM**, which is the default, specifies that the EXCEPTINTERVAL is the same as the INTERVAL.
- **HALF** specifies that the exception interval is defined to be half of the normal interval. For example, the exception interval will be set 30 seconds, if the normal interval is set to 1 minute.

  Note that if you change the INTERVAL parameter for a check, the system will also recalculate the exception interval.
- *hhh:mm* provides a specific exception interval for the check. After raising an exception, the check will run at the exception interval specified. *hhh* indicates the number of hours, from 0 to 999. *mm* indicates the number of minutes, from 0 to 59.

The system starts processing the interval depending on whether you are using the SYNCVAL parameter:

- If you are not using SYNCVAL or using SYNCVAL=SYSTEM, the specified interval time starts ticking away when a check finishes running.
- If you are using SYNCVAL={*hh*:*mm*|*:*mm*}, the specified interval time starts ticking away when the check starts running, to align it with the SYNCVAL start time.

USS={NO | YES}

USS specifies whether the check uses z/OS UNIX System Services.

- **NO**, which is the default, specifies that your check does not require z/OS UNIX System Services.
- **YES** specifies that your check requires z/OS UNIX System Services. If you specify USS=YES, the following occurs:
  - IBM Health Checker for z/OS will wait for this check to complete before shutting down z/OS UNIX System Services
  - This check will not run if z/OS UNIX System Services is down.

  To avoid the delay of waiting for z/OS UNIX System Services to shut down and your check not running if z/OS UNIX System Services is not up, do not specify USS=YES unless your check really needs z/OS UNIX System Services.

VERBOSE={NO|YES}

VERBOSE specifies the verbose mode desired for the check you are adding or replacing:

- **NO** specifies that you do not want to run in verbose mode.
- **YES** specifies that you want to run in verbose mode. Running in verbose mode you see additional messages about non-exception conditions. These messages are only issued when the check is in verbose mode.

  Verbose mode does not take effect until you re-run the check or checks. For example, after you issue a command to turn verbose mode on, you could issue the `F hzsproc` command with the RUN parameter to run the check or checks with verbose mode on.

ENTRYCODE=*entrycode*

ENTRYCODE specifies an optional unique check entry value needed when a check routine contains multiple checks. This value is passed to the check routine in the field Pqe_EntryCode in mapping macro HZSPQE.

ALLOWDYNSEV={NO|YES}

Optional parameter **ALLOWDYNSEV** specifies whether this check can issue check exception messages with a dynamically varying severity level.

- ALLOWDYNSEV=NO, which is the default, specifies that the check is not allowed to specify a dynamic severity. The system uses the SEVERITY defined for the check in either the HZSADDCK service or the ADD|ADDREPLACE,CHECK parameter of HZSPRMxx/MODIFY *hzsproc*.
- ALLOWDYNSEV=YES specifies that the check can specify the severity dynamically. A check defined with ALLOWDYNSEV=YES can use the SEVERITY parameter on the HZSFMSG service (or HZSLFMSG_SEVERITY for REXX service HZSLFMSG) to send check exception messages with a dynamic severity.

  The severity specified dynamically on HZSFMSG or HZSLFMSG overrides the severity defined for the check in either the HZSADDCK service or the UPPDATE and ADD|ADDREPLACE,CHECK parameter of HZSPRMxx/MODIFY *hzsproc*.

See “Writing a check with dynamic severity levels” on page 124.

DOM={SYSTEM|CHECK}

Optional parameter DOM specifies whether the check or the system delete the write to operator (WTO) messages from previous check iterations using delete operator message (DOM) requests.

- DOM=SYSTEM, which is the default, indicates that the system issues DOM requests to delete any WTOs from previous check iterations just before the start of a new check iteration.
- DOM=CHECK indicates that the check is expected to issue its own DOM requests to delete previous check exception WTOs using HZSFMSG REQUEST=DOM or HZSLFMSG_REQUEST=DOM, except in the following cases, where the system will still take care of the DOMs:
  - When a health check gets deactivated, deleted, or refreshed
  - When IBM Health Checker for z/OS ends
  - When a WTO was not related to a check exception. For example certain HZSFMSG REQUEST=HZSMSG WTO messages are not sent as check exceptions and will be DOMed by the system.

  Do not add a check with DOM=CHECK unless it is for a check written to DOM its own exception WTO messages using HZSFMSG REQUEST=DOM or HZSLFMSG_REQUEST=DOM. Note that you cannot change the setting of DOM on the UPDATE parameter of HZSPRMxx/MODIFY *hzsproc* interfaces.

For information on how DOM=CHECK and HZSFMSG REQUEST=DOM/ HZSLFMSG_REQUEST=DOM works, see “Controlling check exception message WTOs and their automation consequences” on page 126.

ADD,PARMLIB

```
ADD,PARMLIB=(suffix1...suffixn)
```

Adds one or more HZSPRMxx parmlib member suffixes to the list of suffixes the system uses to obtain check values. The system immediately processes the statements in the added parmlib members. You can specify a list of up to 124 suffixes.

REPLACE,PARMLIB

```
REPLACE,PARMLIB=(suffix1...suffixn)
     [,{CHECKS|POLICY|ALL}]
```

Replaces the list of HZSPRMxx parmlib members with the specified parmlib member suffixes. You can specify a list of up to 124 suffixes.

REPLACE,PARMLIB first deletes applicable statements that had been processed from the current HZSPRMxx parmlib members and then replaces and processes the statements in the parmlib members specified in the list of suffixes. The system then applies the statements to any new checks.

You can use SET,PARMLIB as a synonym for REPLACE,PARMLIB.

CHECKS

Specifies that you want to delete check definitions added with ADD or ADDREPLACE CHECK statements when you issue this command to replace the list of HZSPRMxx parmlib members.

POLICY
Specifies that you want to delete existing POLICY statements and then add those in the parmlib members specified in the REPLACE,PARMLIB command. POLICY is the default.

ALL
Specifies that you want to replace both checks added with ADD | ADDREPLACE,CHECK and existing policy statements with those specified in the REPLACE,PARMLIB command. The system begins by deleting the checks added with ADD|ADDREPLACE CHECK and existing policy statements before replacing them.

ACTIVATE,POLICY

```
ACTIVATE,POLICY=policyname
```

Specifies the 1-16 character name of the policy that you want to activate to make it the current, active policy. The policy stays in effect as the current active policy until you issue another `ACTIVATE,POLICY=policyname` command to activate a different policy.

After you activate a policy, if you want to ensure that only the values from the new policy will be applied to checks, you must refresh all the relevant checks. Until you refresh the checks, the values being applied to checks will still include any values from the previous policy that are not contradicted by the new policy. See "Some finer points of how policy values are applied" on page 56.

{ADD | ADDREPLACE} ,POLICY

```
{ADD | ADDREPLACE}
     ,POLICY[=policyname][,STATEMENT=name],UPDATE,filters[,update_options],REASON=reason,DATE={date|(date,NOCHECK)}
     |
     ,POLICY[=policyname][,STATEMENT=name],DELETE,filters,REASON=reason,DATE={date|(date,NOCHECK)}
```

Add or replace a policy statement in a policy. The check values in the policy or policy statement are applied whenever a check is added or refreshed. The check values on a new or replaced policy statement are applied when that policy statement is added or replaced.

You must specify the "REASON parameter" and "DATE parameter" when you specify **ADD**, **ADDREPLACE,** or **REMOVE**,POLICY.

We recommend that you use the ADD | ADDREPLACE POLICY statements in the HZSPRMxx parmlib member rather than in the MODIFY command, because:

- It is easy to exceed the number of characters allowed for a command with the POLICY statements.
- Changes made in the parmlib member will be applied at each restart of IBM Health Checker for z/OS.

ADD
Add the new policy statement that follows.

ADDREPLACE
Specifies that the system either add or replace the following policy statement, as appropriate. If the policy statement is new, the system will add it. If the policy statement exists already, the system will replace it with the one specified.

POLICY=*policyname*
The 1-16 character name of the policy in which you are adding or replacing policy statements. The policy name can be up to 16 characters long.

If you do not specify a policy name, the system assigns a default name of DEFAULT for your policy. Use the DISPLAY POLICY command to find the name of a policy.

STATEMENTNAME | STATEMENT | STMT =*stmtname*
STATEMENTNAME specifies the name of the policy statement. The statement name can be up to 16 characters long.

If you do not specify the STATEMENTNAME parameter, the system creates a decimal number name for your statement. For example, the system might create a statement name such as 1 or 2 for a statement. The number can be more than one digit.

UPDATE
Create a policy statement that will update the specified check or checks. You must specify the **REASON** and **DATE** parameters. See the "UPDATE parameter".

DELETE
Create a policy statement that will delete the specified check or checks. You must specify the **REASON** and **DATE** parameters.

filters
You must specify filter CHECK=(*check_owner,check_name*) with ADD | ADDREPLACE POLICY. Other filters are optional. See "filters".

REASON=*reason*
See the "REASON parameter".

DATE={*date* **| (***date***,NOCHECK)}**
DATE specifies when the policy statement was created. The date is specified in the format *yyyymmdd*. If the date specified on the policy statement is earlier than the date for the check, the system does not process the values specified on the policy statement because the policy statement may no longer be appropriate for the updated check.

NOCHECK
By default, IBM Health Checker for z/OS verifies the date for the policy statement, making sure that it is equal to or newer than the date for the check that the statement applies to. Use NOCHECK to bypass date verification so that you do not have to update the policy statement date for minor changes to a check.

REMOVE ,POLICY

```
REMOVE,POLICY[=policyname],STATEMENT=name
```

Remove a policy statement. The check values on the policy statements are applied whenever a check is added or refreshed. The check values on a new or replaced policy or policy statement are applied when that policy statement is added or replaced.

REMOVE
Remove the named policy statement.

POLICY=*policyname*
Specifies the 1-16 character name of the policy for which you are removing a policy statement. The policy name can be up to 16 characters long.

If you do not specify a policy name, the system assigns a default name of DEFAULT for your policy. Use the DISPLAY POLICY command to find the name of a policy.

STATEMENTNAME | STATEMENT | STMT =*stmtname*
STATEMENTNAME specifies the name of the policy statement. The statement name can be up to 16 characters long.

You can use wildcard characters in POLICY and STATEMENTNAME. The command is applied to all matching policies and policy statements. For example:

- POLICY=*,STMT=01 would remove all 01 statements from all policies.
- POLICY=POL1,STMT=S* would remove from policy POL1 all statements with names beginning with S.
- POLICY=*,STMT=S* would remove all policy statements starting with S from all policies.

Examples of DISPLAY output

Example of DISPLAY SUMMARY message output: The following output is displayed in response to the `f hzsproc,display,checks` command:

```
HZS0200I 10.31.08 CHECK SUMMARY      FRAME LAST   F      E   SYS=SY1
CHECK OWNER      CHECK NAME                       STATE STATUS
IBMUSS           USS_MAXSOCKETS_MAXFILEPROC        AE   EXCEPTION-LOW
IBMUSS           USS_AUTOMOUNT_DELAY               AD   ENV N/A
IBMUSS           USS_FILESYS_CONFIG                AE   SUCCESSFUL
IBMRACF          RACF_SENSITIVE_RESOURCES          AE + EXCEPTION-HIGH
IBMRACF          RACF_GRS_RNL                      AE + SUCCESSFUL
IBMCNZ           CNZ_SYSCONS_PD_MODE               AE   SUCCESSFUL
IBMCNZ           CNZ_EMCS_INACTIVE_CONSOLES        AEG  SUCCESSFUL
IBMCNZ           CNZ_SYSCONS_ROUTCODE              AE   EXCEPTION-LOW
IBMCNZ           CNZ_SYSCONS_MSCOPE                AE   EXCEPTION-MED
IBMCNZ           CNZ_EMCS_HARDCOPY_MSCOPE          AE   SUCCESSFUL
IBMCNZ           CNZ_CONSOLE_ROUTCODE_11           AE   EXCEPTION-LOW
IBMCNZ           CNZ_AMRF_EVENTUAL_ACTION_MSGS     AE   EXCEPTION-LOW
IBMCNZ           CNZ_CONSOLE_MSCOPE_AND_ROUTCODE   AE   EXCEPTION-LOW
IBMCNZ           CNZ_CONSOLE_MASTERAUTH_CMDSYS     AE   SUCCESSFUL
IBMCNZ           CNZ_TASK_TABLE                    AE   SUCCESSFUL
IBMGRS           GRS_EXIT_PERFORMANCE              AE   SUCCESSFUL
IBMGRS           GRS_CONVERT_RESERVES              AEG  EXCEPTION-LOW
IBMGRS           GRS_SYNCHRES                      AE   SUCCESSFUL
IBMGRS           GRS_MODE                          AEG  SUCCESSFUL
IBMSDUMP         SDUMP_AUTO_ALLOCATION             AE   EXCEPTION-MED
IBMSDUMP         SDUMP_AVAILABLE                   AE   SUCCESSFUL
IBMVSM           VSM_SQA_THRESHOLD                 AE   SUCCESSFUL
IBMVSM           VSM_CSA_LIMIT                     AE   SUCCESSFUL
IBMVSM           VSM_PVT_LIMIT                     AE   SUCCESSFUL
IBMVSM           VSM_SQA_LIMIT                     AE   SUCCESSFUL
IBMVSM           VSM_CSA_THRESHOLD                 AE   SUCCESSFUL
IBMVSM           VSM_CSA_CHANGE                    AE   SUCCESSFUL
IBMRSM           RSM_HVSHARE                       AE   SUCCESSFUL
IBMRSM           RSM_MEMLIMIT                      AE   EXCEPTION-LOW
IBMRSM           RSM_MAXCADS                       AE   SUCCESSFUL
IBMRSM           RSM_RSU                           AE   SUCCESSFUL
IBMRSM           RSM_REAL                          AE   EXCEPTION-LOW
IBMRSM           RSM_AFQ                           AE   SUCCESSFUL
```

```
  A - ACTIVE          I - INACTIVE
  E - ENABLED         D - DISABLED
  G - GLOBAL CHECK    + - ADDITIONAL WARNING MESSAGES ISSUED
```

Example of DISPLAY DETAIL message output: The following output is displayed in response to a `f hzsproc,display,checks,check=(IBMRSM,RSM_MEMLIMIT),detail` command:

```
HZS0201I 09.20.29 CHECK DETAIL
CHECK(IBMRSM,RSM_MEMLIMIT)
 STATE: ACTIVE(ENABLED)             STATUS: EXCEPTION-LOW
 EXITRTN: IARHCADC
 LAST RAN: 05/01/2006 09:14    NEXT SCHEDULED: (NOT SCHEDULED)
 INTERVAL: ONETIME
 EXCEPTION INTERVAL: SYSTEM
 SEVERITY: LOW
 WTOTYPE: INFORMATIONAL
 SYSTEM DESCCODE: 12
 THERE ARE NO PARAMETERS FOR THIS CHECK
 REASON FOR CHECK:  Performance may be impacted
 MODIFIED BY: N/A
 DEFAULT DATE: 20041006
 ORIGIN: HZSADDCK
 LOCALE: HZSPROC
 DEBUG MODE: OFF  VERBOSE MODE: NO
```

Example of DISPLAY DIAG message output: The following output is displayed in response to a `f hzsproc,display,check(IBMGRS,grs_mode),detail,diag` command. The output shows diagnostic information such as the address of the check routine and message table:

```
HZS0201I 09.22.18 CHECK DETAIL
 CHECK(IBMGRS,GRS_MODE)
  STATE: ACTIVE(DISABLED)    GLOBAL  STATUS: SUCCESSFUL
  EXITRTN: ISGHCADC
  LAST RAN: 05/01/2006 09:14    NEXT SCHEDULED: (DISABLED)
  INTERVAL: ONETIME
  EXCEPTION INTERVAL: SYSTEM
  SEVERITY: LOW
  WTOTYPE: INFORMATIONAL
  SYSTEM DESCCODE: 12
  DEFAULT PARAMETERS:       STAR
  REASON FOR CHECK:  GRS should run in STAR mode to improve
                     performance.
  MODIFIED BY: N/A
  DEFAULT DATE: 20050105
  ORIGIN: HZSADDCK
  LOCALE: HZSPROC
  DEBUG MODE: OFF  VERBOSE MODE: NO
  INTERNAL DIAGNOSTICS -  CHECK TOKEN: 01020038.7FD94000
  ROUTINE: ISGHCGRS-7F038300  MSGTBL: ISGHCMSG-7F0343B8  FUNC: DELETE
  LAST CPU TIME: 0.070  MAX CPU TIME: 0.070
```

Example of DISPLAY POLICY SUMMARY message output: The following output is displayed in response to a `f hzsproc,display,policy,stmt=*` command:

```
 HZS0204I 11.03.45 POLICY SUMMARY     FRAME LAST   F      E   SYS=SY1
 STMT            TYPE  CHECK OWNER      CHECK NAME
 GRSMODE_SEVERITY UPD  IBMGRS           GRS_MODE
```

Example of DISPLAY POLICY DETAIL message output: The following output is displayed in response to a `f hzsproc,display,policy,stmt=*,detail` command:

```
HZS0202I 11.04.44 POLICY DETAIL      FRAME LAST   F      E   SYS=SY1
POLICY STMT: GRSMODE_SEVERITY  ORIGIN: HZSPRMOO        DATE: 20050105
 UPDATE CHECK(IBMGRS,GRS_MODE)
 REASON: update check to high severity
 SEVERITY: HIGH
```

Example of DISPLAY STATUS message output: The following output is displayed in response to a `f hzsproc,display,status` or `f hzsproc,display` command:

```
HZS0203I 15.26.41 HZS INFORMATION FRAME 1 F E SYS=SY39
POLICY(*NONE*)
OUTSTANDING EXCEPTIONS: 1
(SEVERITY NONE: 0 LOW: 0 MEDIUM: 1 HIGH: 0)
ELIGIBLE CHECKS: 4 (CURRENTLY RUNNING: 0)
INELIGIBLE CHECKS: 0 DELETED CHECKS: 0
ASID: 0046 LOG STREAM: NOT DEFINED
LOG STREAM WRITES PER HOUR: 2
LOG STREAM AVERAGE BUFFER SIZE: 1825 BYTES
HZSPDATA RECORDS: 0
ORIGINAL PARMLIB SOURCE: PREV
OPTIONS: NONE
```

Part 2. Developing Checks for IBM Health Checker for z/OS

Chapter 5. Planning checks

The **IBM Health Checker for z/OS** is a component of MVS that provides the framework for checking z/OS system and sysplex configuration parameters and the system environment to help determine places where an installation is deviating from suggested settings or where there might be configuration problems. IBM provides a set of check routines in IBM Health Checker for z/OS, but vendors, consultants, and system programmers can add other check routines.

The objective of a check is to identify potential problems before they impact your availability or, in worst cases, cause outages. The output of a check is messages and reports that help an installation analyze the health of a system.

You can use checks to look for things like:

- Changes in configuration values that occur dynamically over the life of an IPL. Checks that look for changes in these values should run periodically to keep the installation aware of changes accruing since the last IPL, to help ensure a cleaner IPL the next time.
- Threshold levels approaching the upper limits, especially those that might occur gradually or insidiously.
- Single points of failure in a configuration.
- Unhealthy combinations of configurations or values that an installation might not think to check.
- Monitoring checks that create reports of collected data.

A check routine does the following:

- Defines the severity of exceptions it finds and suggests a fix for the exception.
- Defines a timer interval for the check.
- May have default values overridden by installation updates.
- Communicates check results by issuing messages to a buffer associated with the check.

The following are examples of situations customers uncovered running IBM Health Checker for z/OS at different times:

- Configuration abnormalities in what was believed to be a stable system.
- Unexpected values on a system. Investigation revealed changes had been correctly made to that system, but not replicated on other systems.
- Default configurations that were never optimized for performance.
- Outdated settings that didn't support all current applications.
- Mismatched naming conventions that could have led to an outage.
- Dynamic changes accruing over the life of the IPL that can cause problems.

Hints for planning your checks:

- Keep in mind that each check should only check for one thing. This will make it much easier for the installation to resolve exceptions that the check finds and override defaults.
- If you are writing a check that will flag a default or common valid configuration setting as an exception, you should:

- Make sure that the HZSADDCHECK exit routine for your check specifies the INACTIVE parameter on the HZSADDCK macro. INACTIVE specifies that the check should not run until the installation changes the state to active. See Chapter 9, “Writing an HZSADDCHECK exit routine,” on page 191 and “HZSADDCK macro — HZS add a check” on page 260.
- Include information in your check output messages about why the check user is getting an exception message for a default or common valid setting.

Look for great information on writing checks in our Redpaper™: There's lots of great experience-basecd information on writing checks in Redpaper *Exploiting the Health Checker for z/OS infrastructure* (REDP-4590-00).

Sample checks: You will find sample checks, including the source code, on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

Identifying potential checks

Look for potential checks in the following areas:

- **System history** can provide an insight to potential checks.
- **Past system outages** or conditions that produced an alert usually indicate a situation that could be detected by an appropriate check.
- **Support call documentation** can reveal common configuration problems and values.
- **Product documentation** may reveal settings that you wish to check in real time.
- **Single and multisystem configuration** situations.

Within those areas, look for check routine candidates from the following:

- Configuration problems or dynamic installation changes, including common initial setup errors and single points of failure.
- Configuration values do not reflect recommended settings. For example, the CNZ_SYSCONS_MSCOPE check ensures that MVS system consoles are defined to have local message scope, which is recommended.
- Defaults that no longer reflect the current recommendations.
- Configuration recommendations that may have changed as a result of new functions introduced.
- Installation values approaching configuration limits

The life-cycle of a check - check terminology

We'll use the following terms throughout this document:

- **Check iteration:** An instance of a check routine that does the check processing and clean up phases of a check routine. Only one iteration of a particular check, identified by the check and owner name, can run at a time. During refresh processing, a check is reset to its first iteration.
- **Check life-cycle:** The life-cycle of a check is one full cycle of a check, from initialization through delete. Then, when a check is added to the system as part of refresh processing, the life of the check starts all over again.
- **Installation updates:** The installation can update or override some of the default check values you define in the check definition using:
  - SDSF
  - The MODIFY command

– Policy statements in the HZSPRMxx parmlib member

The installation might update some check values to make the check more suitable for their environment or configuration. See Chapter 4, “Managing checks,” on page 43.

- **Refresh process:** Refresh processing first deletes one or more checks from the IBM Health Checker for z/OS and then add the same checks back to the system. The system does the following for each refreshed check:
  – Applies any installation updates to the default values for the check.
  – Clears the 2K work area (PQEChkWork)
  – Resets the check's iteration count to one.
  – Starts the initialization phase for the check, if the check is defined as active.

  For a local check , you can have multiple checks in a single check routine. When you refresh some, but not all, of the checks in a check routine, the system does refresh processing only for the specified checks.

  Refresh processing is kicked off in response to:
  – Refresh request (E action character) from the SDSF CK panel. See “Using SDSF to manage checks” on page 45.
  – The MODIFY (F) *hcproc*,REFRESH operator command. See “Using HZSPRMxx and MODIFY *hzsproc* command” on page 65.

What kind of check do you want to write?

You can develop the following basic types of checks:

- “Local checks”
- “Remote checks” on page 98
- “REXX checks” on page 100

To issue messages from your check, see “Issuing messages for your check - message table checks versus DIRECTMSG checks” on page 102.

Local checks

Local checks run in the IBM Health Checker for z/OS address space, *hzsproc*. You can write local checks in either Metal C or assembler. See “Writing local and remote checks in Metal C” on page 99.

Because the local check runs in the IBM Health Checker address space, writing a local check is simpler than a remote check, but the data you can access might be a little more limited than you can access from a remote check running in the caller's address space. Make sure that your check can access the data it needs from the IBM Health Checker for z/OS address space and that it does not require any potentially disruptive actions, such as I/O intensive operations, serialization, or waits. This is important because if your check hangs in the *hzsproc* address space, it can affect the performance of IBM Health Checker for z/OS and all the other checks.

Local checks must be APF authorized.

The following figure shows the parts of a **local** check. The shaded items show the parts that a check developer must provide:

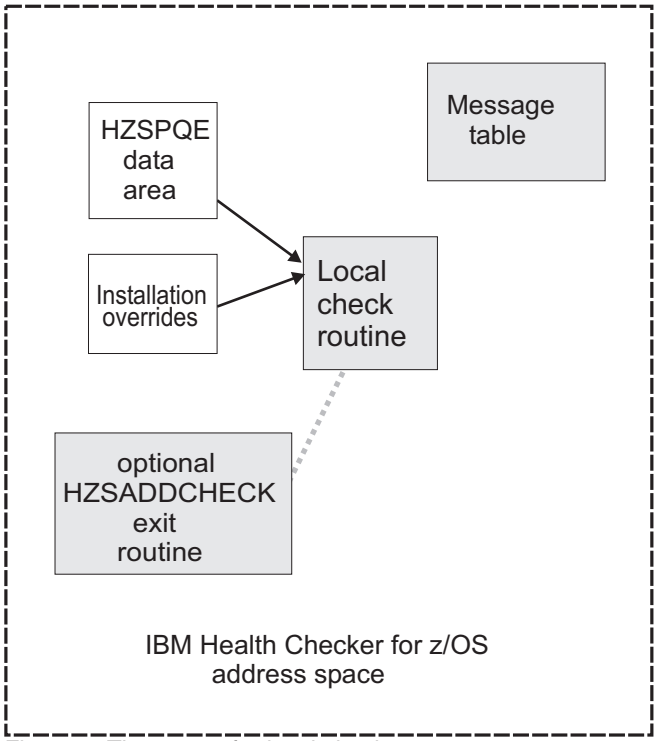


Figure 5. The parts of a local check

Remote checks

Remote checks run as tasks in the address space of the caller. For example, a remote check might run in a server address space so that it can more easily obtain the necessary data about the server space and also more easily read data from data sets.

You can write remote checks in either Metal C or assembler. See "Writing local and remote checks in Metal C" on page 99.

You should write a remote check when:

- You need a check to run in a specific address space, or you cannot access the data you need for your check from the IBM Health Checker for z/OS address space
- Your check requires potentially disruptive actions, such as I/O intensive operations, serialization, or waits. This is important because if your check hangs in the *hzsproc* address space, it can affect the performance of IBM Health Checker for z/OS and all the other checks.

Local and remote check routines share a basic structure, but there are enough differences between them that you'll need to know before you start writing whether you are writing a local or a remote check. A remote check requires synchronization and communication between the remote check routine and IBM Health Checker for z/OS.

IBM Health Checker for z/OS tracks remote checks for you. If the caller's address space where the remote check is running goes down, IBM Health Checker for z/OS treats the check as if it had been deleted. If the IBM Health Checker for

z/OS address space terminates, upon restart it restarts any remote checks that were defined to the system when the address space terminated, unless they have been explicitly deleted.

A remote check need not be APF authorized, but, if not, must be permitted by RACF or other security product.

The following figure shows the parts of a **remote** check. The shaded items show the parts that a check developer must provide:

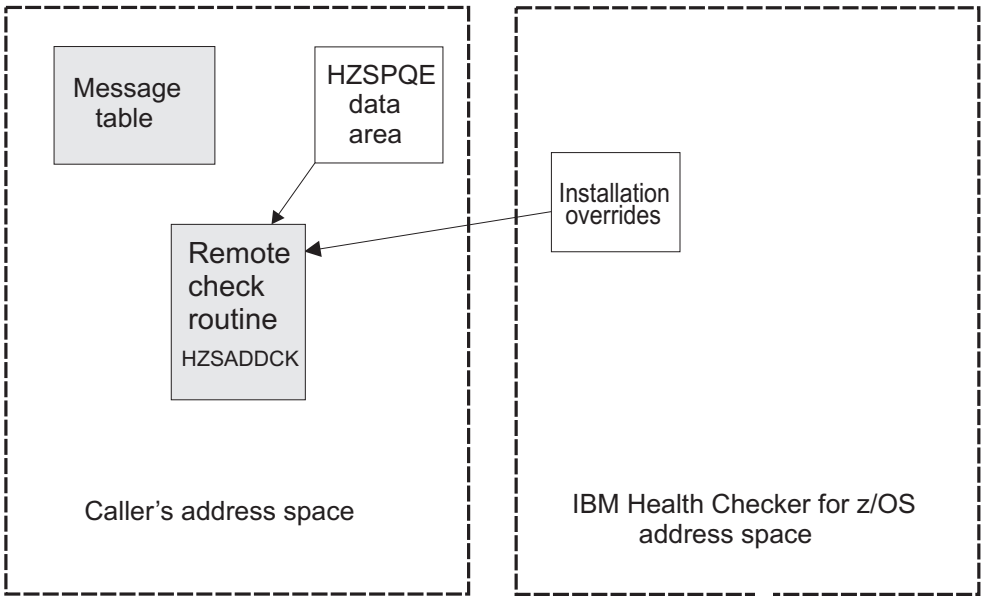


Figure 6. The parts of a remote check

Writing local and remote checks in Metal C

You can write either local or remote checks in Metal C. Metal C programming uses features provided by the XL C compiler as a high level language (HLL) alternative to writing the program in assembly language. Using Metal C, you can:

- Generate code that is independent of the Language Environment®.
- Generate code that follows MVS linkage conventions, making it easy to integrate with the existing code base, such as the IBM Health Checker for z/OS framework.
- Provide support for accessing the data stored in data spaces.
- Provide support for embedding assembly statements, including assembler service calls, into the compiler-generated code, using the __asm keyword.

See *z/OS Metal C Programming Guide and Reference*.

Besides the existing Metal C header files in the runtime library in z/OS UNIX file system directory /usr/include/metal/, IBM Health Checker for z/OS provides generic C header files (hzsh*.h) containing:

- Mappings of IBM Health Checker for z/OS structures and control blocks
- Commonly used Health Checker related constants

These header files are contained in SYS1.SIEAHDRV.H. See Chapter 6, "Writing local check routines," on page 105 or Chapter 7, "Writing remote check routines," on page 135 for more information about the header files and using Metal C to write checks.

You can find IBM Health Checker for z/OS related Metal C samples in z/OS UNIX file system directory /usr/lpp/bcp/samples or on the Web at:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

REXX checks

A REXX check runs in a System REXX address space in an APF authorized environment defined by System REXX. You identify it as a REXX check on the REXX(YES) parameter when defining the check to the system.

A REXX check makes it easy to issue system commands (using the AXRCMD function) and to analyze the output of commands issued. REXX also makes it easy to read data sets or to issue system commands, and parse the retrieved information

You can run System REXX checks in TSO and non-TSO environments.

See the following for information about REXX:

- System REXX in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about the AXRCMD function and coding REXX execs in TSO and non-TSO environments.
- *z/OS TSO/E REXX Reference*
- *z/OS Using REXX and z/OS UNIX System Services*

The following figure shows the parts of a **REXX** check. The shaded items show the parts that a check developer must provide:

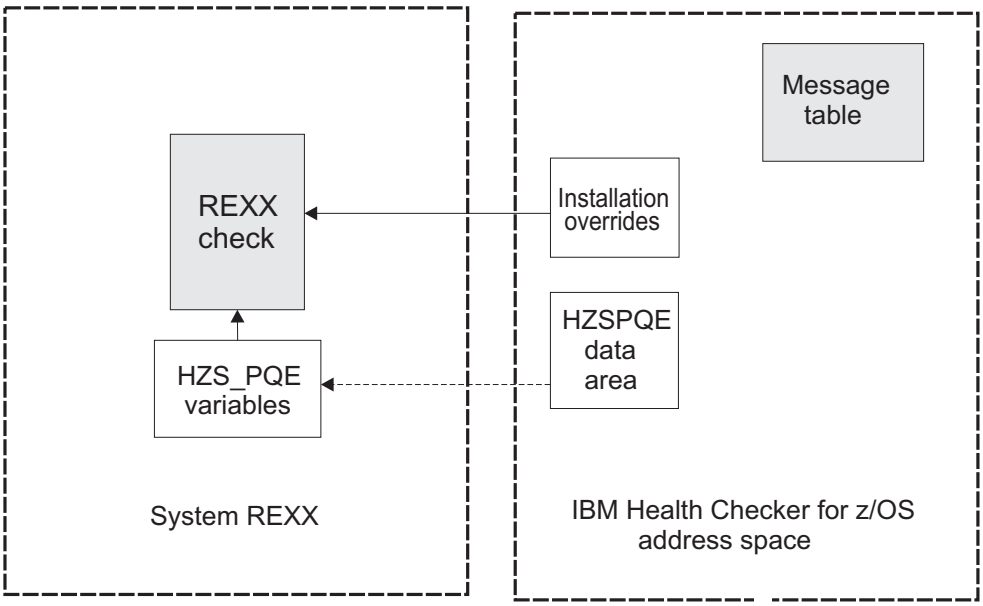


Figure 7. The parts of a REXX check

Summary of checks - differences and similarities

The following table shows some of the differences between local and remote checks:

Table 8. Summary of local, remote, and REXX checks

Local checks	Remote checks	REXX checks
How do I know which type of check to write?		

Table 8. Summary of local, remote, and REXX checks (continued)

Local checks	Remote checks	REXX checks
Write a local check in assembler or Metal C to look at system storage and use assembler services. (You can issue assembler services in either assembler or Metal C.) A local check runs in the IBM Health Checker for z/OS address space, so make sure your check can access the data it needs from there, and does not require a potentially disruptive action, such as a wait. (If your check hangs in the *hzsproc* address space, it can affect the performance of IBM Health Checker for z/OS and all the other checks.) A local check has the advantage of receiving IBM Health Checker for z/OS recovery support.	Write a remote check in assembler or Metal C to look at system storage and use assembler services. (You can issue assembler services in either assembler or Metal C.) A remote check runs in the caller's address space, and is a good choice if: • Your check needs to access data that is hard to reach from the IBM Health Checker for z/OS address space. • Your check requires potentially disruptive actions, such as I/O intensive operations, serialization, or waits.	Write a REXX check to take advantage of the ease in issuing system commands (using the AXRCMD function) and analyzing the output of commands issued. It is easy in a System REXX check to handle I/O. For example, from a REXX check, it is very easy to parse parameters into multiple variables and to read from and write to REXXIN and REXXOUT data sets.
What languages are supported?		
Metal C and assembler	Metal C and assembler	REXX exec
Where does the check run?		
In the IBM Health Checker for z/OS address space.	In the caller's address space.	In a System REXX address space.
What kind of recovery support does the system provide for my check?		
If the check routine abends, the system handles the abend and continues trying to call the check on subsequent iterations.	If the check routine abends, it is up to the application to provide recovery to handle the abend. If the task that issues the HZSADDCK macro defining the check terminates for any reason, including an abend that is not re-tried, the system treats the check as if it is deleted.	If the REXX check abends, the system will mark the check as no longer running for that iteration.
How do I define a check?		
Do one of the following: • For testing purposes, define the check defaults in HZSPRMxx using `ADD|ADDREPLACE CHECK`. • Create a separate HZSADDCHECK exit routine that issues the HZSADDCK service to describe check defaults. You must then add the check to IBM Health Checker for z/OS by adding the exit routine to the HZSADDCHECK exit.	Check routine defines itself by issuing the HZSADDCK macro describing check defaults.	Do one of the following: • Define the check defaults in HZSPRMxx using `ADD|ADDREPLACE CHECK`. • You can also create a separate assembler HZSADDCHECK exit routine that issues the HZSADDCK service to describe check defaults. You must then add the check to IBM Health Checker for z/OS by adding the exit routine to the HZSADDCHECK exit.
Multiple checks per check routine supported?		
Yes - Consolidation of multiple checks in one check routine for a product or element supported and recommended.	Yes - Consolidation of multiple checks in one check routine is supported. Since you must manage the storage for remote checks, whether or not there is any benefit to grouping checks into a single routine depends on how you manage the storage. Note that each remote check, even when grouped in one check routine, must run in a separate task.	Yes - Consolidation of multiple checks in one check exec is supported.
What prompts the processing phase for the check routine?		

Table 8. Summary of local, remote, and REXX checks (continued)

Local checks	Remote checks	REXX checks
Function codes in the HZSPQE data area for local check routines.	Release codes from the IEAVPSE service for remote check routines.	The check invokes HZSLSTRT to initialize the check environment, and HZSLSTOP to indicate that check processing is complete.
Need to synchronize the check routine and the system?		
No - see Chapter 6, "Writing local check routines," on page 105	Yes - see Chapter 7, "Writing remote check routines," on page 135	No - see Chapter 8, "Writing REXX checks," on page 169
Who loads the message table module for the check into storage?		
IBM Health Checker for z/OS	The remote check routine	IBM Health Checker for z/OS
Can I read and write persistent data to the HZSPDATA data set?		
Yes, all the check types can use the HZSPDATA data set.		
How many checks can I run at a time?		
The system can process 20 checks at a time. Processing a check can mean either: • Running a local check • Starting a remote check Once a remote check has been started, it can run on its own and is not subject to the limitation of 20. Thus there is no intrinsic limit to the number of remote checks that can be run at a time.		See System REXX in *z/OS MVS Programming: Authorized Assembler Services Guide*.
How does delete processing work?		
When a check is deleted through either refresh processing or when a user deletes the check, your check will come back to run again whenever ADDNEW or refresh processing occurs, unless you use the DELETE parameter in the active policy.	When a check is deleted, through either refresh processing or when a user deletes the check, the check is marked as deleted and does not come back at subsequent refresh or ADDNEW processing. The delete request is passed to the remote check in a release code and depending on the type of delete release code, the check routine can re-define itself.	A REXX check is not called for delete processing. When a check is deleted through either refresh processing or when a user deletes the check, your check will come back to run again whenever ADDNEW or refresh processing occurs, unless you use the DELETE parameter in the active policy.

Issuing messages for your check - message table checks versus DIRECTMSG checks

The messages that your check issues are the most important part of your check, because they report the results found by the check. For all of the check types, there are two ways to issue check messages:

- Create a separate message table module that defines the check output messages issued by the check routine. A message table has the advantage of allowing you to create much more complete messages. Using a message table also makes it easier to create consistent and maintainable messages and includes formatting support. See "Issuing messages in your local check routine with the HZSFMSG macro" on page 117 for additional information on planning check messages in a message table.
- Issue messages directly from the check routine. You define the message text right in the message request, as follows:
  - For local or remote checks, use HZSFMSG REQUEST=DIRECTMSG, providing the message text in the HZSLFMSG REQUEST=DIRECTMSG request. See "HZSFMSG macro — Issue a formatted check message" on page 307.
  - For REXX checks, use HZSLFMSG_REQUEST='DIRECTMSG', providing the message text right in the HZSLFMSG_REQUEST='DIRECTMSG' input variables. See "Input variables for HZSLFMSG_REQUEST='DIRECTMSG'" on page 242.

When it is time to define a DIRECTMSG check to the system, you must do so using the MESSAGETABLE=*NONE subparameter on the `MODIFY hzsproc` command or in an HZSPRMxx parmlib member on the ADD | ADDREPLACE parameter. See "Using HZSPRMxx and MODIFY *hzsproc* command" on page 65.

Where to next? A road map for developing your check

To create a IBM Health Checker for z/OS check for your component or product, you must do the following:

1. Write a check routine that gathers information, compares current values with suggested settings or looks for configuration problems, and issues messages with the results of the check.
   - For a local check, see Chapter 6, "Writing local check routines," on page 105.
   - For a remote check, see Chapter 7, "Writing remote check routines," on page 135.
   - If you are writing a REXX check, see Chapter 8, "Writing REXX checks," on page 169.
2. Create messages for the check. You can do this in two ways:
   - Create a separate message table module that defines the check output messages issued by the check routine. See "Issuing messages in your local check routine with the HZSFMSG macro" on page 117.
   - Create DIRECTMSG messages right in the check routine in one of the following ways:
     - For local or remote checks, use HZSFMSG REQUEST=DIRECTMSG. See "HZSFMSG macro — Issue a formatted check message" on page 307.
     - For REXX checks, use HZSLFMSG_DIRECTMSG_. See "Input variables for HZSLFMSG_REQUEST='DIRECTMSG'" on page 242.
3. Provide documentation about check-specific installation overrides to allow the installation to override the default check values defined when the check was added. See Chapter 13, "IBM Health Checker for z/OS checks," on page 381.

Chapter 6. Writing local check routines

A local check runs in the IBM Health Checker for z/OS address space, *hzsproc*. You can write a local check in either Metal C or assembler.

To learn about the differences between local and remote checks and deciding which type you want to write, see "Remote checks" on page 98.

In this chapter, we'll cover the following:

- "Metal C or assembler?"
- "Sample local checks" on page 106
- "Local check routine basics" on page 106
- "Defining a local check to IBM Health Checker for z/OS" on page 108
- "Programming considerations" on page 109
- "Using the check parameter parsing service (HZSCPARS)" on page 111
- "Using the HZSPQE data area in your local check routine" on page 111
- "Function codes for local check routines" on page 113
- "Creating and using data saved between restarts" on page 115
- "Issuing messages in your local check routine with the HZSFMSG macro" on page 117
- "Writing a check with dynamic severity levels" on page 124
- "Controlling check exception message WTOs and their automation consequences" on page 126
- "Defining the variables for your messages" on page 120
- "The well-behaved local check routine - recommendations and recovery considerations" on page 127
- "Building Metal C checks" on page 130
- "Debugging checks" on page 132
- Chapter 9, "Writing an HZSADDCHECK exit routine," on page 191

Metal C or assembler?

As mentioned above, you can write a local or remote check in either Metal C or assembler. The concepts in this section apply to either language.

Metal C lets you create a IBM Health Checker for z/OS compatible load module that follows MVS linkage conventions. You can also easily use assembler macros, such as HZSFMSG, HZSCPARS, or any other assembler macro, in your Metal C check routine using the __asm keyword.

If you are writing in Metal C, IBM Health Checker for z/OS provides generic C header files (hzsh*.h) in SYS1.SIEAHDRV.H containing the following mappings of IBM Health Checker for z/OS structures and control blocks and commonly used Health Checker related constants:

Table 9. Correlation between IBM Health Checker for z/OS mapping macros and Metal C header files

Assembler mapping macros in SYS1.MACLIB	Description	Metal C header file in SYS1.SIEAHDRV.H
HZSPQE	Individual check data area	HZSHPQE
HZSDPQE	Individual deleted check data area	HZSHDPQE
HZSMGB	Message data area	HZSHMGB
HZSQUAA	HZSQUERY return information data area	HZSHQUAA
HZSZCPAR	Check parameter area	HZSHCPAR
HZSZENF	Event data for ENF 67	HZSHENF
HZSZHCKL	Message buffer log entry	HZSHHCKL
HZSZCONS	Check return and reason codes	HZSHCONS
	Common shared types	HZSHTYPE
	Include that pulls in all the other header files above	HZSH

For detailed information about Metal C, see *z/OS Metal C Programming Guide and Reference*. You will also want to use the sample checks in "Sample local checks."

Sample local checks

Of course you're going to read this entire chapter to understand everything you need to know about writing a local check routine. But we also have what you're really looking for - samples:

- Metal C samples in z/OS UNIX file system directory /usr/lpp/bcp/samples:
  - **hzscchkr.c** - Sample Metal C local check routine demonstrating how to issue check messages.
  - **hzscchkp.c** - Sample Metal C local check demonstrating how to handle parameters and use persistent data.
  - **hzscadd.c** - Sample Metal C HZSADDCHECK exit routine.
  - **hzssmake.mk** - Sample Metal C makefile to build sample health checks.
- Assembler samples in SYS1.SAMPLIB:
  - **HZSSADCK** - Sample assembler HZSADDCHECK exit routine.
  - **HZSSCHKP** - Sample assembler local check routine.
  - **HZSSCHKR** - Sample assembler local check routine.
  - **HZSSMSGT** - Sample message input.

You'll find all these and more check samples on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

Local check routine basics

A check routine is a program that gathers installation information and looks for problems, and then issues the check results in messages. IBM Health Checker for z/OS writes the check exception messages as WTOs or to the message buffer. The check routine runs in the IBM Health Checker for z/OS address space, which has superuser authority.

When IBM Health Checker for z/OS calls the check routine, register 1 points to a parameter list containing the address of the HZSPQE data area for the check (as well as the address of the 4K dynamic work area). For a Metal C check, use the HZSHPQE header contained in SYS1.SIEAHDRV.H, which mirrors the HZSPQE data area. The HZSPQE data area for a check contains:

- The defaults defined for the check.
- A 2K check work area, a pointer to a 4K dynamic work area, and a pointer to a 2GB - 4K dataspace for use by the check.
- A function code indicating why the check was called.
- Any installation update values.

The check routine should not update the HZSPQE data area except for the 2K check work area. See "Using the HZSPQE data area in your local check routine" on page 111.

We recommend that you keep the check routine very simple. At a high level, **your check will consist of:**

1. Reentrant entry and exit linkage ("Assembler reentrant entry and exit linkage" on page 110) and other setup.
2. Handling of input parameters, if any, for your check when the system indicates that parameter data has changed. See "Using the check parameter parsing service (HZSCPARS)" on page 111.
3. The meat of the check - checking for potential problems on a system.
4. Issuing messages using the HZSFMSG macro ("Issuing messages in your local check routine with the HZSFMSG macro" on page 117)
5. Defining your message variables in the HZSMGB data area ("Defining the variables for your messages" on page 120)

Limit a check to looking at one setting or one potential problem. Limiting the scope of a check will make it easier for the installation using the check to:

- Resolve any exceptions that the check finds by either fixing the exception, overriding the setting, or deactivating the check.
- Set appropriate override values for check defaults such as severity or interval.

Do not set return and reason codes for your check routine. The system will return a result for you in the PQE_Result field when you use HZSFMSG REQUEST=CHECKMSG macro request (for exception messages) or the HZSFMSG REQUEST=STOP macro request (to stop the check). Do not set this field in your check routine.

Use the 2K check work area: Use the 2K check work area in field PQEChkWork for data you want to retain through check iterations for the life of the check, until the check is refreshed or deleted. Using the 2K check work area allows you to avoid obtaining additional resources for your check routine. Prior to the Init function code call, the system sets the 2K work area to zeros.

Use the 4K dynamic work area: Use the 4K dynamic work area for data you want to last for only one function code call. The check routine can find the address of the 4K dynamic work area in:

- Register 0 on entry to the check routine.
- The second word of the parameter list pointed to by Register 1
- Field PQE_DynamicAreaAddr in the HZSPQE data area

Using the 4K dynamic work area allows you to avoid obtaining additional resources for your check routine. However, you cannot rely on the contents of this area being set to a specific value **between** check function calls or check iterations.

Use the 2GB - 4K dataspace: Use the dataspace for data you want to last for only one function code call. The check routine can find the address of the dataspace in field PQE_DataspaceALET in the HZSPQE data area. Our sample check HZSSCHKP demonstrates how to use this dataspace and release the dataspace pages at the end of the check iteration. See “Sample local checks” on page 106.

If you do obtain additional resources for your check routine besides the 2K and 4K work area, the storage must be either:

- Obtained and freed in the same function code processing.
- Owned by the jobstep task and freed no later than PQE_Function_Code_Delete processing.

For complete information on managing virtual storage in a program, see *z/OS MVS Programming: Authorized Assembler Services Guide*

The PQEChkWork field should be the only field your check routine writes to in the HZSPQE data area. The check routine can write to the 2K PQEChkWork field in the HZSPQE data area, and the system saves the entire area for subsequent calls to the check routine. The system clears the 2K PQEChkWork user area before calling the check with the PQE_Function_Code_Init function code. Changes made to any other HZSPQE fields are not saved between function codes.

You can also, of course, write to the 4K dynamic work area pointed to by field PQE_DynamicAreaAddr.

Group checks for a single element or product in a single check routine. You can group multiple uniquely named checks for a single element or product in a single check routine. This can help to reduce system overhead and simplify maintenance. If you are using an HZSADDCHECK exit routine to add your local checks to the system, you should also use a single exit routine to add related checks to the system. Code your check routine to look for the entry code passed in field PQE_Entry_Code, (from the ENTRYCODE parameter on the HZSADDCK call or HZSPRMxx parmlib member) and pass control to processing for the check indicated. Note that the IBM Health Checker for z/OS will not verify the uniqueness of the entry codes you define for your checks.

When you group local checks together in a single check routine, each check still gets its own HZSPQE data area. Checks cannot communicate with each other.

Do not attempt to communicate between individual checks. Even though you may have placed all of your check routines in the same module, do not rely on communication between them. Each check is intended to stand by itself.

Defining a local check to IBM Health Checker for z/OS

Starting with z/OS V1R8, there are two ways to add your local check to the system:

- After you've written your check, use the ADD | ADDREPLACE CHECK parameter in an HZSPRMxx parameter to define check defaults and add the check. See "ADD or ADDREPLACE CHECK parameters".

- Write an authorized HZSADDCHECK exit routine running in the IBM Health Checker for z/OS address space, as described in Chapter 9, “Writing an HZSADDCHECK exit routine,” on page 191. The HZSADDCHECK exit routine describes the information about your local check or checks. The HZSADDCHECK exit routine invokes the HZSADDCK macro to:
  - Identify the check, providing values such as the check owner, check name, check routine name, and message table name.
  - Specifies the default values for the check, such as the check interval, check parameter, and check severity.

Programming considerations

Environment

IBM Health Checker for z/OS calls the check routine in primary mode from the IBM Health Checker for z/OS address space.

- **Minimum authorization:** Authorized
- **Address space:** IBM Health Checker for z/OS
- **State:** Supervisor
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=SASN=HASN
- **AMODE:** 31
- **ASC mode:** Primary
- **Key:** System defined. The system will choose a key for a check and use it for all function code calls to the check routine. The key will match the key in field TCBPKF.
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Control parameters:** Control parameters are in the IBM Health Checker for z/OS address space

Requirements

- Many installations are multilevel secure systems, and check developers must be aware of the multilevel system environment..
- The check routine must be able to handle the IBM Health Checker for z/OS function or release codes:
  - “Function codes for local check routines” on page 113
  - “Release codes for remote check routines” on page 145

  See “The well-behaved local check routine - recommendations and recovery considerations” on page 127.
- The check routine load module and message table must reside in an APF-authorized library. The system will treat them as reentrant.

Restrictions

None

Gotchas

- If your check routine gets an abend X'290', reason code `xxxx4007`, it could mean that the routine is not in an APF authorized library.

- The check routine is reentrant, so your check routine must use the LIST and EXECUTE forms of the any z/OS macros with parameter lists, including the HZS macros.

Input Registers

When a check receives control the contents of the registers are as follows:

Register
Contents

Register 0
Address of the 4K dynamic work area

Register 1
Address of an 8 byte parameter list containing:
- The 4 byte address of the HZSPQE for the check
- The 4 byte address of the 4K dynamic work area

Register 13
Address of a 144 byte save area

Register 14
Return address

Register 15
Address of the check routine

Output Registers

When a check returns control, the contents of the registers are as follows:

Register
Contents

Register 0 - 15
The check routine does not have to place any information in this register, and does not have to restore its contents to what they were when the exit routine received control

Establishing a recovery routine for a check

Establishing an ESTAEX or IEAARR routine in the check routine will provide recovery from errors encountered during check execution. See Writing recovery routines in *z/OS MVS Programming: Assembler Services Guide*.

The check routine continues to be invoked at the interval defined unless three consecutive calls fail, in which case the check is placed in a disabled state.

See “The well-behaved local check routine - recommendations and recovery considerations” on page 127.

Assembler reentrant entry and exit linkage

To see an example of how to code the reentrant entry and exit linkage for a local assembler check routine, see sample check HZSSCHKR in SYS1.SAMPLIB or on the Web at:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

For more information about entry and exit linkage, see the following topics in *z/OS MVS Programming: Assembler Services Guide*:

- Linkage Conventions
- Example of Using the Linkage Stack

Note that for Metal C checks, you do not have to specify this assembler linkage. All you need to do is specify compiler option RENT - See "Building Metal C checks" on page 130.

Using the check parameter parsing service (HZSCPARS)

If your local or remote check includes parameters, you can use the HZSCPARS check parameter parsing service to parse parameters. You can use HZSCPARS in either an assembler or Metal C check routine. When HZSCPARS finds a parameter error, it issues appropriate error messages for you using the REASON=PARS*xxxx* reason values on the HZSFMSG macro. This means that your check routine does not have to issue error messages for parameter errors. See "HZSFMSG macro — Issue a formatted check message" on page 307 for explanations of all the REASON=PARS*xxxx* values.

If you are using HZSCPARS for a check that expects a parameter or parameters but does not get one, HZSCPARS considers this an error and issues an error message.

Your check routine can also use REASON=PARS*xxxx* on HZSFMSG REQUEST=HZSMSG to issue parsing error messages in the course of doing their own parameter parsing.

You will use HZSCPARS REQUEST=PARSE in your check routine to allocate a parameter area, mapped by mapping macro HZSZCPAR, that describes the parsed parameters for the check. You can free this parameter area using HZSCPARS REQUEST=FREE . For a local check, if you do not free the parameter area, the system will delete the parameter area upon return from the check routine.

See "HZSCPARS macro — HZS Check Parameter Parsing" on page 294 for complete information.

For an example of using the HZSCPARS macro in a check routine, see sample HZSSRCHC, which you can fine in SYS1.SAMPLIB or on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

Note that your check routine must still issue the HZSFMSG REQUEST=STOP request when HZSCPARS finds a parameter error - see "Function codes for local check routines" on page 113 and "Release codes for remote check routines" on page 145.

Using the HZSPQE data area in your local check routine

The HZSPQE data area contains all the information a check routine needs, including the defaults defined in the HZSADDCHECK exit routine r and any installation overrides to those defaults. As we mentioned above, for Metal C checks, the IBM Health Checker for z/OS header file HZSHPQE mirrors the HZSPQE data area mapping.

The HZSPQE contains a number of sections, but some of the most important are:

- The PQE_DynamicAreaAddr, which contains the address of the 4K dynamic user area.
- PQEChkParms, which shows the current values for the check.
- PQEChkWork, which is the 2K check work area.

The table below shows the structure and some of the most important fields in the HZSPQE data area.

Table 10. Important fields in the HZSPQE data area for a local check routine

Field name	Meaning
PQEHeader section - contains general control block information.	
PQE_DynamicAreaAddr	The address of a 4K dynamic work area. The system does not clear this work area before or after a function code call. Use the 4K dynamic work area for data you want to last for only one function code call. You cannot rely on the contents of this area being set to a specific value **between** check function calls or check iterations. This field does not apply to remote checks.
PQEStatus section - contains status information about the check.	
PQE_ CleanupInDifferentTaskThanCheck	This bit indicates that the cleanup function is executing under a different task than the check function. If this bit is on, and the task that ran the check function obtained a resource owned by the current task, the local check routine does not need to use the cleanup function to free the resource. See "Function codes for local check routines" on page 113. This bit applies only to local checks.
PQE_Function_Code	This field indicates the function code for the check. The check routine receives control in response to one of the following function codes: PQE_Function_Code_Init, PQE_Function_Code_Check, PQE_Function_Code_Cleanup, or PQE_Function_Code_Delete. This bit applies only to local checks. Release code information for remote checks is mapped by the HZSZCONS mapping macro - see "Release codes for remote check routines" on page 145.
PQE_DataspaceALET	The ALET of a data space on the DU-AL that the check routine can use for any purpose. The check routine must not assume that any of the storage is 0. The check can use all the storage in the range 1000-x'7FFEFFF'. This field is only valid for local checks (non-REXX).
PQE_DataspaceSTOKEN	The STOKEN of the data space addressed by PQE_DataspaceALET. If the check routine uses more than two pages of data space storage it should issue DSPSERV RELEASE using this STOKEN and the used range upon completion of the check function (or in the cleanup function). This field is only valid for local checks (non-REXX).
PqeChkInfo section - contains the defaults defined in the HZSADDCHECK exit routine r for the check	
PQE_Entry_Code	Contains the identifier (entry code) assigned for the check in the HZSADDCHECK exit routine r. The entry code is used when a check routine contains multiple checks.
PqeChkParms section - contains the installation overrides for default parameters for the check from HZSPRMxx and the Modify command (F *hzsproc*).	
PQE_LookAtParms	A bit indicating that the parameters have changed. If this bit is on, the check routine should read the PQE_ParmArea and PQE_PARMLen fields in PQE_Function_Code_Check processing.
PQE_Verbose	A byte indicating whether the check is in verbose mode.
PQE_Debug	A byte indicating whether the check is in debug mode.
PQE_ParmLen	Contains the length of the parameter area. Quotes surrounding the PARMS value in an operator command or HZSPRMxx statement are not included in the resulting length. For example, PARMS('THE_PARM') will result in a length of 8.
PQE_ParmArea	The area containing the user parameters. Quotes surrounding the PARMS value in an operator command or HZSPRMxx statement are not included.

Table 10. Important fields in the HZSPQE data area for a local check routine (continued)

Field name	Meaning
PQEChkWork section - 2K check work area used and mapped by the check routine as needed. The system zeros the 2K user PQEChkWork user area before calling the check with function code PQE_Function_Code_Init. A check routine can both write and read from this field, and the system will retain this information for subsequent calls to the check routine. Changes made to any other HZSPQE fields are not saved between function calls.	

Function codes for local check routines

IBM Health Checker for z/OS invokes a local check routine with a function code to indicate why it was called. All the function code calls will run under the same jobstep task, but you **cannot assume** that any of these function codes will run in the same task as a preceding function.

In general:

- PQE_Function_Code_Init (Init function) is called once for the life of the check (which lasts until the check is deleted).
- PQE_Function_Code_Check (Check function) is called at the specified interval for the check
- PQE_Function_Code_Cleanup (Cleanup function) is called right after the Check function
- PQE_Function_Code_Delete (Delete function) is called once at the end of the life of the check.

The following table summarizes the function codes provided by IBM Health Checker for z/OS, showing what the check should do for each PQE_Function_Code_ and when IBM Health Checker for z/OS invokes it:

Table 11. Summary of function codes for local checks

Function	Check and system actions	When is it invoked?
Init	**What should the check do?** For PQE_Function_Code_Init, the check routine should validate that the environment is suitable for the check. If it is not, issue the HZSFMSG REQUEST=STOP macro to stop the check. If you obtain additional storage for the check, obtain it in Init processing and obtain it in jobstep-task owned storage. (You cannot assume that each function code runs under the same task.)	• Refresh • When a check is added • When a check transitions to the active enabled state
	What does the system do? The system does the following setup steps to prepare for multiple check iteration: • Initializes the HZSPQE data area with default and override values for the check. • Passes the default and installation overrides to the check in the HZSPQE data area for the check. • Obtains 2K of workarea storage mapped by field PQEChkWork. This storage is zeroed for Init processing and lasts for the life of the check. • Obtains 4K of dynamic work area pointed to by field PQE_DynamicAreaAddr. The contents of this work area are not set to any particular value and are not preserved across check iterations.	

Table 11. Summary of function codes for local checks (continued)

Function	Check and system actions	When is it invoked?
Check	**What should the check do?** For PQE_Function_Code_Check, the check routine should: 1. Check to see if the PQE_LookatParm bit is set on, indicating either that this is the first iteration of the check, or that the installation has changed the check parameters since the last iteration. If the bit is on, validate the parameters in the PQE_UserParmArea of the HZSPQE data area. If the check finds bad installation parameters, it should: a. Issue an error message indicating what the problem is. b. Issue the HZSFMSG REQUEST=STOP,REASON=BADPARM macro request to stop the check. See "HZSFMSG macro — Issue a formatted check message" on page 307. 2. Check for the setting or potential problem it was designed to report on. 3. Report check results using the HZSFMSG service to issue exception messages, reports, and other messages that tell the installation the results of and how to respond to conditions found by the check. You can issue a particular message multiple times in a check routine. For an exception message, issue the HZSFMSG REQUEST=CHECKMSG request. See "Issuing messages in your local check routine with the HZSFMSG macro" on page 117.	• After Init function • At specified check interval • When check run is requested. • When a check parameter changes.
	What does the system do? • If a check abends for three iterations in a row, the system stops calling the check, which will not run again until it is refreshed or its parameter is changed. • Obtains 4K of dynamic work area pointed to by field PQE_DynamicAreaAddr . The contents of this work area are not set to any particular value and are not preserved across check iterations.	
Cleanup	**What should the check do?** For PQE_Function_Code_Cleanup, the check routine should clean up anything that you want cleaned between check iterations. For example, cleanup anything that you are not cleaning up in Check processing, or that must be cleaned up if Check processing abends. If you obtained resources owned by the current task during check function processing, check the PQE_CleanupInDifferentTaskThanCheck bit. If the bit is on, the system has already cleaned up the resources for you.	• After Check function
	What does the system do? The system obtains 4K of dynamic work area pointed to by field PQE_DynamicAreaAddr. The contents of this work area are not set to any particular value and are not preserved across check iterations.	
Delete	**What should the check do?** For PQE_Function_Code_Delete, the check routine should free any storage obtained during Init or Check processing that has not yet been freed.	• Delete • Refresh • When the check transitions out of the active enabled state. For example, when the check issues HZSFMSG with the STOP request. • When the IBM Health Checker for z/OS address space stops.
	What does the system do? The system: • Obtains 4K of dynamic work area pointed to by field PQE_DynamicAreaAddr . The contents of this work area are not set to any particular value and are not preserved across check iterations. • Stops calling the check.	

Creating and using data saved between restarts

Your check can use the HZSPDATA data set for persistent data. Persistent data is data that you want to save between restarts of either the system or IBM Health Checker for z/OS. When you issue the HZSPWRIT macro to write persistent data, the system saves data from two IPLs in HZSPDATA, the current IPL and the IPL prior to the current. Then, for each IPL, HZSPDATA contains two instances of data - one for the first iteration of the check and another for the most recent iteration of the check that wrote data to HZSPDATA. The first instance does not change for the life of the IPL, but the second instance is replaced each time a check writes data to the HZSPDATA data set.

You can read data from the HZSPDATA data set using the HZSPREAD macro. Commonly, checks use HZSPDATA to compare current data to the data saved in the HZSPDATA data set from one of the saved IPLs.

We have a couple of tips for you in using HZSPREAD and HZSPWRIT macros to read and write persistent data:

- Before you try to work with the persistent data that you read from the HZSPDATA data set, make sure your code checks for the following HZSPREAD return codes:
  - Return code 8, reason code X'xxxx082D', equate symbol HzspreadRsn_NoMatch indicates that no persistent data exists for this check.
  - Return code 8, reason code X'xxxx0830', equate symbol HzspreadRsn_DataDoesNotExist indicates that there is persistent data saved for this check, but not for the requested IPL.
- Tips for using HZSPWRIT:
  - You cannot delete data from the HZSPDATA data set once you have written it there. You can only replace the data in the current IPL instance in HZSPDATA.
  - You cannot write a null record to HZSPDATA.
  - You can issue multiple HZSPWRIT requests in a single check iteration. If the check iteration completes normally (returns to its caller), all of the data accumulated by HZSPWRIT requests for that iteration are written to HZSPDATA. If the check iteration does not complete normally, none of the data provided on HZSPWRIT requests for that check iteration is written to HZSPDATA.

Gotcha: After your check writes data to the HZSPDATA data set using HZSPWRIT, it takes one hour before data is actually hardened. That means that if the installation restarts IBM Health Checker for z/OS before an hour or re-IPL less than an hour has elapsed since the last HZSPWRIT, the data will not be saved in the HZSPDATA data set. IBM Health Checker for z/OS operates this way so that if a problem such as the following occurs, the system does not retain the data in the HZSPDATA data set:

- The check iteration completes with an abend
- A remote check iteration is unsuccessful
- An invocation of HZSPWRIT is unsuccessful

Note that an unsuccessful check iteration or HZSPWRIT invocation does not have any correlation to whether or not the check detected one or more exceptions.

Planning for persistent data: Sample HZSALLCP in SYS1.SAMPLIB shows how to allocate and initialize the HZSPDATA data set. When you are allocating space for

the HZSPDATA data set, keep in mind that in "Allocate the HZSPDATA data set to save check data between restarts" on page 10, we tell customers to define the HZSPDATA data set with a logical record length of 4096. You must plan for four sets of data: for each of the two instances for both the current and previous IPLs.

Authorization for HZSPDATA: You can define RACF profiles in the XFACILIT class for resources accessing the HZSPDATA.

Note that checks reading from or writing to the HZSPDATA data set must be both APF authorized and also have indicated not to do security checks, or they must have the appropriate access (READ or UPDATE) to either of the following:

- XFACILIT class resource HZS.sysname.checkowner.PDATA
- XFACILIT class resource HZS.sysname.checkowner.checkname.PDATA

See "HZSPREAD macro — Read Check Persistent Data" on page 339 and "HZSPWRIT macro — Write Check Persistent Data" on page 349 for information about authorization for checks to the HZSPDATA data set.

The following example shows how you might define a RACF profile for read or update access to HZSPDATA data set for a check:

```
RDEFINE XFACILIT HZS.sysname.checkowner.checkname.PDATA UACC(NONE)
PERMIT HZS.sysname.checkowner.checkname.PDATA CLASS(XFACILIT) ID(hzspdid) ACCESS(READ|UPDATE)
SETROPTS CLASSACT(XFACILIT) RACLIST(XFACILIT)
```

If you have already RACLISTed the XFACILIT or FACILITY class, which you probably have if you have IBM Health Checker for z/OS set up, just use the REFRESH parameter on the SETROPTS statement:

```
SETROPTS RACLIST(XFACILIT) REFRESH
```

Use the SECCHECK(UNAUTH|ALL) parameter in your code to specify whether you want the system to verify the security for writing to or reading from HZSPDATA. See "HZSPWRIT macro — Write Check Persistent Data" on page 349 and "HZSPREAD macro — Read Check Persistent Data" on page 339.

Using ENF event code 67 to listen for check status changes

If your check is authorized, it can use the ENFREQ LISTEN service to detect check status changes. On the ENFREQ service, specify theX'20000000' status change event qualifier and the listener user exit routine that is to receive control after the specified event occurs. The listener user exit specified receives control when IBM Health Checker for z/OS comes up and notifies the check routine of the status change.

To listen for ENF event code 67, you must specify the qualifying events on the BITQUAL parameter, which specifies a 32-byte field, a hexadecimal constant, or a register containing the address of a 32-byte field containing a bit-mapped qualifier that further defines the event. The qualifier is mapped by mapping macro HZSZENF. The BITQUAL value for the status change event is Enf067_BitQual_StatusChange in the HZSZENF mapping macro. This might mean on eof the following:

- The check completed with a different result than the last time it ran. For example, the check ran successfully after the previous check run issued an exception or vice versa.
- The check was deactivated or deleted

The check then might want to issue the HZSQUERY macro to get information about the check.

This event may not be presented if IBM Health Checker for z/OS is terminating (indicated by a X'40000000' ENF 067 event for NotAvailable - see “Using ENF event code 67 to listen for IBM Health Checker for z/OS availability” on page 139).

If the check routine decides it is no longer interested in knowing if IBM Health Checker for z/OS is up or not, it can issue the ENFREQ REQUEST=DELETE request to delete the listen request.

For information about ENFREQ and listener exits, see:

- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
- Listening for system events in *z/OS MVS Programming: Authorized Assembler Services Guide*

Issuing messages in your local check routine with the HZSFMSG macro

To issue a message with check results in your check routine, you must use the HZSFMSG macro (“HZSFMSG macro — Issue a formatted check message” on page 307), which you can issue in either an assembler or Metal C check routine.

This section only covers using the HZSFMSG macro to issue a message, but a message also consists of a few other ingredients. When your check runs, the system assembles the message from the following:

- The actual text and explanation for your check messages are defined in your message table, see “Issuing messages in your local check routine with the HZSFMSG macro.”
- The variables for your check messages are defined in the HZSMGB data area from your check routine. See “Defining the variables for your messages” on page 120.

Note that you can omit the message table and issue messages directly from the check routine in one of the following ways:

- For local or remote checks, use HZSFMSG REQUEST=DIRECTMSG. See “HZSFMSG macro — Issue a formatted check message” on page 307.
- For REXX checks, use the REQUEST='DIRECTMSG' subfunction of the HZSLFMSG function. See “Input variables for HZSLFMSG_REQUEST='DIRECTMSG'” on page 242.

You can issue the following kinds of messages in your check routine:

- Exception messages and other check results messages (CHECKMSG or DIRECTMSG request). For an overview of the various message types, see Table 20 on page 216.
- IBM Health Checker for z/OS messages (HZSMSG request)
- IBM Health Checker for z/OS messages that indicate that the check is stopped (STOP request). If your check routine issues HZSFMSG with the STOP request, it prompts the system to call the delete function code for the check.

You can issue a particular message multiple times in a single iteration of a check - a check routine should always issue an exception message to report an error.

Check messages are important because they report the results of the check to an installation. Each check should issue at least:

- One or more messages for any exception found to the setting the check is looking for.
- A message indicating that no exceptions were found, when appropriate.

If an HZSFMSG macro call is incorrect, the system issues system abend X'290' with a unique reason code and creates a logrec error record. The system checks the following for each HZSFMSG call:

- To see that the HZSMGB data area (input to checks describing message identifiers and variables) is complete
- That the message is in the message table
- That the number of inserts provided on the call exactly matches the number required to complete the message
- That each variable definition is between 1-256 characters long

The reason codes for system abend X'290' describe the message error. See *z/OS MVS System Codes*.

HZSFMSG updates the PQE_Result field in the HZSPQE as follows:

- For a specified severity of HIGH, the system sets the check result to 12
- For a specified severity of MEDIUM, the system sets the check result to 8
- For a specified severity of LOW, the system sets the check result to 4

PQE_Result is set to 0 when the check is called. See "Examples" on page 337.

For information on coding the message texts and explanation for messages, see "Issuing messages in your local check routine with the HZSFMSG macro" on page 117.

Reporting check exceptions

When a check detects a system condition or setting that runs counter to the values that the check is looking for, the check should issue an exception message to report the exception. For an exception message, the system displays both the message text and the entire message explanation in the message buffer. The message should include a detailed explanation of the error **and** the appropriate action that the installation should take to resolve the condition. If you are writing a check that checks for a setting that conflicts with the default for the setting, you should include in your check output information about **why** the check user is getting an exception message for a default setting.

Along with an exception message, IBM Health Checker for z/OS will issue a line showing the severity and the return code for the check. The check will continue to run at the defined intervals, reporting the exception each time until the exception condition is resolved.

The following example shows an exception message issued to the message buffer:

```
CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
START TIME: 05/25/2005 09:42:56.690844
CHECK DATE: 20040703  CHECK SEVERITY: HIGH

* High Severity Exception *

IRRH204E The RACF_SENSITIVE_RESOURCES check has found one or
more potential errors in the security controls on this system.
```

```
  Explanation:  The RACF security configuration check has found one or
    more potential errors with the system protection mechanisms.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator and the system auditor.

  System Programmer Response:  Examine the report that was produced by
    the RACF check. Any data set which has an "E" in the "S" (Status)
    column has excessive authority allowed to the data set. That
    authority may come from a universal access (UACC) or ID(*) access
    list entry which is too permissive, or if the profile is in WARNING
    mode. If there is no profile, then PROTECTALL(FAIL) is not in
    effect. Any data set which has a "V" in the "S" (Status) field is
    not on the indicated volume. Remove these data sets from the list
    or allocate the data sets on the volume.

    Asterisks ("****") in the UACC, WARN, and ID(*) columns indicate
    that there is no RACF profile protecting the data set. Data sets
    which do not have a RACF profile are flagged as exceptions, unless
    SETROPTS PROTECTALL(FAIL) is in effect for the system.

    If a valid user ID was specified as a parameter to the check, that
    user's authority to the data set is checked. If the user has an
    excessive authority to the data set, that is indicated in the USER
    column. For example, if the user has ALTER authority to an
    APF-authorized data set, the USER column contains "<Read" to
    indicate that the user has more than READ authority to the data set.

  Problem Determination:  See the RACF System Programmer's Guide and
    the RACF Auditor's Guide for information on the proper controls for
    your system.

  Source:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Reference Documentation:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Automation:  None.

  Check Reason:  Sensitive resources should be protected.

END TIME: 05/25/2005 09:43:13.717882  STATUS: EXCEPTION-HIGH
    APF-authorized data set, the USER column contains "
```

The **Check Reason:** field display the default reason in an exception message without installation parameter overrides.

See "Issuing a REXX check exception message" for an example of how to issue an exception message from a REXX check.

Example - Issuing a DIRECTMSG message for a REXX check: For a check that has no message table associated with it, you can issue a check message directly from the check routine, as shown in the example below. REXX sample check SYS1.SAMPLIB(HZSSXCHN) also shows DIRECTMSG calls.

```
/* Set up exception message input for HZSLFMSG */
/* Required input variables: */
HZSLFMSG_REQUEST='DIRECTMSG'
HZSLFMSG_REASON='CHECKEXCEPTION'
HZSLFMSG_DIRECTMSG_ID='UTHH003E'
HZSLFMSG_DIRECTMSG_TEXT='Brief exception summary'
/* Optional input variables: */
HZSLFMSG_DIRECTMSG.EXPL='The exception explanation for UTHR003E'
HZSLFMSG_DIRECTMSG.AUTOMATION='Automation text for UTHR003E'
HZSLFMSG_DIRECTMSG.SOURCE='Source text for UTHR003E'
/* Call HZSLFMSG */
HZSLFMSG_RC = HZSLFMSG()

/* Set up report message input for HZSLFMSG */
HZSLFMSG_REQUEST='DIRECTMSG'
HZSLFMSG_REASON='CHECKREPORT'
HZSLFMSG_DIRECTMSG_TEXT='Single line report message'
/* Call HZSLFMSG */
HZSLFMSG_RC = HZSLFMSG()
```

Defining the variables for your messages

The variable information for your check messages is defined in the HZSMGB data area by your check routine. The check routine defines information about variables and points to the HZSMGB data area for variable values. For Metal C check routines, the HZSMGB data area layout is mirrored in the HZSHMGB header.

There are two HZSMGB formats you can use to map your keywords:

- **MGBFORMAT=0**: Requires you to point to a separately defined area in storage where the length and value of the variable are defined, mapped by MGB_InsertD. See “Using default HZSMGB data area format (MGBFORMAT=0)” on page 121.
- **MGBFORMAT=1**: Allows you to specify the length and address of the variables in HZSMGB fields MGB1_MsgInsertDesc_Length and MGB1_MsgInsertDesc_Addr in the MGB1_MsgInsertDesc mapping. See “Using HZSMGB data area format MGBFORMAT=1” on page 123.

Figure 14 on page 201 shows how messages with variables get resolved at check runtime.

Use the following guidelines in defining variables for your messages:

Match up the number of variables in the HZSMGB data area and the message table, because if you end up with a mismatch, your check will abend when it issues the HZSFMSG macro to issue the message. Look in the logrec error record or *z/OS MVS System Codes* to find the description of the reason code issued with the abend.

To keep text on the same line, replace blank characters, X'40', with the required blank character X'44'.

If I use the same variable twice in a message, do I have to define it twice in the HZSMGB data area? Yes, every time you use a variable, even if you use the same variable several times in the same message, you must point to separate entries in the MGB_Inserts field for each variable instance. However, each of the entries for an identical variable can point to the same area in storage where the variable length and value are specified for the variable.

Can I build the HZSMGB information for all my check messages once at initialization and then reuse them whenever the check runs? Tempting idea, but no. The problem with this method is that there's no guarantee that the HZSPQE data area for the check will be in the same place for any given run of your check. Although the contents of the PQEChkWork section are the same for every run of the check, it's location is not. Thus if you try to point within your PQEChkWork area for variable information, the offset will be the same, but the full address probably will not be.

On the other hand, if you are pointing into either your check routine module or an area that you GETMAINed at initialization to build your HZSMGB data area, those areas will stay the same, and so the build once/use multiple times approach might work. But this is a tricky maneuver.

In the HZSMGB data area, variables do not have variable names. You insert the length (MGB_MsgILen field) and value (MGB_MsgIVal field) for a variable without using the variable name you use in the check routine.

Can I have a null variable? You can indeed have a null variable by defining a variable length of zero in the MGB_MsgILen field.

What happens if I make a mistake updating HZSMGB? If you make a mistake while updating HZSMGB so that your variable values are not compatible with the variable attributes in the message output at check runtime, your check will most likely abend with system abend code X'290' and a reason code that describes the error. The system also writes a record to SYS1.LOGREC that provides additional detail in the variable recording area (VRA).

Using default HZSMGB data area format (MGBFORMAT=0)

Figure 8 on page 122 shows an example of how you define the message variables in your check routine:

1 shows an example of defining the message number in the MGB_MessageNumber.

2 shows an example of filling in the MGB_InsertCnt field with the number of variables for your message.

3 shows an example of putting the address of one variable into the MGB_Inserts field. This address points to the area in storage where the length and value of the variable are defined, mapped by MGB_InsertD.

4 shows an example of defining the length and value of the variable in the MGB_MsgILen and MGB_MsgIVal fields for the variable in storage. These fields are in the MGB_InsertD mapping.

5 shows an example of issuing a message. Note that this example shows a local message. **For a remote check**, the HZSFMSG macro must include the REMOTE=YES, HANDLE=*handle*, and MSGTABLE=*msgtable* parameters.

6 shows how the variable address, length, and value are defined to be stored in the HZSMGB data area or in storage.

7 shows an example of creating an area big enough in the HZSMGB for the information about all your variables. To create enough room for all your variables,

use the formula HZSMGB_LEN + (*n*-1)*L'MGB_inserts where *n* is the number of inserts. HZSMGB_LEN by itself will provide room for only one insert.

Figure 8 shows check routine code that defines variable data in the HZSMGB:

```
************************************************************
* Issue a message with two inserts                         *
************************************************************
         SYSSTATE ARCHLVL=1
* save regs, get dynamic storage, chain saveareas, set usings
         LA    2,TheMGBArea
         ST    2,TheMGBAddr
         USING HZSMGB,2
        1  MVC   MGB_MessageNumber,=F'1' Message 1
        2  MVC   MGB_insert_cnt,=F'2'  Two inserts
         LA    3,Insert1Area         Address of first insert
        3  ST    3,MGB_Inserts         Save insert address
         LA    3,Insert2Area         Address of second insert
         USING MGB_MsgInsertD,3
        4  MVC   MGB_MsgILen,=AL2(L'Insert2Val)  Insert length
         MVC   MGB_MsgIVal(L'Insert2Val),MyMod Insert value
         DROP  3
         ST    3,MGB_Inserts+4       Save insert address
        5  HZSFMSG  REQUEST=CHECKMSG,MGBADDR=TheMGBAddr,       *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,FMSGL)
         DROP  2
*
* Place code to check return/reason codes here
*
* free dynamic storage, restore regs
         BR 14
MyMod    DC   CL8'MYMODULE'  6
* Area for first insert
Insert1Area DS 0H
Insert1Len  DC AL2(L'Insert1Val)
Insert1Val  DC C'CSA   '
         LTORG ,
         HZSZCONS ,            Return code information
         HZSMGB   ,            Insert mapping
DYNAREA  DSECT
LRETCODE DS   F
LRSNCODE DS   F
* Area for 2 inserts (HZSMGB_LEN accounts for one, so
* we add one more "length of MGB_Inserts")
TheMGBAddr DS A  7
TheMGBArea DS CL(HZSMGB_LEN+1*L'MGB_Inserts)
* Area for second insert
Insert2Area DS 0H
Insert2Len  DS AL2(L'Insert2Val)
Insert2Val  DC X'00950000'
         HZSFMSG MF=(L,FMSGL),PLISTVER=MAX
DYNAREA_LEN EQU *-DYNAREA
```

Figure 8. Example of issuing a message with variables in an assembler check

Important fields in the HZSMGB data area include:

Table 12. Important MGBFORMAT=0 fields in the HZSMGB data area for check message variables

Field name	Meaning
MGB_MessageNumber MGB_ID	Fullword field containing the value identifying each message. These fields are the same - there are two names for this field. This field corresponds to the xreftext value for each message. For example, the xreftext value for a message is coded as follows: `<msgnum xreftext="001">TESTMSG1I</msgnum>`
MGB_InsertCnt	Fullword field containing the number of variables (or inserts) to follow.
MGB_Inserts MGB_InsertAddr	These fields are the same - there are two names for this field. This field contains an array of pointers, each of which contains the address in storage of an area for a specific variable. This area is mapped by Mgb_MsgInsertD.
MGB_MsgInsertD	A structure in the HZSMGB data area that describes the length and value of the variable: • MGB_MsgILen, which is a 2 byte field containing the length of the variable. • MGB_MsgIVal, which contains the value of the variable.

Using HZSMGB data area format MGBFORMAT=1

1 shows an example of defining the message number in the MGB1_MessageNumber field.

2 shows an example of filling in the MGB1_Insert_Cnt field with the number of variables for your message.

3 shows examples of defining the length and address of the variable in the MGB1_MsgInsertDesc_Length and MGB1_MsgInsertDesc_Addr fields for the variable in storage. These fields are in the MGB1_MsgInsertDesc mapping.

4 shows an example of issuing a message. Note that this example shows a local message. **For a remote check**, the HZSFMSG macro must include the REMOTE=YES, HANDLE=*handle*, and MSGTABLE=*msgtable* parameters.

5 shows how the variable address, length, and value are defined to be stored in the HZSMGB data area or in storage.

6 shows an example of creating an area big enough in the HZSMGB1 for the information about all your variables. To create enough room for all your variables, use the formula HZSMGB1_LEN1 + (n)*MGB1_MsgInsertDesc_Len where n is the number of inserts.

Figure 9 on page 124 shows check routine code that defines variable data in the HZSMGB data area using MGBFORMAT=1:

```
************************************************************
* Issue a message with two inserts                         *
************************************************************
         SYSSTATE ARCHLVL=2
* save regs, get dynamic storage, chain saveareas, set usings
         LA 2,TheMGBArea
         ST 2,TheMGBAddr
         USING HZSMGB1,2 1    MVC   MGB1_MessageNumber,=F'1' Message 1 2    MVC   MGB1_insert_cnt,=F'2' Two inserts
         DROP  2
         PUSH  USING
         USING MGB1_MsgInsertDesc,TheMSGInsertDesc1 3    MVC   MGB1_MsgInsertDesc_Length,=AL2(L'Insert1Val) Insert length
         LA    15,Insert1Val
         ST    15,MGB1_MsgInsertDesc_Addr  Insert address
         POP   USING
         PUSH  USING
         USING MGB1_MsgInsertDesc,TheMGBInsertDesc2
         MVC   MGB1_MsgInsertDesc_Length,=AL2(L'Insert2Val) Insert length
         LA    15,Insert2Val
         ST    15,MGB1_MsgInsertDesc_Addr  Insert address
         POP   USING 4      HZSFMSG REQUEST=CHECKMSG,MGBADDR=TheMGBAddr,                    *
               MGBFORMAT=1,                                                 *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,                           *
               MF=(E,FMSGL)
*
* Place code to check return/reason codes here
*
* free dynamic storage, restore regs
         BR    14 5
* Area for first insert
Insert1Val DC C'CSA '
* Area for second insert
Insert2Val DC X'00950000'
         LTORG ,
         HZSZCONS , Return code information
         HZSMGB , Insert mapping
DYNAREA  DSECT
LRETCODE DS F
LRSNCODE DS F
TheMGBAddr DS A
* Area for 2 inserts     6
TheMGBArea     DS CL(HZSMGB_LEN1)
TheMSGInsertDesc1  DS CL(MGB1_MsgInsertDesc_Len)
TheMSGInsertDesc2  DS CL(MGB1_MsgInsertDesc_Len)
HZSFMSG  MF=(L,FMSGL),PLISTVER=MAX
DYNAREA_LEN EQU *-DYNAREA
```

Figure 9. Example of issuing a message with variables using MGBFORMAT=1

Important MGBFORMAT=1 fields in the HZSMGB data area include:

Table 13. Important MGBFORMAT=1 fields in the HZSMGB data area for check message variables

Field name	Meaning
MGB1_MessageNumber MGB1_ID	Fullword field containing the value identifying each message. These fields are the same - there are two names for this field. This field corresponds to the xreftext value for each message. For example, the xreftext value for a message is coded as follows: `<msgnum xreftext="001">TESTMSG1I</msgnum>`
MGB1_InsertCnt	Fullword field containing the number of variables (or inserts) to follow.
MGB1_MsgInsert Desc_Length	The length of the variable. For a null variable, use a length of zero.
MGB1_MsgInsert Desc_Addr	The address of the variable. For a null variable, you need not set this field.

Writing a check with dynamic severity levels

You can create your check routine to issue check exception messages with a dynamically varying severity level, giving users more control over how exception messages are issued and handled. For example, you might use the dynamic severity function for checks that inspect a system setting value and compare it against a threshold. As the value approaches the high threshold, the check can vary the severity of the exception, depending on how close to the threshold the value is.

For information on how users work with checks enabled with dynamic severity, see “Customizing check exceptions with dynamically varying severity” on page 31

How to enable a check for dynamic severity: In order to allow the customer to use the dynamic severity function do the following:

1. Define check parameters that let the check know what severity exception to issue. For example, if your check looks for parameter 'p', define the following parameters:
   - p_HIGH
   - p_MED
   - p_LOW
   - p_NONE

   In addition, define the corresponding short forms of the parameter, for example for our case, you would define p_H, p_M, p_L, and p_N parameters. And, because it's always likely that customers could forget the exact parameter specifications, try to build in the obvious mistakes. For example, a customer might try p_HI when the correct parameter is p_HIGH, so we recommend that you also support p_HI and p_NO (as a short version of p_NONE).

   We have the following tips for you in defining dynamic severity check parameters: Note that the parameter names you create cannot exceed the 16 character parameter length limit for the ADD/UPDATE CHECK interface!
   - Of course, the customer should not specify multiple forms for a given severity, but it is not worth while enforcing that in your check routine. Instead, program your check to accept the longest parameter. In other words, if a customer specifies both p_HIGH and p_H parameters for a check, your check should use the value from p_HIGH.
   - If you are writing a new check that exploits dynamic severity, we recommend that you use the suffixed parameters (p_HIGH, p_MED, p_LOW and so on) and do not code a non-suffixed, non-dynamic version of the 'p' parameter.

     On the other hand, if you are upgrading an existing check to use dynamic severity, keep the plain non-suffixed parameter and add the _HIGH, _MED, _LOW, _NONE variants as well. Then make sure to code your check to use the rule that if any of the dynamic severity variants are present, then the non-dynamic severity variant is ignored. That way, you don't have to worry about interpreting cases where the customer specifies both a non-dynamic 'p' version of the parameter and a dynamic p_HIGH version.

     If you are adding dynamic severity parameters for an old check, and none of the existing parameters allow an underscore either within the parameter name or value, you can code your check to assume that the customer is using a dynamic severity specification if you find an underscore within the parameter string.
   - Code your check so that the customer does not need to specify a severity for all parameters. They should be able to specify just the ones that they want.
2. Code your dynamic severity check to issue exception messages with the SEVERITY parameter on the HZSFMSG service or using REXX function HZSLFMSG_SEVERITY, as follows:
   - If no dynamic severity parameter is provided and the criterion for an exception based on the parameter is met, issue a severity-system message
   - Else if a p_HIGH parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-high message

- Else if a p_MED parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-medium message
- Else if a p_LOW parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-low message
- Else if a p_NONE parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-none message

Note that the severity specified on HZSFMSG or HZSLFMSG_SEVERITY overrides the default severity defined for the check when it was added.

3. Add the check with ALLOWDYNSEV=YES specified. See the ADD or ADDREPLACE CHECK paramters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68 or "HZSADDCK macro — HZS add a check" on page 260.

How dynamic severity and the SEVERITY in the check definition interact: As we mention above, a check using dynamic severity overrides the severity specified in the check definition. So that was easy. But just to keep things interesting, note that the implicit WTO handling of exception messages that is derived from the severity in either the check definition or the dynamic check severity being used can be overridden by a WTOTYPE specified in the check definition. See WTOTYPE in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Controlling check exception message WTOs and their automation consequences

By default, IBM Health Checker for z/OS issues DOM requests to delete any check exception message WTOs left behind from previous check iterations. It does this DOMing right before the start of the new check iteration. That means that each time the check generates an exception, it also sends a new exception WTO, which also kicks off any automation actions you've set up for your installation.

So, what if you want more control over check exception WTOs and their automation consequences? For example, let's say you have a check that runs every hour. Now let's say that your check begins generating identical exceptions that you've automated on to prompt a beeper call to your favorite system programmer. You have not yet resolved the exception issue, and the installation policy is to not disable checks generating exceptions. That's just good practice, right? And yet your check might generate a lot of WTOs and beeper calls to that poor system programmer while the issue gets resolved.

That's where DOM control comes in! Starting with z/OS V1R13, IBM Health Checker for z/OS you can use the following functions that help you control whether you want to suppress WTOs and any automation actions they trigger for a check that is generating exceptions:

1. Add your check to the product using the DOM(CHECK) parameter on the HZSPRMxx and MODIFY *hzsproc* command. See ADD or ADDREPLACE CHECK parameters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.
2. From your check you decide when to DOM WTOs from previous check runs using the HZSFMSG REQUEST=DOM macro ("HZSFMSG macro — Issue a formatted check message" on page 307) or the REXX HZSLFMSG_REQUEST='DOM' function "HZSLFMSG function" on page 240.

Realizing the benefits of this function is all in the timing:

- If your check (added with DOM(CHECK)) is generating multiple identical unresolved exceptions, your check can wait to DOM the exception WTO (with HZSFMSG REQUEST=DOM or the REXX HZSLFMSG_REQUEST='DOM' function) until the exception condition is resolved. This way, your check is still running, but the exception WTOs from previous iterations of the check do not get DOMed. That means that exception messages from this check are just recorded in the message buffer and not sent as WTOs that set off automation actions.
- If your check is running successfully and is not generating an exception in a check iteration or is generating different check exceptions between iterations, your check should issue HZSFMSG REQUEST=DOM or the REXX HZSLFMSG_REQUEST='DOM' function to DOM WTOs from previous iterations. That way any subsequent exception will be sent as a WTO and will kick off any defined automation actions.

On the other hand, if you always want to delete WTOs for your check, and never wish to suppress duplicate exception WTOs, you will want to specify or default to DOM(SYSTEM) when you add your check, and let the system take care of DOMing of check WTOs for you!

The well-behaved local check routine - recommendations and recovery considerations

Make your check clean up after itself, because the system won't do it for you: IBM Health Checker for z/OS does not perform end-of-task cleanup for your check on a regular basis. Check routines should track resources, such as storage obtained, ENQs, locks, and latches, in the PQE_ChkWork field.

Release resources within the same function code processing: Whenever possible, the check routine should release resources within the same function code processing that it obtained. Releasing resources in a different function code call is error prone, because you cannot assume that the cleanup function processing will run under the same task as the Check function. If the Cleanup function does not run under the same task as Check function, it means that the task under which the Check function was running has been terminated.

Have your check stop itself when the environment is inappropriate: If your check routine encounters an environmental condition that will prevent the check from returning useful results, your check routine should stop itself and not run again until environmental conditions change and your code requests it to run. Your check should do the following to respond to an inappropriate environment:

1. Issue an information message to describe why the check is not running. For example, you might issue the following message to let check users know that the environment is not appropriate for the check, and when the check will run again:

   ```
   The server is down.
   When the server is available, the check will run again.
   ```

2. Issue the HZSFMSG service to stop itself:

   ```
   HZSFMSG REQEST=STOP,REASON=ENVNA
   ```

3. Make sure that your product or check includes code that can detect a change in the environment and start running the check again when appropriate. To start running the check, issue the following HZSCHECK service:

   ```
   HZSCHECK REQUEST=RUN,CHECKOWNER=checkowner,CHECKNAME=checkname
   ```

If the environment is still not appropriate when your code runs the check, it can always stop itself again.

Your check should not add itself in an inappropriate environment: If you use a HZSADDCHECK exit routine r to add your checks to the system, note that some checks or product code might add or delete checks to the system in response to changes in system environmental conditions. For example, if a check or product detects that a system environment is inappropriate for the check, it might then add only the checks useful in the current environment by invoking the HZSADDCHCK registration exit with an ADDNEW request (from the HZSCHECK service, the F *hzsproc* command, or in the HZSPRMxx parmlib member. You should add similar code to your HZSADDCHECK exit routine r to make sure that your checks don't run if they will not return useful results in the current environment. This code might:

- Delete checks that do not apply in the current environment
- Run a check so that it can check the environment and disable itself if it is inappropriate in the current environment. Consider supporting a check PARM so the installation may indicate the condition is successful and not an error.

If your check can never be valid for the current IPL, consider not even adding it from your HZSADDCHECK exit routine when you detect that situation. For example, if a check is relevant only when in XCF LOCAL mode but the system is not in that mode (and cannot change to that mode), there is no reason even to add the check.

Have your check stop itself for bad parameters: If your check routine is passed a bad parameter, it should stop itself using the HZSFMSG service:

```
HZSFMSG REQUEST=STOP,REASON=BADPARM
```

This request will also issue predefined HZS1001E error message to indicate what the problem is. The check routine will not be called again until it is refreshed or its parameters are changed. REQUEST=STOP prevents the check from running again and sets the results in the PQE_Result field of HZSPQE. The system sets the result field based on the severity value for the check. See “Issuing messages in your local check routine with the HZSFMSG macro” on page 117 for examples and complete information.

Plan recovery for abends: Your check routine should be designed to handle abends. If on three consecutive check iterations:

- HZSFMSG issues abend X'290'
- The check abends and its recovery does not retry

then the system renders the check inactive until the check is refreshed, or parameters for the check are changed. If the check routine has obtained a resource that needs to be released under the same function code processing, but the check routine abends, a recovery routine can release that resource. IBM suggests that you use either an ESTAEX or IEAARR recovery routine.

In some cases you may not want your check to be stopped when an abend occurs because some abend causing conditions might simply clear with time. For example, if your check abends as a result of getting garbled data from an unserialized resource, such as a data area in the midst of an MVC, your check should provide its own recovery to:

- Retry the check a pre-determined number of times.
- If the check fails again, the check should stop running, but not stop itself.

This allows the check to try running again at the next specified interval, with every chance of success this time.

Take advantage of verbose and debug modes in your check:

IBM Health Checker for z/OS has support for the following modes:

- Debug mode, which tells the system to output extra messages designed to help you debug your check. IBM Health Checker for z/OS outputs some extra messages in debug mode, and some checks do also. When a check runs in debug mode, each message line is prefaced by a message ID, which can be helpful in pinpointing the problem. For example, report messages are not prefaced by message IDs unless a check is running in debug mode.

  There are two ways to issue extra messages in debug mode:

  – Use conditional logic such that when in debug mode (when field PQE_DEBUG in mapping macro HZSPQE has the value PQE_DEBUG_ON), your check issues additional messages.
  – Code debug type messages - see "Planning your debug messages" on page 203

  Users can turn on debug mode using the DEBUG=ON parameter in the MODIFY *hzsproc* command, in HZSPRMxx, or by overtyping the DEBUG field in SDSF to ON.
- Verbose mode, which tells the system to output messages with additional detail about non-exception information found by the check. (RACF checks, for example, issue additional detail in verbose mode.) To issue extra messages in verbose mode, use conditional logic such that when in verbose mode (when field PQE_VERBOSE in mapping macro HZSPQE has the value PQE_VERBOSE_YES), your check issues additional messages.

  Users can turn on verbose mode using the VERBOSE=YES parameter in the F *hzsproc* command or in HZSPRMxx.

Look for logrec error records when you test your check: When testing your check, be sure to look for logrec error records. The system issues abend X'290' if the system encounters an error while a message is being issued, and issues a logrec error record and a description of the problem in the variable recording area (VRA).

Save time, save trouble - test your check with these commands: When you have written your check, test it with the following commands to find some of the most common problems people make in writing checks:

```
F hzsproc,UPDATE,CHECK(check_owner,check_name),DEBUG=ON
F hzsproc,UPDATE,CHECK(check_owner,check_name),PARM=parameter,REASON=reason,DATE=date
F hzsproc,DELETE,CHECK(check_owner,check_name),FORCE=YES
F hzsproc,DISPLAY,CHECK(check_owner,check_name),DETAIL
```

Avoid disruptive practices in your check routine: The IBM Health Checker for z/OS philosophy is to keep check routines very simple. IBM recommends that checks read but not update system data and try to avoid disruptive behavior such as:

- Modifying system control blocks
- I/O intensive operations, such as reading a data set
- Serialization
- Waits (directly or by services you call)
- Creating new tasks
- Creating new address spaces

We're recommending against these practices because they require more overhead, complicate your check routine, and, more seriously, can affect the performance of other system functions. In addition, these practices can affect the running of other checks, since only 20 local check routines can be in control concurrently. But you'll need to decide what's appropriate on a check by check basis. An ENQ, for example, serializing on a control block, can indeed affect the performance of other functions that might need that control block. However, the downside of not serializing is that a check might get information that is not consistent. You must weigh the cost to customers of the chance of getting inconsistent data versus the costs of using an ENQ in terms of system performance and IBM Health Checker for z/OS processing.

See also "Debugging checks" on page 132.

Building Metal C checks

To make it easier to compile and build, link-edit, and bind a Metal C check, IBM Health Checker for z/OS provides a sample makefile, hzssmake.mk, for use with the z/OS UNIX System Services make utility. This makefile compiles and builds the sample files shipped in z/OS UNIX file system directory /usr/lpp/bcp/samples, where the makefile itself is shipped also.

Before you use the makefile, make sure you update the HCHECK_LOADLIB variable in the makefile. This variable names the dataset where the makefile will store the final load modules. This should be an APF authorized dataset in the link list, suitable for your installation.

To create all sample load modules, change to the directory where the hzssmake.mk file is stored and invoke the make utility like this:

```
make -f hzssmake.mk
```

Check out the other make rules in the makefile, in particular the cleanup rules. You can invoke cleanup, for example, using the following command:

```
make -f hzssmake.mk clean
```

This command will clean up all intermediate files, but will keep the generated load modules.

Once built, your Metal C load modules are ready to be registered with IBM Health Checker for z/OS as you would any other check. See:

- "Defining a local check to IBM Health Checker for z/OS" on page 108
- "Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS" on page 140
- "Creating product code that automatically registers checks at initialization" on page 197

For a Metal C sample HZSADDCHECK exit routine r, look for hzscadd.c in /usr/lpp/bcp/samples .

For more information about the make utility and the other utilities used in the makefile, see Shell command descriptions in *z/OS UNIX System Services Command Reference*.

```
############################################################
# Name: HZSSMAKE                                           #
#                                                          #
# Description:  Makefile for building Metal C sample       #
```

```
#               local and remote health checks and             #
#               a sample HZSADDCHECK exit routine.             #
#                                                              #
#  COMPONENT: IBM Health Checker for z/OS (SCHZS)              #
#                                                              #
#  PROPRIETARY STATEMENT:                                      #
#                                                              #
#    Licensed Materials - Property of IBM                      #
#    5650-ZOS                                                  #
#    Copyright IBM Corp. 2009                                  #
#                                                              #
#    US Government Users Restricted Rights - Use, duplication  #
#    or disclosure restricted by GSA ADP Schedule Contract with#
#    IBM Corp.                                                 #
#                                                              #
#  END OF PROPRIETARY STATEMENT                                #
#                                                              #
#  STATUS = HBB7770                                            #
#                                                              #
# Change Activity:                                             #
#                                                              #
# $L0=METALC    HBB7770 20081202 PDGIO: Initial version        #
# $L1=METALC    HBB7770 20090513 RDUT:  Updated options,targets#
#                                                              #
################################################################

# The load modules created via this makefile will be put into this PDSE
# dataset. Change this to an APF authorized dataset in the link list,
# suitable for your installation.
# The linker/binder will create the PDSE, if it does not exist yet.
HCHECK_LOADLIB =HLQ.LOADLIB

# Location of Health Checker header filesHC_INCLUDES = "//'SYS1.SIEAHDR.H'"

# (Metal-) C compiler utility
CC = c99

# (Metal-) C compiler flags
# nosearch  - avoids using the non-Metal C header files
# I         - specifies our include paths, since nosearch disabled most
# metal + S - makes it Metal C instead of "regular" C/C++
# longname  - optional, but allows for longer than 8 character names
CFLAGS = -S -Wc,metal,longname,nosearch \
         -I /usr/include/metal,$(HC_INCLUDES)

# Assembler utility
AS = as

# Assembler flags
# rent - requests reentrant code; required for health checks
# goff - optional, but allows for longer than 8 character names
ASFLAGS = -mrent -mgoff

# Linker/binder utility
LD = ld

# Linker/binder flags
# ac=1 - assigns authorization code; required for health checks
# rent - requests reentrant code; required for health checks
# -S   - resolves system services (PAUSE token handling by remote
#        health checks) via SYSLIB CCSLIB
LDFLAGS  = -bac=1 -brent
LDFLAGSR = -S "//'SYS1.CSSLIB'"

# The four sample health checks and the one sample exit routine
HCHECK_TGTS = hzscchkp hzscchkr hzscrchc hzscrchk hzscadd
```

Local check routine

```
# Default rule
all: $(HCHECK_TGTS)

# *Uncomment* this rule, if you would like to keep the intermediate
# output files, in particular the generated .s assembler source,
# instead of letting 'make' delete them automatically.
#.SECONDARY:

# Rule for cleaning up intermediate output
clean:
 rm -f *.o *.s

# Rule for cleaning up all output
cleanall:
 rm -f *.o *.s
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscchkp)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscchkr)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscrchc)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscrchk)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscadd)'"

# Rule for compiling a Metal C file into assembly language
%.s: %.c
 $(CC) $(CFLAGS) $<

# Rule for creating object code from assembly language
%.o: %.s
 $(AS) $(ASFLAGS) -o $@ $<

# Rules for creating LOAD modules (executable) from the object code
hzscchkp: hzscchkp.o
 $(LD) $(LDFLAGS) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscchkr: hzscchkr.o
 $(LD) $(LDFLAGS) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscadd: hzscadd.o
 $(LD) $(LDFLAGS) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscrchc: hzscrchc.o
 $(LD) $(LDFLAGS) $(LDFLAGSR) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscrchk: hzscrchk.o
 $(LD) $(LDFLAGS) $(LDFLAGSR) -o "//'${HCHECK_LOADLIB}($@)'" $<
```

Debugging checks

Naturally, we hope you'll never need this section and that all your checks will run perfectly the very first time. However, if you do run into trouble, this section will help you debug your check routine and HZSADDCHECK exit routine.

Was my check added to the system? Use the F *hzsproc*,DISPLAY CHECK(*checkowner*,*checkname*) to display the check you're adding to the system. If your check shows up, it was successfully added to the system. If it does not show up, it was not added to the system.

You can also check the return code from the HZSADDCK invocation in your HZSADDCHECK exit routine (for local checks) or check routine (for remote checks). A return code greater than 4 often indicates that there was a problem adding the check to the system. See “HZSADDCK macro — HZS add a check” on page 260.

Turn on debug mode: Running in debug mode can help you debug your check, because in debug mode:

- Each message line is prefaced by a message ID, which can be helpful in pinpointing the problem. For example, report messages are not prefaced by message IDs unless a check is running in debug mode.
- Debug messages, which may contain information about the error, are issued only when the check is in debug mode.

You can turn on debug mode for a check that is not running properly using the DEBUG parameter in the MODIFY *hzsproc* command, in HZSPRMxx, or by overtyping the DEBUG field in SDSF to ON.

Create a recovery routine for your check routine if you need additional diagnostic data for your check routine. See "Establishing a recovery routine for a check" on page 110.

Debug HZSFMSG abends: If the system finds an error in a HZSFMSG macro call to issue a message, the system issues system abend X'290' with a unique reason code and creates a logrec error record. See the information for abend X'290' in *z/OS MVS System Codes* for a description of the abend reason codes.

If the abend is caused by an incorrect macro call, the system issues the following accompanying information:

- Logrec error record. Use EREP to view logrec errors, see "Using EREP to Obtain Records from the Logrec Log Stream " in *z/OS MVS Diagnosis: Tools and Service Aids*.
- A symptom dump written to the console and to the system log
- A SYSMDUMP, if you add a SYSMDUMP DD statement to *hzsproc*, the IBM Health Checker for z/OS procedure.

  Note that the contents and data set disposition of your SYSMDUMP depends on the DISP= option you use on the DD statement. See "Obtaining ABEND dumps" in *z/OS MVS Diagnosis: Tools and Service Aids*.
- There may be additional diagnostic data in the register at time of the abend that can help with debugging. See "ABEND Codes" on page 332 for the kinds of diagnostic data that may be available.

  If your check routine has a recovery routine, the SDWA for the recovery routine will contain these registers in the SDWAGRSV field.

If the abend is caused by the system, the system issues an SVC dump.

Where is my check routine? I need to locate it for debugging. If you do not receive an abend for a problem, you can locate a local check routine and message table (to use in a SLIP trap, for example) using the DIAG parameter on the F *hzsproc*,DISPLAY command. For example, you can use the `f hzsproc,display,check(IBMGRS,grs_mode),detail,diag` command. Note the diagnostic information, including the location of the check routine and message table in the output example below:

```
HZS0201I 13.06.05 CHECK DETAIL     716
CHECK(IBMGRS,GRS_MODE)
 STATE: ACTIVE(ENABLED)     GLOBAL  STATUS: SUCCESSFUL
 EXITRTN: ISGHCADC
 LAST RAN: 07/06/2005 12:49    NEXT SCHEDULED: (NOT SCHEDULED)
 INTERVAL: ONETIME              SEVERITY: LOW
 WTOTYPE: INFORMATIONAL
 SYSTEM DESCCODE: 12
```

```
DEFAULT PARAMETERS:       STAR
REASON FOR CHECK:  GRS should run in STAR mode to improve
                   performance.
MODIFIED BY: N/A
DEFAULT DATE: 20050105
DEBUG MODE: OFF
INTERNAL DIAGNOSTICS -  CHECK TOKEN: 01020038.7FE9F000
ROUTINE: ISGHCGRS-7F2B4BC8  MSGTBL: ISGHCMSG-7F222120  FUNC: CLEANUP
LAST CPU TIME: 0.041  MAX CPU TIME: 0.041
```

Where is my HZSADDCHECK exit routine? If you need to locate the address of your HZSADDCHECK exit routine for a local check, to set a SLIP trap, for example, use the display command following:

```
DISPLAY PROG,EXIT,EXITNAME=HZSADDCHECK_exit_routine,DIAG
```

The system issues message CSV464I displaying information about the exit, including the exit entry point address, the load point address of the exit routine module, and other diagnostic information for exit routine.

Using SLIP traps for debugging: If you need to set a SLIP trap for either your check routine or HZSADDCHECK exit routine, we suggest that you set a SLIP trap on any error event in the IBM Health Checker for z/OS address space instead of setting it on an abend X'290'. This will give you the information you need to handle both the X'290' abend and any other unexpected problem.

Use the two hints directly above this one to find the addresses of your check routine and HZSADDCHECK exit routine, for use in setting SLIP traps.

Chapter 7. Writing remote check routines

A remote check runs as a task in the caller's address space. To learn about the differences between local and remote checks and deciding which type you want to write, see “Remote checks” on page 98.

A IBM Health Checker for z/OS check gathers information about the system environment and parameters, compares them to suggested settings or looks for configuration problems, and then informs customers of the results through detailed messages. Because remote checks run in the caller's address space (rather than the IBM Health Checker for z/OS address space) you must ensure communication between the remote check routine and IBM Health Checker for z/OS.

To learn about the differences between local and remote checks and deciding which type you want to write, see “Remote checks” on page 98.

In this chapter, we'll cover the following:

- “Metal C or assembler?” on page 105
- “Sample checks” on page 136
- “Remote check routine basics” on page 137
- “Programming considerations” on page 138
- “Preparing for check definition - making sure IBM Health Checker for z/OS is up and running” on page 139
- “Allocate a pause element token using IEAVAPE” on page 140
- “Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS” on page 140
- “Pause the remote check routine with IEAVPSE” on page 143
- “Using HZSCHECK REQUEST=OPSTART and REQUEST=OPCOMPLETE to communicate check start and stop to IBM Health Checker for z/OS” on page 143
- “Using the check parameter parsing service (HZSCPARS)” on page 111
- “Using the HZSPQE data area in your remote check routine” on page 144
- “Release codes for remote check routines” on page 145
- “Creating and using data saved between restarts” on page 115
- “Issuing messages in your remote check routine with the HZSFMSG macro” on page 150
- “Writing a check with dynamic severity levels” on page 124
- “Controlling check exception message WTOs and their automation consequences” on page 126
- “Defining the variables for your messages” on page 120
- “Recommendations and recovery considerations for remote checks” on page 161
- “Building Metal C checks” on page 130
- “Debugging checks” on page 166

Metal C or assembler?

As mentioned above, you can write a local or remote check in either Metal C or assembler. The concepts in this section apply to either language.

Remote check routine

Metal C lets you create a IBM Health Checker for z/OS compatible load module that follows MVS linkage conventions. You can also easily use assembler macros, such as HZSFMSG, HZSCPARS, or any other assembler macro, in your Metal C check routine using the __asm keyword.

If you are writing in Metal C, IBM Health Checker for z/OS provides generic C header files (hzsh*.h) in SYS1.SIEAHDRV.H containing the following mappings of IBM Health Checker for z/OS structures and control blocks and commonly used Health Checker related constants:

Table 14. Correlation between IBM Health Checker for z/OS mapping macros and Metal C header files

Assembler mapping macros in SYS1.MACLIB	Description	Metal C header file in SYS1.SIEAHDRV.H
HZSPQE	Individual check data area	HZSHPQE
HZSDPQE	Individual deleted check data area	HZSHDPQE
HZSMGB	Message data area	HZSHMGB
HZSQUAA	HZSQUERY return information data area	HZSHQUAA
HZSZCPAR	Check parameter area	HZSHCPAR
HZSZENF	Event data for ENF 67	HZSHENF
HZSZHCKL	Message buffer log entry	HZSHHCKL
HZSZCONS	Check return and reason codes	HZSHCONS
	Common shared types	HZSHTYPE
	Include that pulls in all the other header files above	HZSH

For detailed information about Metal C, see *z/OS Metal C Programming Guide and Reference*. You will also want to use the sample checks in "Sample local checks" on page 106.

Sample checks

Of course you're going to read this entire chapter to understand everything you need to know about writing a check routine. But we also have what you're really looking for - samples:

- Metal C samples in z/OS UNIX file system directory /usr/lpp/bcp/samples:
  - **hzscrchc.c** - Sample Metal C remote check routine demonstrating how to issue check messages and how to issue HZSCPARS, the parameter parsing service.
  - **hzscrchk.c** - Sample Metal C remote check demonstrating how to handle parameters and use persistent data.
  - **hzscadd.c** - Sample Metal C HZSADDCHECK exit routine.
  - **hzssmake.mk** - Sample Metal C makefile to build sample health checks.
- Assembler samples in SYS1.SAMPLIB:
  - **HZSSRCHK** - Sample remote check routine.
  - **HZSSRCHC** - Sample showing the use of the HZSCPARS parameter parsing service.
  - **HZSSMSGT** - Sample message input.

You'll find all these and more check samples on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

Remote check routine basics

A check routine is a program that gathers installation information and looks for problems, and then issues the check results in messages. IBM Health Checker for z/OS writes the check exception messages as WTOs or to the message buffer. The remote check routine can run anywhere with any authority, with access granted by RACF XFACILIT class profiles.

When IBM Health Checker for z/OS calls the remote check routine, it passes the check a release code and the check can issue the HZSCHECK REQUEST=OPSTART service request to obtain a copy of the HZSPQE data area of the check. For a Metal C check, use the HZSHPQE header contained in SYS1.SIEAHDRV.H, which mirrors the HZSPQE data area mapping. The HZSPQE data area for a check contains:

- The defaults defined for the check.
- A 2K check work area
- Any installation update values.

The check routine should not update the HZSPQE data area except for the 2K check work area. See "Using the HZSPQE data area in your local check routine" on page 111.

We recommend that you keep the check routine very simple. At a high level, **your remote check will consist of:**

1. Handling of input parameters, if any, for your check when the system indicates that parameter data has changed. See "Using the check parameter parsing service (HZSCPARS)" on page 111.
2. The meat of the check - checking for potential problems on a system.
3. Issuing messages using the HZSFMSG macro ("Issuing messages in your local check routine with the HZSFMSG macro" on page 117)
4. Defining your message variables in the HZSMGB data area ("Defining the variables for your messages" on page 120)

Limit a check to looking at one setting or one potential problem. Limiting the scope of a check will make it easier for the installation using the check to:

- Resolve any exceptions that the check finds by either fixing the exception, overriding the setting, or deactivating the check.
- Set appropriate override values for check defaults such as severity or interval.

Do not set return and reason codes for your check routine. The system will return a result for you in the PQE_Result field when you use HZSFMSG REQUEST=CHECKMSG macro request (for exception messages) or the HZSFMSG REQUEST=STOP macro request (to stop the check). Do not set this field in your check routine.

Use the 2K check work area: Use the 2K check work area in field PQEChkWork for data you want to retain through check iterations for the life of the check, until the check is refreshed or deleted. Using the 2K check work area allows you to avoid obtaining additional resources for your check routine. Prior to the Init function code call, the system sets the 2K work area to zeros.

The PQEChkWork field should be the only field your check routine writes to in the HZSPQE data area. The check routine can write to the 2K PQEChkWork field in the HZSPQE data area, and the check can save the PQEChkWork user area for subsequent calls by issuing the HZSCHECK REQUEST=OPCOMPLETE. The system clears the 2K PQEChkWork user area before calling the check with the HZS_Remote_Function_InitRun release code. Changes made to any other HZSPQE fields are not saved between function codes.

Group checks for a single element or product in a single check routine. You can group multiple uniquely named checks for a single element or product in a single check routine. This can help to reduce system overhead and simplify maintenance. If you are using an HZSADDCHECK exit routine to add your local checks to the system, you should also use a single exit routine to add related checks to the system. Code your check routine to look for the entry code passed in field PQE_Entry_Code, (from the ENTRYCODE parameter on the HZSADDCK call) and pass control to processing for the check indicated. Note that the IBM Health Checker for z/OS will not verify the uniqueness of the entry codes you define for your checks.

Do not attempt to communicate between individual checks. Even though you may have placed all of your check routines in the same module, do not rely on communication between them. Each check is intended to stand by itself.

Programming considerations

Environment

For a remote check, the environment is up to you because it will be running in your address space rather than the IBM Health Checker for z/OS address space. Read the environment and requirements information for the IBM Health Checker for z/OS macros that your check issues. See Chapter 12, “IBM Health Checker for z/OS HZS macros,” on page 259.

Requirements

- Minimum authorization for your remote check task is problem state, PSW key 8-15. If the caller is APF authorized, it can connect to IBM Health Checker for z/OS without specifically defining authorization to the RACF profiles below. However, when problem state and key 8-15 and not APF authorized, or when SECCHECK=ALL is specified, the caller must be authorized for control access to any of the following:
  - XFACILIT class resource HZS.sysname.ADD
  - XFACILIT class resource HZS.sysname.checkowner.ADD
  - XFACILIT class resource HZS.sysname.checkowner.checkname.ADD
- Many installations are multilevel secure systems, and check developers must be aware of the multilevel system environment.
- The check routine must be able to handle the IBM Health Checker for z/OS release codes. See “Release codes for remote check routines” on page 145.
- Each remote check must run in its own task, even If you group remote checks together in one check routine.

Restrictions

None

Establishing a recovery routine for a check

Establishing an ESTAEX or IEAARR routine in the check routine provides recovery from errors encountered during check execution. See Writing recovery routines in *z/OS MVS Programming: Assembler Services Guide*.

If the task that issues the HZSADDCK macro defining check defaults terminates for any reason, including an abend that is not re-tried, the system treats the check as if it is deleted.

Preparing for check definition - making sure IBM Health Checker for z/OS is up and running

A remote check can only define itself when IBM Health Checker for z/OS is up and running. There are two ways to determine whether IBM Health Checker for z/OS is up and running:

- An APF-authorized remote check can use the ENFREQ LISTEN service to specify a listen exit for ENF event code 67 that tells the check routine that IBM Health Checker for z/OS is up and running. Then, when the remote check routine is assured that IBM Health Checker for z/OS is up and running, it can issue the HZSADDCK macro to define itself.
- An unauthorized remote check cannot use the ENFREQ LISTEN service, so it must periodically re-try the HZSADDCK macro until IBM Health Checker for z/OS is up and running.

Using ENF event code 67 to listen for IBM Health Checker for z/OS availability

If your remote check is authorized, it can use the ENFREQ LISTEN service to see when IBM Health Checker for z/OS is up and running, or whether a check's status has changed. On the ENFREQ service, you specify the specific event for which you would like to listen (IBM Health Checker for z/OS availability) and the listener user exit routine that is to receive control after the specified event occurs. The listener user exit specified receives control when IBM Health Checker for z/OS comes up and notifies the remote check routine, which can then define itself using HZSADDCK.

To listen for ENF event code 67, you must specify the qualifying events on the BITQUAL parameter, which specifies a 32-byte field, a hexadecimal constant, or a register containing the address of a 32-byte field containing a bit-mapped qualifier that further defines the event. The qualifiers are mapped by mapping macro HZSZENF. The defined BITQUAL values are:

Qualifier
Information type

X'80000000'
IBM Health Checker for z/OS is available. Field Enf067_BitQual_Available in the HZSZENF mapping macro.

X'40000000'
IBM Health Checker for z/OS has terminated and is not available. Field Enf067_BitQual_NotAvailable in the HZSZENF mapping macro.

X'20000000'
The status has changed for a check. Field Enf067_BitQual_StatusChange in the HZSZENF mapping macro. This might mean:

- The check completed with a different result than the last time it ran. For example, the check ran successfully after the previous check run issued an exception or vice versa.
- The check was deactivated or deleted

If you are monitoring this event, upon receiving this you would probably want to issue the HZSQUERY macro to get information about the check.

This event may not be presented if IBM Health Checker for z/OS is terminating (indicated by a X'40000000' ENF 067 event for NotAvailable).

Note that it is possible that IBM Health Checker for z/OS will not be up any longer by the time the check routine issues the HZSADDCK routine to define the check to the system. In this case, if the check was using ENFREQ to LISTEN, it should return to listening again. If the check was periodically re-trying the HZSADDCK macro, it should go on trying.

If the check routine decides it is no longer interested in knowing if IBM Health Checker for z/OS is up or not, it can issue the ENFREQ REQUEST=DELETE request to delete the listen request.

For information about ENFREQ and listener exits, see:

- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
- Listening for system events in *z/OS MVS Programming: Authorized Assembler Services Guide*

Allocate a pause element token using IEAVAPE

To aid in synchronization between the remote check task and IBM Health Checker for z/OS, you must allocate a pause element token (PET). The PET is used as follows:

- You provide the PET to the system when the check routine issues an HZSADDCK service. When the routine issues an HZSCHECK service, the system returns the PET that the check routine needs to use when it pauses.
- IBM Health Checker for z/OS uses the PET to tell the check to start running (using the IEAVRLS service)

Use the IEAVAPE service in your remote check routine to allocate a PET. Note that you can only use a PET in one pause/release cycle. Once a task is paused and released, you'll need the updated PET returned by IEAVPSE to pause the check routine next time.

You must always specify a value of IEA_UNAUTHORIZED for the auth_level parameter when your remote check issues the IEAVAPE service, even if the calling program is authorized.

See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for the IEAVAPE and IEAVPSE services and Synchronizing tasks in the *z/OS MVS Programming: Assembler Services Guide*.

Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS

A remote check does not require a separate HZSADDCHECK exit routine to identify and describe your check. All you have to do to define (identify, describe, and add) a check to IBM Health Checker for z/OS is issue the HZSADDCK macro.

IBM Health Checker for z/OS assigns a **handle** to a remote check. The handle identifies the remote check task to IBM Health Checker for z/OS for many different check functions that require coordination between the remote check and IBM Health Checker for z/OS. The handle is returned by IBM Health Checker for z/OS when the remote check routine issues the HZSADDCK service to define the check. Each time the check defines itself, the check routine, IBM Health Checker for z/OS assigns a new handle to the check routine. The check routine uses the handle to identify itself each time it starts a check iteration, issues a function code, issues a check message (HZSFMSG service) or other IBM Health Checker for z/OS service request (HZSCPARS, HZSCHECK), and completes a check iteration.

You must ensure that the remote check task has the authorization to define itself as a remote check. Authorization requires either:

- That the remote check task be APF authorized
- That the calling program has CONTROL access to the SAF resource HZS.*sysname.checkowner.checkname*.ADD in the XFACILIT class.

IBM Health Checker for z/OS processes the default values for the check from the HZSADDCK macro call, and applies any installation updates to the defaults.

Use the following guidelines in defining defaults for your check on the HZSADDCK macro call in your remote check routine:

- **The CHECKOWNER field should reflect both the company and component or product name:** For quick identification of checks, we suggest that the owner field include a company identifier and component or product name. For example, CHECKOWNER name IBMGRS reflects both the company and component that owns the check.
- **Define a meaningful CHECKNAME for your check:** Create a meaningful, descriptive name for your check. Your CHECKNAME should start with a component or product prefix so that you can easily identify where a check comes from. In addition, using the prefix ensures that all the checks for a particular component or product will be grouped together in an SDSF check display, if supported on your system. For example, IBM's virtual storage management (VSM) checks all start with VSM. (See Chapter 13, “IBM Health Checker for z/OS checks,” on page 381.)
- **Specify REMOTE=YES** to indicate that the HZSADDCK macro call comes from a remote check routine.
- **Define an output field for the remote check HANDLE:** To coordinate functions between the remote check routine and IBM Health Checker for z/OS, the system returns an identifying handle in the HANDLE parameter on the HZSADDCK macro. You must use this handle when your issue the HZSFMSG macro to issue a check message, a function code, and other processes.
- **Specify the PETOKEN parameter:** For a remote check routine, you must specify the PET returned from the IEAVAPE macro call issued previously in the check routine.
- **Using the DATE parameters:** The HZSADDCK DATE parameter specifies when the setting or value being checked was defined. This will alert customers to check the installation updates for this check. An installation update also has an associated date, and when the installation update date is older than the DATE parameter specified on HZSADDCK, the system:
  - Does not apply the update
  - Issues a message to inform the installation of the circumstance.

If you change your check, you should update the HZSADDCK DATE parameter only if you want to make sure that the installation takes a look at your check again to make sure any installation updates are still appropriate.

- **Assign a severity to your check based on the problems your check is looking for** and how critical they are. The severity you choose will determine how the system handles the exception messages that your check routine issues with the HZSFMSG service:
  - SEVERITY(HIGH) indicates that the check routine is checking for high-severity problems in an installation. All exception messages that the check issues with the HZSFMSG service will be issued to the console as critical eventual action messages.
  - SEVERITY(MEDIUM) indicates that the check is looking for problems that will degrade the performance of the system. All exception messages the check issues with HZSFMSG will be issued to the console as eventual action messages.
  - SEVERITY(LOW) indicates that the check is looking for problems that will not impact the system immediately, but that should be investigated. All exception messages the check issues with HZSFMSG will be issued to the console as informational messages.

  Installations can update the SEVERITY value in the HZSADDCHECK exit routine r using either the SEVERITY or WTOTYPE parameter in an installation update.
- **Selecting an INTERVAL and EINTERVAL for your check:** Keep the following in mind when selecting an interval for a check:
  - The INTERVAL parameter specifies how often the check will run. But you can also specify an exception interval (EINTERVAL), which lets you specify a more frequent interval for the check to run if it has raised an exception.
  - A check INTERVAL must be 1 minute or longer.
  - The specified INTERVAL or EINTERVAL time starts ticking away when a check finishes running.
- **Specify whether your check requires UNIX System Services:** Use the USS keyword to specify whether your check requires z/OS UNIX System Services. Any check that uses UNIX System Services such as DUB should specify USS=YES. If you specify USS=YES for a check, the system will run the check only when UNIX System Services are available.

A program check encountered when invoking HZSADDCK for a remote check should be handled as an expected situation that can be tried again at least once. The most likely case being that the re-try will find that IBM Health Checker for z/OS is currently stopped. There should be no dump and no recording to LOGREC.

Example of an HZSADDCK macro call for a remote check

The following example shows an **assembler** HZSADDCK macro call for a remote check:

```
HZSADDCK CHECKNAME=RNAME,
         CHECKOWNER=ROWNER,
         REMOTE=YES,
         HANDLE=RHANDLE,
         PETOKEN=RPETOKEN,
         DATE=RDATE2,
         REASON=RREASON,
         REASONLEN=RREASONLEN,
         SEVERITY=LOW,
```

```
         INTERVAL=TIMER,
         HOURS=RHOURS,MINUTES=RMINUTES,
         RETCODE=RetCode,
         RSNCODE=RsnCode
*
* Place code to check return/reason codes here
*
ROWNER     DC   CL16'IBMABC'
RNAME      DC   CL32'A_CHECK'
RDATE      DC   CL8'20060112'
RREASON    DC   CL32'Verify widgets are present.'
RREASONLEN DC   A(L'RREASON)
RHOURS     DC   H'1'
RMINUTES   DC   H'0'
         HZSZCONS              Return code information
DYNAREA  DSECT
RHANDLE    DS   CL16
RPETOKEN   DS   CL16
RRETCODE   DS   F
RRSNCODE   DS   F
         HZSADDCK MF=(L,ADDCKL),PLISTVER=MAX
```

For complete information, see "HZSADDCK macro — HZS add a check" on page 260.

Pause the remote check routine with IEAVPSE

Use the IEAVPSE service in a remote check routine to pause the check routine task after one processing phase is finished and wait for IBM Health Checker for z/OS to tell it when to resume running. When you issue IEAVPSE to pause the remote check routine, the service returns an updated pause element token (PET). You must use the updated PET the next time you pause the remote check routine.

See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for the IEAVAPE and IEAVPSE services and Synchronizing tasks in the *z/OS MVS Programming: Assembler Services Guide*.

Using HZSCHECK REQUEST=OPSTART and REQUEST=OPCOMPLETE to communicate check start and stop to IBM Health Checker for z/OS

A remote check routine must use the HZSCHECK macro REQUEST=OPSTART or OPCOMPLETE to communicate to IBM Health Checker for z/OS when the check is starting or stopping itself because the check has completed an iteration:

```
HZSCHECK REMOTE=YES,
         HANDLE=handle,
         REQUEST=OPSTART or REQUEST=OPCOMPLETE
```

For information, see "HZSCHECK macro — HZS Check command request" on page 279.

Using the check parameter parsing service (HZSCPARS)

If your local or remote check includes parameters, you can use the HZSCPARS check parameter parsing service to parse parameters. You can use HZSCPARS in either an assembler or Metal C check routine. When HZSCPARS finds a parameter error, it issues appropriate error messages for you using the REASON=PARS*xxxx* reason values on the HZSFMSG macro. This means that your check routine does

not have to issue error messages for parameter errors. See "HZSFMSG macro — Issue a formatted check message" on page 307 for explanations of all the REASON=PARS*xxxx* values.

If you are using HZSCPARS for a check that expects a parameter or parameters but does not get one, HZSCPARS considers this an error and issues an error message.

Your check routine can also use REASON=PARS*xxxx* on HZSFMSG REQUEST=HZSMSG to issue parsing error messages in the course of doing their own parameter parsing.

You will use HZSCPARS REQUEST=PARSE in your check routine to allocate a parameter area, mapped by mapping macro HZSZCPAR, that describes the parsed parameters for the check. You can free this parameter area using HZSCPARS REQUEST=FREE . For a local check, if you do not free the parameter area, the system will delete the parameter area upon return from the check routine.

See "HZSCPARS macro — HZS Check Parameter Parsing" on page 294 for complete information.

For an example of using the HZSCPARS macro in a check routine, see sample HZSSRCHC, which you can fine in SYS1.SAMPLIB or on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

Note that your check routine must still issue the HZSFMSG REQUEST=STOP request when HZSCPARS finds a parameter error - see "Function codes for local check routines" on page 113 and "Release codes for remote check routines" on page 145.

Using the HZSPQE data area in your remote check routine

The HZSPQE data area contains all the information a check routine needs, including the defaults defined in the HZSADDCHECK exit routine and any installation overrides to those defaults. The HZSPQE contains a number of sections, but some of the most important are:

- PQEChkParms, which shows the current values for the check.
- PQEChkWork, which is the 2K check work area.

The table below shows the structure and some of the most important fields in the HZSPQE data area.

Table 15. Important fields in the HZSPQE data area for a remote check routine

Field name	Meaning
PqeChkInfo section - contains the defaults defined in the HZSADDCHECK exit routine for the check	
PQE_Entry_Code	Contains the identifier (entry code) assigned for the check in the HZSADDCHECK exit routine. The entry code is used when a check routine contains multiple checks.
PqeChkParms section - contains the installation overrides for default parameters for the check from HZSPRMxx and the Modify command (F *hzsproc*).	
PQE_LookAtParms	A bit indicating that the parameters have changed. If this bit is on, the check routine should read the PQE_ParmArea and PQE_PARMLen fields in PQE_Function_Code_Check processing.
PQE_Verbose	A byte indicating whether the check is in verbose mode.

Table 15. Important fields in the HZSPQE data area for a remote check routine (continued)

Field name	Meaning
PQE_Debug	A byte indicating whether the check is in debug mode.
PQE_ParmLen	Contains the length of the parameter area. Quotes surrounding the PARMS value in an operator command or HZSPRMxx statement are not included in the resulting length. For example, PARMS('THE_PARM') will result in a length of 8.
PQE_ParmArea	The area containing the user parameters. Quotes surrounding the PARMS value in an operator command or HZSPRMxx statement are not included.

PQEChkWork section - 2K check work area used and mapped by the check routine as needed. The system zeros the 2K user PQEChkWork user area before calling the check with function code PQE_Function_Code_Init. A check routine can both write and read from this field, and the system will retain this information for subsequent calls to the check routine. Changes made to any other HZSPQE fields are not saved between function calls.

Release codes for remote check routines

There are two kinds of release codes for remote checks:

- **Pre-defined release codes:** When IBM Health Checker for z/OS unpauses a remote check task that issued the IEAVPSE service, the remote check receives a pre-defined release code that tells the remote check routine why it was called. Remote checks should always check the release code on being unpaused (IEAVRLS service) by the system. The equates for the release codes are provided in the HZSZCONS mapping macro. The release codes are similar to the function codes that local checks use and are described in Table 16 on page 146.
- **Application-defined release codes:** An application can unpause a remote check with its own, application-defined release code. Release codes in the range X'C00000' to X'FFFFFF' are reserved for this use and will not conflict with the pre-defined IBM Health Checker for z/OS release codes. See “Ending a check that is coupled with an application” on page 147for an example for using your own release codes.

For remote checks, the pre-defined IBM Health Checker for z/OS release codes include:

- The INITRUN function is invoked once for the life of the check (which lasts until the check is deleted or deactivated), to do initialization processing and run the check for the first time. In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_InitRun.
- The RUN function is called to indicate that the remote check should run and do check cleanup after the initial run (INITRUN) of the check, at the specified interval. In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_Run.
- Delete functions:
  - The DELETE function indicates that a user issued a DELETE request on either UPDATE or POLICY STMT in the HZSPRMxx parmlib member on or a F *hzsproc* command. In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_Delete.
  - The DELETE_REFRESH function indicates that IBM Health Checker for z/OS deleted the check as part of refresh processing. In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_DeleteRefresh.
  - The DELETE_TERM function indicates that the system deleted the check when the IBM Health Checker for z/OS address space went down. In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_DeleteTerm.

See “Ending a check that is coupled with an application” on page 147.

- The RESTART function indicates that IBM Health Checker for z/OS has been restarted so that the remote check can re-define itself (using the HZSADDCK macro). In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_Restart.
- The DEACTIVATE function indicates that the remote check has been deactivated. In the HZSZCONS mapping macro, this release code is HZS_Remote_Function_Deactivate.

The following table summarizes the release codes for remote checks, showing what the check should do for each one and when IBM Health Checker for z/OS invokes them:

Table 16. Summary of release codes for remote checks

Release code	Check and system actions	When is it invoked?
INITRUN and RUN	**What should the check do?** The check routine should: • Issue HZSCHECK REQUEST=OPSTART. • Validate that the environment is suitable for the check. If it is not, issue the HZSFMSG REQUEST=STOP macro to stop the check. • Check to see if the PQE_LookatParm bit is set on, indicating either that this is the first iteration of the check, or that the installation has changed the check parameters since the last iteration. If the bit is on, validate the parameters in the PQE_ParmArea field of the HZSPQE data area. If the check finds bad installation parameters, it should: 1. Issue an error message indicating what the problem is. 2. Issue the HZSFMSG REQUEST=STOP,REASON=BADPARM macro request to stop the check. See “HZSFMSG macro — Issue a formatted check message” on page 307. • For INITRUN, obtain any first-time-called resources needed for the check. • Perform the check, including issuing exception messages (HZSFMSG service), reports and other messages. You can issue a particular message multiple times in a check routine. • Clean up anything that you want cleaned between check iterations. • Issue HZSCHECK REQUEST=OPCOMPLETE • Issue IEAVPSE to pause the check and to look at the release code upon being released.. **What does the system do?** The system does the following setup steps to prepare for multiple check iteration: • Initializes the HZSPQE data area with default and override values for the check. • Passes the default and installation overrides to the check in the HZSPQE data area for the check. • Obtains 2K of workarea storage mapped by field PQEChkWork. This storage is zeroed for Init processing and lasts for the life of the check.	• Refresh • When a check is added • When a check transitions to the active enabled state • At specified check interval • When a check parameter changes
DELETE	**What should the check do?** The check should: • Issue HZSCHECK REQUEST=OPSTART • Free all storage resources, including the message table, and any storage obtained during INITRUN or RUN • Issue HZSCHECK REQUEST=OPCOMPLETE • The check should deallocate its pause element token using IEAVDPE and terminate. **What does the system do?** The system stops calling the check.	A user deletes the check using F *hzsproc* or the HZSPRMxx parmlib member.

Table 16. Summary of release codes for remote checks (continued)

Release code	Check and system actions	When is it invoked?
DELETE_ REFRESH	**What should the check do?** IBM Health Checker for z/OS deleted the check as part of refresh processing. The check should: • Issue HZSCHECK REQUEST=OPSTART • Free all storage resources, including the message table, and any storage obtained during INITRUN or RUN • Issue HZSCHECK REQUEST=OPCOMPLETE • Load the check routine and message table into storage that will persist as long as the check needs them. • Issue the IEAVAPE service to allocate a PET, if it had not already obtained one. • Issue HZSADDCK to re-define itself and get a new handle. • Issue IEAVPSE to pause the check and to look at the release code upon being released. **What does the system do?** The system stops calling the check and creates the HZSDPQE for the remote check. Then, upon receiving the HZSADDCK redefining the check, it starts calling the check again.	• Refresh
DELETE_ TERM	**What should the check do?** The system deleted the check when the IBM Health Checker for z/OS address space went down. The check should: • Issue HZSCHECK REQUEST=OPSTART • Free all storage resources, including the message table, and any storage obtained during INITRUN or RUN • Issue HZSCHECK REQUEST=OPCOMPLETE • Issue IEAVPSE to pause the check and to look at the release code upon being released. **What does the system do?** The system stops calling the check.	• When the IBM Health Checker for z/OS address space goes down
RESTART	**What should the check do?** The IBM Health Checker for z/OS address space has been restarted. (This release code does **not** indicate that the check is restarting.) The check should: • Load the check routine and message table into storage that will persist as long as the check needs them. • Issue the IEAVAPE to allocate a PET, if it had not already obtained one. • Issue HZSADDCK to re-define itself and get a new handle. • Issue IEAVPSE to pause the check and to look at the release code upon being released. **What does the system do?** Continues to initialize IBM Health Checker for z/OS.	• When the IBM Health Checker for z/OS address space is restarted
DEACTIVATE	**What should the check do?** A user deactivated the check using the F *hzsproc* command or the HZSPRMxx parmlib member. The check routine should: • Issue HZSCHECK REQUEST=OPSTART • Free all storage resources, including the message table, and any storage obtained during INITRUN or RUN • Issue HZSCHECK REQUEST=OPCOMPLETE • Issue IEAVPSE to pause the check and to look at the release code upon being released. **What does the system do?** The system stops calling the check.	• Refresh • When the check transitions out of the active enabled state. For example, when the check issues HZSFMSG with the STOP request. • When the IBM Health Checker for z/OS address space stops.

Ending a check that is coupled with an application

If you write a check that is coupled with an application or product, you should code it so that the check is stopped and deleted when the application shuts down. In this section, we recommend a protocol to do this. Following this protocol is particularly important when IBM Health Checker for z/OS is ending at the same time as the application. Our protocol is as follows:

1. Define a release code for your application in the range X'C00000' to X'FFFFFF' (see "Release codes for remote check routines" on page 145), which you can use to signal a check routine to stop and delete itself.
2. Code your remote check so that when it receives a DELETE_TERM release code, the check records that fact for use by the process controlling application termination.
3. When an application begins to terminate and the remote check is already paused after receiving a DELETE_TERM release code, release the check using the application release code defined in Step 1 above.
4. When an application begins to terminate and the remote check is **not** paused by a DELETE_TERM release code, issue the HZSCHECK REQUEST=DELETE service to release the remote check with the DELETE release code.

   It is possible that a timing collision can occur in this process, in which the remote check is just about to be released with DELETE_TERM at the same time that the HZSCHECK REQUEST=DELETE service is issued. In this case, consider using a loop of STIMER or STIMERM invocations to wait in short intervals to see if the DELTE_TERM or DELETE has completed, and continue with application termination when one of those has completed.

Creating and using data saved between restarts

Your check can use the HZSPDATA data set for persistent data. Persistent data is data that you want to save between restarts of either the system or IBM Health Checker for z/OS. When you issue the HZSPWRIT macro to write persistent data, the system saves data from two IPLs in HZSPDATA, the current IPL and the IPL prior to the current. Then, for each IPL, HZSPDATA contains two instances of data - one for the first iteration of the check and another for the most recent iteration of the check that wrote data to HZSPDATA. The first instance does not change for the life of the IPL, but the second instance is replaced each time a check writes data to the HZSPDATA data set.

You can read data from the HZSPDATA data set using the HZSPREAD macro. Commonly, checks use HZSPDATA to compare current data to the data saved in the HZSPDATA data set from one of the saved IPLs.

We have a couple of tips for you in using HZSPREAD and HZSPWRIT macros to read and write persistent data:

- Before you try to work with the persistent data that you read from the HZSPDATA data set, make sure your code checks for the following HZSPREAD return codes:
  - Return code 8, reason code X'xxxx082D', equate symbol HzspreadRsn_NoMatch indicates that no persistent data exists for this check.
  - Return code 8, reason code X'xxxx0830', equate symbol HzspreadRsn_DataDoesNotExist indicates that there is persistent data saved for this check, but not for the requested IPL.
- Tips for using HZSPWRIT:
  - You cannot delete data from the HZSPDATA data set once you have written it there. You can only replace the data in the current IPL instance in HZSPDATA.
  - You cannot write a null record to HZSPDATA.
  - You can issue multiple HZSPWRIT requests in a single check iteration. If the check iteration completes normally (returns to its caller), all of the data accumulated by HZSPWRIT requests for that iteration are written to

HZSPDATA. If the check iteration does not complete normally, none of the data provided on HZSPWRIT requests for that check iteration is written to HZSPDATA.

Gotcha: After your check writes data to the HZSPDATA data set using HZSPWRIT, it takes one hour before data is actually hardened. That means that if the installation restarts IBM Health Checker for z/OS before an hour or re-IPL less than an hour has elapsed since the last HZSPWRIT, the data will not be saved in the HZSPDATA data set. IBM Health Checker for z/OS operates this way so that if a problem such as the following occurs, the system does not retain the data in the HZSPDATA data set:

- The check iteration completes with an abend
- A remote check iteration is unsuccessful
- An invocation of HZSPWRIT is unsuccessful

Note that an unsuccessful check iteration or HZSPWRIT invocation does not have any correlation to whether or not the check detected one or more exceptions.

Planning for persistent data: Sample HZSALLCP in SYS1.SAMPLIB shows how to allocate and initialize the HZSPDATA data set. When you are allocating space for the HZSPDATA data set, keep in mind that in "Allocate the HZSPDATA data set to save check data between restarts" on page 10, we tell customers to define the HZSPDATA data set with a logical record length of 4096. You must plan for four sets of data: for each of the two instances for both the current and previous IPLs.

Authorization for HZSPDATA: You can define RACF profiles in the XFACILIT class for resources accessing the HZSPDATA.

Note that checks reading from or writing to the HZSPDATA data set must be both APF authorized and also have indicated not to do security checks, or they must have the appropriate access (READ or UPDATE) to either of the following:

- XFACILIT class resource HZS.sysname.checkowner.PDATA
- XFACILIT class resource HZS.sysname.checkowner.checkname.PDATA

See "HZSPREAD macro — Read Check Persistent Data" on page 339 and "HZSPWRIT macro — Write Check Persistent Data" on page 349 for information about authorization for checks to the HZSPDATA data set.

The following example shows how you might define a RACF profile for read or update access to HZSPDATA data set for a check:

```
RDEFINE XFACILIT HZS.sysname.checkowner.checkname.PDATA UACC(NONE)
PERMIT HZS.sysname.checkowner.checkname.PDATA CLASS(XFACILIT) ID(hzspdid) ACCESS(READ|UPDATE)
SETROPTS CLASSACT(XFACILIT) RACLIST(XFACILIT)
```

If you have already RACLISTed the XFACILIT or FACILITY class, which you probably have if you have IBM Health Checker for z/OS set up, just use the REFRESH parameter on the SETROPTS statement:

```
SETROPTS RACLIST(XFACILIT) REFRESH
```

Use the SECCHECK(UNAUTH|ALL) parameter in your code to specify whether you want the system to verify the security for writing to or reading from HZSPDATA. See "HZSPWRIT macro — Write Check Persistent Data" on page 349 and "HZSPREAD macro — Read Check Persistent Data" on page 339.

Issuing messages in your remote check routine with the HZSFMSG macro

To issue a message with check results in your check routine, you must use the HZSFMSG macro ("HZSFMSG macro — Issue a formatted check message" on page 307), which you can issue in either an assembler or Metal C check routine.

This section only covers using the HZSFMSG macro to issue a message, but a message also consists of a few other ingredients. When your check runs, the system assembles the message from the following:

- The actual text and explanation for your check messages are defined in your message table, see "Issuing messages in your local check routine with the HZSFMSG macro" on page 117.
- The variables for your check messages are defined in the HZSMGB data area from your check routine. See "Defining the variables for your messages" on page 120.

Note that you can omit the message table and issue messages directly from the check routine in one of the following ways:

- For local or remote checks, use HZSFMSG REQUEST=DIRECTMSG. See "HZSFMSG macro — Issue a formatted check message" on page 307.
- For REXX checks, use the REQUEST='DIRECTMSG' subfunction of the HZSLFMSG function. See "Input variables for HZSLFMSG_REQUEST='DIRECTMSG'" on page 242.

You can issue the following kinds of messages in your check routine:

- Exception messages and other check results messages (CHECKMSG or DIRECTMSG request). For an overview of the various message types, see Table 20 on page 216.
- IBM Health Checker for z/OS messages (HZSMSG request)
- IBM Health Checker for z/OS messages that indicate that the check is stopped (STOP request). If your check routine issues HZSFMSG with the STOP request, it prompts the system to call the delete function code for the check.

You can issue a particular message multiple times in a single iteration of a check - a check routine should always issue an exception message to report an error.

For a remote check, the HZSFMSG macro call must:

- Specify REMOTE=YES
- Specify the handle that identifies the check to IBM Health Checker for z/OS on the HANDLE parameter. The system assigns and returns the handle to the remote check when the check issues the HZSADDCHK macro to define the check to the system. See "Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS" on page 140.
- Specify the location of the message table for the check in the MSGTABLE parameter. (Local checks do not have to specify the location of the message table because both the check and the message table are in the IBM Health Checker for z/OS address space.)

  Note that a remote check must also load the message table into storage.

For example, a remote check might issue a check exception message with the following HZSFMSG macro call:

```
HZSFMSG REQUEST=CHECKMSG,MGBADDR=Addr_Of_MGB1,
      MGBFORMAT=1,
      REMOTE=YES,HANDLE=CK_Handle,
      MsgTable=Addr_Of_MsgTable,
      MF=(E,HZSFMSG_List)
```

Check messages are important because they report the results of the check to an installation. Each check should issue at least:

- One or more messages for any exception found to the setting the check is looking for.
- A message indicating that no exceptions were found, when appropriate.

If an HZSFMSG macro call is incorrect, the system issues system abend X'290' with a unique reason code and creates a logrec error record. The system checks the following for each HZSFMSG call:

- To see that the HZSMGB data area (input to checks describing message identifiers and variables) is complete
- That the message is in the message table
- That the number of inserts provided on the call exactly matches the number required to complete the message
- That each variable definition is between 1-256 characters long

The reason codes for system abend X'290' describe the message error. See *z/OS MVS System Codes*.

HZSFMSG updates the PQE_Result field in the HZSPQE as follows:

- For a specified severity of HIGH, the system sets the check result to 12
- For a specified severity of MEDIUM, the system sets the check result to 8
- For a specified severity of LOW, the system sets the check result to 4

PQE_Result is set to 0 when the check is called. See “Examples” on page 337.

For information on coding the message texts and explanation for messages, see “Issuing messages in your local check routine with the HZSFMSG macro” on page 117.

Reporting check exceptions

When a check detects a system condition or setting that runs counter to the values that the check is looking for, the check should issue an exception message to report the exception. For an exception message, the system displays both the message text and the entire message explanation in the message buffer. The message should include a detailed explanation of the error **and** the appropriate action that the installation should take to resolve the condition. If you are writing a check that checks for a setting that conflicts with the default for the setting, you should include in your check output information about **why** the check user is getting an exception message for a default setting.

Along with an exception message, IBM Health Checker for z/OS will issue a line showing the severity and the return code for the check. The check will continue to run at the defined intervals, reporting the exception each time until the exception condition is resolved.

The following example shows an exception message issued to the message buffer:

```
CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
START TIME: 05/25/2005 09:42:56.690844
CHECK DATE: 20040703  CHECK SEVERITY: HIGH

* High Severity Exception *

IRRH204E The RACF_SENSITIVE_RESOURCES check has found one or
more potential errors in the security controls on this system.

  Explanation:  The RACF security configuration check has found one or
    more potential errors with the system protection mechanisms.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator and the system auditor.

  System Programmer Response:  Examine the report that was produced by
    the RACF check. Any data set which has an "E" in the "S" (Status)
    column has excessive authority allowed to the data set. That
    authority may come from a universal access (UACC) or ID(*) access
    list entry which is too permissive, or if the profile is in WARNING
    mode. If there is no profile, then PROTECTALL(FAIL) is not in
    effect. Any data set which has a "V" in the "S" (Status) field is
    not on the indicated volume. Remove these data sets from the list
    or allocate the data sets on the volume.

    Asterisks ("****") in the UACC, WARN, and ID(*) columns indicate
    that there is no RACF profile protecting the data set. Data sets
    which do not have a RACF profile are flagged as exceptions, unless
    SETROPTS PROTECTALL(FAIL) is in effect for the system.

    If a valid user ID was specified as a parameter to the check, that
    user's authority to the data set is checked. If the user has an
    excessive authority to the data set, that is indicated in the USER
    column. For example, if the user has ALTER authority to an
    APF-authorized data set, the USER column contains "<Read" to
    indicate that the user has more than READ authority to the data set.

  Problem Determination:  See the RACF System Programmer's Guide and
    the RACF Auditor's Guide for information on the proper controls for
    your system.

  Source:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Reference Documentation:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Automation:  None.

  Check Reason:  Sensitive resources should be protected.

END TIME: 05/25/2005 09:43:13.717882  STATUS: EXCEPTION-HIGH
    APF-authorized data set, the USER column contains "
```

The **Check Reason:** field display the default reason in an exception message without installation parameter overrides.

See "Issuing a REXX check exception message" for an example of how to issue an exception message from a REXX check.

Example - Issuing a DIRECTMSG message for a REXX check: For a check that has no message table associated with it, you can issue a check message directly from the check routine, as shown in the example below. REXX sample check SYS1.SAMPLIB(HZSSXCHN) also shows DIRECTMSG calls.

```
/* Set up exception message input for HZSLFMSG */
/* Required input variables: */
HZSLFMSG_REQUEST='DIRECTMSG'
HZSLFMSG_REASON='CHECKEXCEPTION'
HZSLFMSG_DIRECTMSG_ID='UTHH003E'
HZSLFMSG_DIRECTMSG_TEXT='Brief exception summary'
/* Optional input variables: */
HZSLFMSG_DIRECTMSG.EXPL='The exception explanation for UTHR003E'
HZSLFMSG_DIRECTMSG.AUTOMATION='Automation text for UTHR003E'
HZSLFMSG_DIRECTMSG.SOURCE='Source text for UTHR003E'
/* Call HZSLFMSG */
HZSLFMSG_RC = HZSLFMSG()

/* Set up report message input for HZSLFMSG */
HZSLFMSG_REQUEST='DIRECTMSG'
HZSLFMSG_REASON='CHECKREPORT'
HZSLFMSG_DIRECTMSG_TEXT='Single line report message'
/* Call HZSLFMSG */
HZSLFMSG_RC = HZSLFMSG()
```

Defining the variables for your messages

The variable information for your check messages is defined in the HZSMGB data area by your check routine. The check routine defines information about variables and points to the HZSMGB data area for variable values. For Metal C check routines, the HZSMGB data area layout is mirrored in the HZSHMGB header.

There are two HZSMGB formats you can use to map your keywords:

- **MGBFORMAT=0**: Requires you to point to a separately defined area in storage where the length and value of the variable are defined, mapped by MGB_InsertD. See “Using default HZSMGB data area format (MGBFORMAT=0)” on page 121.
- **MGBFORMAT=1**: Allows you to specify the length and address of the variables in HZSMGB fields MGB1_MsgInsertDesc_Length and MGB1_MsgInsertDesc_Addr in the MGB1_MsgInsertDesc mapping. See “Using HZSMGB data area format MGBFORMAT=1” on page 123.

Figure 14 on page 201 shows how messages with variables get resolved at check runtime.

Use the following guidelines in defining variables for your messages:

Match up the number of variables in the HZSMGB data area and the message table, because if you end up with a mismatch, your check will abend when it issues the HZSFMSG macro to issue the message. Look in the logrec error record or *z/OS MVS System Codes* to find the description of the reason code issued with the abend.

To keep text on the same line, replace blank characters, X'40', with the required blank character X'44'.

If I use the same variable twice in a message, do I have to define it twice in the HZSMGB data area? Yes, every time you use a variable, even if you use the same variable several times in the same message, you must point to separate entries in the MGB_Inserts field for each variable instance. However, each of the entries for an identical variable can point to the same area in storage where the variable length and value are specified for the variable.

Can I build the HZSMGB information for all my check messages once at initialization and then reuse them whenever the check runs? Tempting idea, but no. The problem with this method is that there's no guarantee that the HZSPQE data area for the check will be in the same place for any given run of your check. Although the contents of the PQEChkWork section are the same for every run of the check, it's location is not. Thus if you try to point within your PQEChkWork area for variable information, the offset will be the same, but the full address probably will not be.

On the other hand, if you are pointing into either your check routine module or an area that you GETMAINed at initialization to build your HZSMGB data area, those areas will stay the same, and so the build once/use multiple times approach might work. But this is a tricky maneuver.

In the HZSMGB data area, variables do not have variable names. You insert the length (MGB_MsgILen field) and value (MGB_MsgIVal field) for a variable without using the variable name you use in the check routine.

Can I have a null variable? You can indeed have a null variable by defining a variable length of zero in the MGB_MsgILen field.

What happens if I make a mistake updating HZSMGB? If you make a mistake while updating HZSMGB so that your variable values are not compatible with the variable attributes in the message output at check runtime, your check will most likely abend with system abend code X'290' and a reason code that describes the error. The system also writes a record to SYS1.LOGREC that provides additional detail in the variable recording area (VRA).

Using default HZSMGB data area format (MGBFORMAT=0)

Figure 8 on page 122 shows an example of how you define the message variables in your check routine:

1 shows an example of defining the message number in the MGB_MessageNumber.

2 shows an example of filling in the MGB_InsertCnt field with the number of variables for your message.

3 shows an example of putting the address of one variable into the MGB_Inserts field. This address points to the area in storage where the length and value of the variable are defined, mapped by MGB_InsertD.

4 shows an example of defining the length and value of the variable in the MGB_MsgILen and MGB_MsgIVal fields for the variable in storage. These fields are in the MGB_InsertD mapping.

5 shows an example of issuing a message. Note that this example shows a local message. **For a remote check**, the HZSFMSG macro must include the REMOTE=YES, HANDLE=*handle*, and MSGTABLE=*msgtable* parameters.

6 shows how the variable address, length, and value are defined to be stored in the HZSMGB data area or in storage.

7 shows an example of creating an area big enough in the HZSMGB for the information about all your variables. To create enough room for all your variables, use the formula HZSMGB_LEN + (*n*-1)*L'MGB_inserts where *n* is the number of inserts. HZSMGB_LEN by itself will provide room for only one insert.

Figure 8 on page 122 shows check routine code that defines variable data in the HZSMGB:

```
************************************************************
* Issue a message with two inserts                         *
************************************************************
         SYSSTATE ARCHLVL=1
* save regs, get dynamic storage, chain saveareas, set usings
         LA    2,TheMGBArea
         ST    2,TheMGBAddr
         USING HZSMGB,2
        1  MVC   MGB_MessageNumber,=F'1' Message 1
        2  MVC   MGB_insert_cnt,=F'2'  Two inserts
         LA    3,Insert1Area         Address of first insert
        3  ST    3,MGB_Inserts         Save insert address
         LA    3,Insert2Area         Address of second insert
         USING MGB_MsgInsertD,3
        4  MVC   MGB_MsgILen,=AL2(L'Insert2Val)  Insert length
         MVC   MGB_MsgIVal(L'Insert2Val),MyMod Insert value
         DROP  3
         ST    3,MGB_Inserts+4       Save insert address
        5  HZSFMSG  REQUEST=CHECKMSG,MGBADDR=TheMGBAddr,       *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,FMSGL)
         DROP  2
*
* Place code to check return/reason codes here
*
* free dynamic storage, restore regs
         BR 14
MyMod    DC   CL8'MYMODULE'  6
* Area for first insert
Insert1Area DS 0H
Insert1Len  DC AL2(L'Insert1Val)
Insert1Val  DC C'CSA   '
         LTORG ,
         HZSZCONS ,            Return code information
         HZSMGB   ,            Insert mapping
DYNAREA  DSECT
LRETCODE DS   F
LRSNCODE DS   F
* Area for 2 inserts (HZSMGB_LEN accounts for one, so
* we add one more "length of MGB_Inserts")
TheMGBAddr DS A   7
TheMGBArea DS CL(HZSMGB_LEN+1*L'MGB_Inserts)
* Area for second insert
Insert2Area DS 0H
Insert2Len  DS AL2(L'Insert2Val)
Insert2Val  DC X'00950000'
         HZSFMSG MF=(L,FMSGL),PLISTVER=MAX
DYNAREA_LEN EQU *-DYNAREA
```

Figure 10. Example of issuing a message with variables in an assembler check

Important fields in the HZSMGB data area include:

Table 17. Important MGBFORMAT=0 fields in the HZSMGB data area for check message variables

Field name	Meaning
MGB_MessageNumber MGB_ID	Fullword field containing the value identifying each message. These fields are the same - there are two names for this field. This field corresponds to the xreftext value for each message. For example, the xreftext value for a message is coded as follows: `<msgnum xreftext="001">TESTMSG1I</msgnum>`
MGB_InsertCnt	Fullword field containing the number of variables (or inserts) to follow.
MGB_Inserts MGB_InsertAddr	These fields are the same - there are two names for this field. This field contains an array of pointers, each of which contains the address in storage of an area for a specific variable. This area is mapped by Mgb_MsgInsertD.

Table 17. Important MGBFORMAT=0 fields in the HZSMGB data area for check message variables (continued)

Field name	Meaning
MGB_MsgInsertD	A structure in the HZSMGB data area that describes the length and value of the variable: • MGB_MsgILen, which is a 2 byte field containing the length of the variable. • MGB_MsgIVal, which contains the value of the variable.

Using HZSMGB data area format MGBFORMAT=1

1 shows an example of defining the message number in the MGB1_MessageNumber field.

2 shows an example of filling in the MGB1_Insert_Cnt field with the number of variables for your message.

3 shows examples of defining the length and address of the variable in the MGB1_MsgInsertDesc_Length and MGB1_MsgInsertDesc_Addr fields for the variable in storage. These fields are in the MGB1_MsgInsertDesc mapping.

4 shows an example of issuing a message. Note that this example shows a local message. **For a remote check**, the HZSFMSG macro must include the REMOTE=YES, HANDLE=*handle*, and MSGTABLE=*msgtable* parameters.

5 shows how the variable address, length, and value are defined to be stored in the HZSMGB data area or in storage.

6 shows an example of creating an area big enough in the HZSMGB1 for the information about all your variables. To create enough room for all your variables, use the formula HZSMGB1_LEN1 + (n)*MGB1_MsgInsertDesc_Len where n is the number of inserts.

Figure 9 on page 124 shows check routine code that defines variable data in the HZSMGB data area using MGBFORMAT=1:

```
************************************************************
* Issue a message with two inserts                         *
************************************************************
         SYSSTATE ARCHLVL=2
* save regs, get dynamic storage, chain saveareas, set usings
         LA 2,TheMGBArea
         ST 2,TheMGBAddr
         USING HZSMGB1,2 1    MVC   MGB1_MessageNumber,=F'1' Message 1 2    MVC   MGB1_insert_cnt,=F'2' Two inserts
         DROP  2
         PUSH  USING
         USING MGB1_MsgInsertDesc,TheMSGInsertDesc1 3    MVC   MGB1_MsgInsertDesc_Length,=AL2(L'Insert1Val) Insert length
         LA    15,Insert1Val
         ST    15,MGB1_MsgInsertDesc_Addr  Insert address
         POP   USING
         PUSH  USING
         USING MGB1_MsgInsertDesc,TheMGBInsertDesc2
         MVC   MGB1_MsgInsertDesc_Length,=AL2(L'Insert2Val) Insert length
         LA    15,Insert2Val
         ST    15,MGB1_MsgInsertDesc_Addr  Insert address
         POP   USING 4      HZSFMSG REQUEST=CHECKMSG,MGBADDR=TheMGBAddr,                    *
               MGBFORMAT=1,                                          *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,                    *
               MF=(E,FMSGL)
*
* Place code to check return/reason codes here
*
* free dynamic storage, restore regs
         BR    14 5
* Area for first insert
Insert1Val DC C'CSA '
* Area for second insert
Insert2Val DC X'00950000'
         LTORG ,
         HZSZCONS , Return code information
         HZSMGB , Insert mapping
DYNAREA  DSECT
LRETCODE DS F
LRSNCODE DS F
TheMGBAddr DS A
* Area for 2 inserts     6
TheMGBArea     DS CL(HZSMGB_LEN1)
TheMSGInsertDesc1  DS CL(MGB1_MsgInsertDesc_Len)
TheMSGInsertDesc2  DS CL(MGB1_MsgInsertDesc_Len)
HZSFMSG  MF=(L,FMSGL),PLISTVER=MAX
DYNAREA_LEN EQU *-DYNAREA
```

Figure 11. Example of issuing a message with variables using MGBFORMAT=1

Important MGBFORMAT=1 fields in the HZSMGB data area include:

Table 18. Important MGBFORMAT=1 fields in the HZSMGB data area for check message variables

Field name	Meaning
MGB1_MessageNumber MGB1_ID	Fullword field containing the value identifying each message. These fields are the same - there are two names for this field. This field corresponds to the xreftext value for each message. For example, the xreftext value for a message is coded as follows: `<msgnum xreftext="001">TESTMSG1I</msgnum>`
MGB1_InsertCnt	Fullword field containing the number of variables (or inserts) to follow.
MGB1_MsgInsert Desc_Length	The length of the variable. For a null variable, use a length of zero.
MGB1_MsgInsert Desc_Addr	The address of the variable. For a null variable, you need not set this field.

Writing a check with dynamic severity levels

You can create your check routine to issue check exception messages with a dynamically varying severity level, giving users more control over how exception messages are issued and handled. For example, you might use the dynamic severity function for checks that inspect a system setting value and compare it against a threshold. As the value approaches the high threshold, the check can vary the severity of the exception, depending on how close to the threshold the value is.

For information on how users work with checks enabled with dynamic severity, see “Customizing check exceptions with dynamically varying severity” on page 31

How to enable a check for dynamic severity: In order to allow the customer to use the dynamic severity function do the following:

1. Define check parameters that let the check know what severity exception to issue. For example, if your check looks for parameter 'p', define the following parameters:
   - p_HIGH
   - p_MED
   - p_LOW
   - p_NONE

   In addition, define the corresponding short forms of the parameter, for example for our case, you would define p_H, p_M, p_L, and p_N parameters. And, because it's always likely that customers could forget the exact parameter specifications, try to build in the obvious mistakes. For example, a customer might try p_HI when the correct parameter is p_HIGH, so we recommend that you also support p_HI and p_NO (as a short version of p_NONE).

   We have the following tips for you in defining dynamic severity check parameters: Note that the parameter names you create cannot exceed the 16 character parameter length limit for the ADD/UPDATE CHECK interface!
   - Of course, the customer should not specify multiple forms for a given severity, but it is not worth while enforcing that in your check routine. Instead, program your check to accept the longest parameter. In other words, if a customer specifies both p_HIGH and p_H parameters for a check, your check should use the value from p_HIGH.
   - If you are writing a new check that exploits dynamic severity, we recommend that you use the suffixed parameters (p_HIGH, p_MED, p_LOW and so on) and do not code a non-suffixed, non-dynamic version of the 'p' parameter.

     On the other hand, if you are upgrading an existing check to use dynamic severity, keep the plain non-suffixed parameter and add the _HIGH, _MED, _LOW, _NONE variants as well. Then make sure to code your check to use the rule that if any of the dynamic severity variants are present, then the non-dynamic severity variant is ignored. That way, you don't have to worry about interpreting cases where the customer specifies both a non-dynamic 'p' version of the parameter and a dynamic p_HIGH version.

     If you are adding dynamic severity parameters for an old check, and none of the existing parameters allow an underscore either within the parameter name or value, you can code your check to assume that the customer is using a dynamic severity specification if you find an underscore within the parameter string.
   - Code your check so that the customer does not need to specify a severity for all parameters. They should be able to specify just the ones that they want.
2. Code your dynamic severity check to issue exception messages with the SEVERITY parameter on the HZSFMSG service or using REXX function HZSLFMSG_SEVERITY, as follows:
   - If no dynamic severity parameter is provided and the criterion for an exception based on the parameter is met, issue a severity-system message
   - Else if a p_HIGH parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-high message

- Else if a p_MED parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-medium message
- Else if a p_LOW parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-low message
- Else if a p_NONE parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-none message

Note that the severity specified on HZSFMSG or HZSLFMSG_SEVERITY overrides the default severity defined for the check when it was added.

3. Add the check with ALLOWDYNSEV=YES specified. See the ADD or ADDREPLACE CHECK paramters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68 or "HZSADDCK macro — HZS add a check" on page 260.

How dynamic severity and the SEVERITY in the check definition interact: As we mention above, a check using dynamic severity overrides the severity specified in the check definition. So that was easy. But just to keep things interesting, note that the implicit WTO handling of exception messages that is derived from the severity in either the check definition or the dynamic check severity being used can be overridden by a WTOTYPE specified in the check definition. See WTOTYPE in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Controlling check exception message WTOs and their automation consequences

By default, IBM Health Checker for z/OS issues DOM requests to delete any check exception message WTOs left behind from previous check iterations. It does this DOMing right before the start of the new check iteration. That means that each time the check generates an exception, it also sends a new exception WTO, which also kicks off any automation actions you've set up for your installation.

So, what if you want more control over check exception WTOs and their automation consequences? For example, let's say you have a check that runs every hour. Now let's say that your check begins generating identical exceptions that you've automated on to prompt a beeper call to your favorite system programmer. You have not yet resolved the exception issue, and the installation policy is to not disable checks generating exceptions. That's just good practice, right? And yet your check might generate a lot of WTOs and beeper calls to that poor system programmer while the issue gets resolved.

That's where DOM control comes in! Starting with z/OS V1R13, IBM Health Checker for z/OS you can use the following functions that help you control whether you want to suppress WTOs and any automation actions they trigger for a check that is generating exceptions:

1. Add your check to the product using the DOM(CHECK) parameter on the HZSPRMxx and MODIFY *hzsproc* command. See ADD or ADDREPLACE CHECK parameters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.
2. From your check you decide when to DOM WTOs from previous check runs using the HZSFMSG REQUEST=DOM macro ("HZSFMSG macro — Issue a formatted check message" on page 307) or the REXX HZSLFMSG_REQUEST='DOM' function "HZSLFMSG function" on page 240.

Realizing the benefits of this function is all in the timing:

- If your check (added with DOM(CHECK)) is generating multiple identical unresolved exceptions, your check can wait to DOM the exception WTO (with HZSFMSG REQUEST=DOM or the REXX HZSLFMSG_REQUEST='DOM' function) until the exception condition is resolved. This way, your check is still running, but the exception WTOs from previous iterations of the check do not get DOMed. That means that exception messages from this check are just recorded in the message buffer and not sent as WTOs that set off automation actions.
- If your check is running successfully and is not generating an exception in a check iteration or is generating different check exceptions between iterations, your check should issue HZSFMSG REQUEST=DOM or the REXX HZSLFMSG_REQUEST='DOM' function to DOM WTOs from previous iterations. That way any subsequent exception will be sent as a WTO and will kick off any defined automation actions.

On the other hand, if you always want to delete WTOs for your check, and never wish to suppress duplicate exception WTOs, you will want to specify or default to DOM(SYSTEM) when you add your check, and let the system take care of DOMing of check WTOs for you!

Recommendations and recovery considerations for remote checks

Recovery needed for your check routine is basically the same as for any other program - the following recommendations are not, for the most part, unique to writing a check routine.

Make your check clean up after itself, because the system won't do it for you: IBM Health Checker for z/OS does not perform any end-of-task cleanup for your check. Check routines should track resources, such as storage obtained, ENQs, locks, and latches, in the PQE_ChkWork field.

Have your check stop itself when the environment is inappropriate: If your check routine encounters an environmental condition that will prevent the check from returning useful results, your check routine should stop itself and not run again until environmental conditions change and your code requests it to run. Your check should do the following to respond to an inappropriate environment:

1. Issue an information message to describe why the check is not running. For example, you might issue the following message to let check users know that the environment is not appropriate for the check, and when the check will run again:

   ```
   The server is down.
   When the server is available, the check will run again.
   ```

2. Issue the HZSFMSG service to stop itself:

   ```
   HZSFMSG REQEST=STOP,REASON=ENVNA
   ```

3. Make sure that your product or check includes code that can detect a change in the environment and start running the check again when appropriate. To start running the check, issue the following HZSCHECK service:

   ```
   HZSCHECK REQUEST=RUN,CHECKOWNER=checkowner,CHECKNAME=checkname
   ```

   If the environment is still not appropriate when your code runs the check, it can always stop itself again.

Your check should not add itself in an inappropriate environment: If you use a HZSADDCHECK exit routine r to add your checks to the system, note that some checks or product code might add or delete checks to the system in response to

changes in system environmental conditions. For example, if a check or product detects that a system environment is inappropriate for the check, it might then add only the checks useful in the current environment by invoking the HZSADDCHCK registration exit with an ADDNEW request (from the HZSCHECK service, the F *hzsproc* command, or in the HZSPRMxx parmlib member. You should add similar code to your HZSADDCHECK exit routine r to make sure that your checks don't run if they will not return useful results in the current environment. This code might:

- Delete checks that do not apply in the current environment
- Run a check so that it can check the environment and disable itself if it is inappropriate in the current environment. Consider supporting a check PARM so the installation may indicate the condition is successful and not an error.

If your check can never be valid for the current IPL, consider not even adding it from your HZSADDCHECK exit routine when you detect that situation. For example, if a check is relevant only when in XCF LOCAL mode but the system is not in that mode (and cannot change to that mode), there is no reason even to add the check.

Have your check stop itself for bad parameters: If your check routine is passed a bad parameter, it should stop itself using the HZSFMSG service:

```
HZSFMSG REQUEST=STOP,REASON=BADPARM
```

This request will also issue predefined HZS1001E error message to indicate what the problem is. The check routine will not be called again until it is refreshed or its parameters are changed. REQUEST=STOP prevents the check from running again and sets the results in the PQE_Result field of HZSPQE. The system sets the result field based on the severity value for the check. See “Issuing messages in your local check routine with the HZSFMSG macro” on page 117 for examples and complete information.

Take advantage of verbose and debug modes in your check:

IBM Health Checker for z/OS has support for the following modes:

- Debug mode, which tells the system to output extra messages designed to help you debug your check. IBM Health Checker for z/OS outputs some extra messages in debug mode, and some checks do also. When a check runs in debug mode, each message line is prefaced by a message ID, which can be helpful in pinpointing the problem. For example, report messages are not prefaced by message IDs unless a check is running in debug mode.

  There are two ways to issue extra messages in debug mode:

  – Use conditional logic such that when in debug mode (when field PQE_DEBUG in mapping macro HZSPQE has the value PQE_DEBUG_ON), your check issues additional messages.
  – Code debug type messages - see “Planning your debug messages” on page 203

  Users can turn on debug mode using the DEBUG=ON parameter in the MODIFY *hzsproc* command, in HZSPRMxx, or by overtyping the DEBUG field in SDSF to ON.
- Verbose mode, which tells the system to output messages with additional detail about non-exception information found by the check. (RACF checks, for example, issue additional detail in verbose mode.) To issue extra messages in verbose mode, use conditional logic such that when in verbose mode (when

field PQE_VERBOSE in mapping macro HZSPQE has the value PQE_VERBOSE_YES), your check issues additional messages.

Users can turn on verbose mode using the VERBOSE=YES parameter in the F *hzsproc* command or in HZSPRMxx.

Plan recovery for your check: Your check routine should be designed to handle abends. If the task that issues the HZSADDCK macro defining check defaults terminates for any reason, including an abend that is not re-tried, the system treats the check as if it is deleted.

In some cases you may not want your check to be stopped when an abend occurs because some abend causing conditions might simply clear with time. For example, if your check abends as a result of getting garbled data from an unserialized resource, such as a data area in the midst of an MVC, your check should provide its own recovery to:

- Retry the check a pre-determined number of times.
- If the check fails again, the check should stop running, but not stop itself.

This allows the check to try running again at the next specified interval, with every chance of success this time.

Look for logrec error records when you test your check: When testing your check, be sure to look for logrec error records. The system issues abend X'290' if the system encounters an error while a message is being issued, and issues a logrec error record and a description of the problem in the variable recording area (VRA).

Save time, save trouble - test your check with these commands: When you have written your check, test it with the following commands to find some of the most common problems people make in writing checks:

```
F hzsproc,UPDATE,CHECK(check_owner,check_name),DEBUG=ON
F hzsproc,UPDATE,CHECK(check_owner,check_name),PARM=parameter,REASON=reason,DATE=date
F hzsproc,DELETE,CHECK(check_owner,check_name),FORCE=YES
F hzsproc,DISPLAY,CHECK(check_owner,check_name),DETAIL
```

Avoid modifying system control blocks in your check routine: The IBM Health Checker for z/OS philosophy is to keep check routines very simple. IBM recommends that checks read but not update system data and try to avoid disruptive behavior such as modifying system control blocks.

See also “Debugging checks” on page 166.

Building Metal C checks

To make it easier to compile and build, link-edit, and bind a Metal C check, IBM Health Checker for z/OS provides a sample makefile, hzssmake.mk, for use with the z/OS UNIX System Services make utility. This makefile compiles and builds the sample files shipped in z/OS UNIX file system directory /usr/lpp/bcp/samples, where the makefile itself is shipped also.

Before you use the makefile, make sure you update the HCHECK_LOADLIB variable in the makefile. This variable names the dataset where the makefile will store the final load modules. This should be an APF authorized dataset in the link list, suitable for your installation.

To create all sample load modules, change to the directory where the hzssmake.mk file is stored and invoke the make utility like this:

```
make -f hzssmake.mk
```

Check out the other make rules in the makefile, in particular the cleanup rules. You can invoke cleanup, for example, using the following command:

```
make -f hzssmake.mk clean
```

This command will clean up all intermediate files, but will keep the generated load modules.

Once built, your Metal C load modules are ready to be registered with IBM Health Checker for z/OS as you would any other check. See:

- "Defining a local check to IBM Health Checker for z/OS" on page 108
- "Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS" on page 140
- "Creating product code that automatically registers checks at initialization" on page 197

For a Metal C sample HZSADDCHECK exit routine r, look for hzscadd.c in /usr/lpp/bcp/samples .

For more information about the make utility and the other utilities used in the makefile, see Shell command descriptions in *z/OS UNIX System Services Command Reference*.

```
#############################################################
# Name: HZSSMAKE                                            #
#                                                           #
# Description:  Makefile for building Metal C sample        #
#               local and remote health checks and          #
#               a sample HZSADDCHECK exit routine.          #
#                                                           #
#  COMPONENT: IBM Health Checker for z/OS (SCHZS)           #
#                                                           #
#  PROPRIETARY STATEMENT:                                   #
#                                                           #
#    Licensed Materials - Property of IBM                   #
#    5650-ZOS                                               #
#    Copyright IBM Corp. 2009                               #
#                                                           #
#    US Government Users Restricted Rights - Use, duplication  #
#    or disclosure restricted by GSA ADP Schedule Contract with#
#    IBM Corp.                                              #
#                                                           #
#  END OF PROPRIETARY STATEMENT                             #
#                                                           #
#  STATUS = HBB7770                                         #
#                                                           #
# Change Activity:                                          #
#                                                           #
# $L0=METALC    HBB7770 20081202 PDGIO: Initial version     #
# $L1=METALC    HBB7770 20090513 RDUT:  Updated options,targets#
#                                                           #
#############################################################

# The load modules created via this makefile will be put into this PDSE
# dataset. Change this to an APF authorized dataset in the link list,
# suitable for your installation.
# The linker/binder will create the PDSE, if it does not exist yet.
HCHECK_LOADLIB =HLQ.LOADLIB

# Location of Health Checker header filesHC_INCLUDES = "//'SYS1.SIEAHDR.H'"

# (Metal-) C compiler utility
```

```
CC = c99

# (Metal-) C compiler flags
# nosearch  - avoids using the non-Metal C header files
# I         - specifies our include paths, since nosearch disabled most
# metal + S - makes it Metal C instead of "regular" C/C++
# longname  - optional, but allows for longer than 8 character names
CFLAGS = -S -Wc,metal,longname,nosearch \
         -I /usr/include/metal,$(HC_INCLUDES)

# Assembler utility
AS = as

# Assembler flags
# rent - requests reentrant code; required for health checks
# goff - optional, but allows for longer than 8 character names
ASFLAGS = -mrent -mgoff

# Linker/binder utility
LD = ld

# Linker/binder flags
# ac=1 - assigns authorization code; required for health checks
# rent - requests reentrant code; required for health checks
# -S   - resolves system services (PAUSE token handling by remote
#        health checks) via SYSLIB CCSLIB
LDFLAGS  = -bac=1 -brent
LDFLAGSR = -S "//'SYS1.CSSLIB'"

# The four sample health checks and the one sample exit routine
HCHECK_TGTS = hzscchkp hzscchkr hzscrchc hzscrchk hzscadd

# Default rule
all: $(HCHECK_TGTS)

# *Uncomment* this rule, if you would like to keep the intermediate
# output files, in particular the generated .s assembler source,
# instead of letting 'make' delete them automatically.
#.SECONDARY:

# Rule for cleaning up intermediate output
clean:
 rm -f *.o *.s

# Rule for cleaning up all output
cleanall:
 rm -f *.o *.s
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscchkp)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscchkr)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscrchc)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscrchk)'"
 - tso -t "DELETE '${HCHECK_LOADLIB}(hzscadd)'"

# Rule for compiling a Metal C file into assembly language
%.s: %.c
 $(CC) $(CFLAGS) $<

# Rule for creating object code from assembly language
%.o: %.s
 $(AS) $(ASFLAGS) -o $@ $<

# Rules for creating LOAD modules (executable) from the object code
hzscchkp: hzscchkp.o
 $(LD) $(LDFLAGS) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscchkr: hzscchkr.o
 $(LD) $(LDFLAGS) -o "//'${HCHECK_LOADLIB}($@)'" $<
```

```
hzscadd: hzscadd.o
 $(LD) $(LDFLAGS) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscrchc: hzscrchc.o
 $(LD) $(LDFLAGS) $(LDFLAGSR) -o "//'${HCHECK_LOADLIB}($@)'" $<

hzscrchk: hzscrchk.o
 $(LD) $(LDFLAGS) $(LDFLAGSR) -o "//'${HCHECK_LOADLIB}($@)'" $<
```

Debugging checks

Naturally, we hope you'll never need this section and that all your checks will run perfectly the very first time. However, if you do run into trouble, this section will help you debug your check routine and HZSADDCHECK exit routine.

Was my check added to the system? Use the `F hzsproc,DISPLAY CHECK(checkowner,checkname)` to display the check you're adding to the system. If your check shows up, it was successfully added to the system. If it does not show up, it was not added to the system.

You can also check the return code from the HZSADDCK invocation in your HZSADDCHECK exit routine (for local checks) or check routine (for remote checks). A return code greater than 4 often indicates that there was a problem in adding the check to the system. See "HZSADDCK macro — HZS add a check" on page 260.

Turn on debug mode: Running in debug mode can help you debug your check, because in debug mode:

- Each message line is prefaced by a message ID, which can be helpful in pinpointing the problem. For example, report messages are not prefaced by message IDs unless a check is running in debug mode.
- Debug messages, which may contain information about the error, are issued only when the check is in debug mode.

You can turn on debug mode for a check that is not running properly using the DEBUG parameter in the MODIFY *hzsproc* command, in HZSPRMxx, or by overtyping the DEBUG field in SDSF to ON.

Create a recovery routine for your check routine if you need additional diagnostic data for your check routine. See "Establishing a recovery routine for a check" on page 139.

Debug HZSFMSG abends: If the system finds an error in a HZSFMSG macro call to issue a message, the system issues system abend X'290' with a unique reason code and creates a logrec error record. See the information for abend X'290' in *z/OS MVS System Codes* for a description of the abend reason codes.

If the abend is caused by an incorrect macro call, the system issues the following accompanying information:

- Logrec error record. Use EREP to view logrec errors, see "Using EREP to Obtain Records from the Logrec Log Stream " in *z/OS MVS Diagnosis: Tools and Service Aids*.
- A symptom dump written to the console and to the system log
- A SYSMDUMP, if you add a SYSMDUMP DD statement to *hzsproc*, the IBM Health Checker for z/OS procedure.

Note that the contents and data set disposition of your SYSMDUMP depends on the DISP= option you use on the DD statement. See "Obtaining ABEND dumps" in *z/OS MVS Diagnosis: Tools and Service Aids*.

- There may be additional diagnostic data in the register at time of the abend that can help with debugging. See "ABEND Codes" on page 332 for the kinds of diagnostic data that may be available.

  If your check routine has a recovery routine, the SDWA for the recovery routine will contain these registers in the SDWAGRSV field.

If the abend is caused by the system, the system issues an SVC dump.

Chapter 8. Writing REXX checks

A REXX check consists of an exec containing one or more remote checks coded in REXX language instructions. This code is interpreted and executed by System REXX and runs in a System REXX address space, in an APF authorized environment defined by System REXX. You can identify your check as a REXX check by using the REXX(YES) parameter in the check definition.

Use the following documents for guidance on coding in the REXX language:

- *z/OS TSO/E REXX User's Guide*
- *z/OS TSO/E REXX Reference*
- System REXX in *z/OS MVS Programming: Authorized Assembler Services Guide*

Look for REXX check information in our Redpaper: There's lots of great experience-basecd information on writing REXX checks in Redpaper *Exploiting the Health Checker for z/OS infrastructure* (REDP-4590-00).

In this chapter, we'll cover the following:

- "Sample REXX checks"
- "REXX check basics" on page 170
- "Using input data sets in a TSO-environment REXX check" on page 174
- "Using REXXIN data sets" on page 174
- "Using REXXOUT data sets" on page 175
- "Defining a REXX check to IBM Health Checker for z/OS" on page 177
- "Issuing messages from your REXX check with the HZSLFMSG function" on page 180
- "Writing a check with dynamic severity levels" on page 124
- "Controlling check exception message WTOs and their automation consequences" on page 126
- "The well-behaved REXX check - recommendations and recovery considerations" on page 187
- "Debugging REXX checks" on page 188

Sample REXX checks

Of course you're going to read this entire chapter to understand everything you need to know about writing a REXX check. But we also have what you're really looking for - REXX check samples in SYS1.SAMPLIB:

- **HZSSXCHK** - Sample REXX checks.
- **HZSSXCHN** - Sample REXX check showing the use of HZSLFMSG_REQUEST='DIRECTMSG'.
- **HZSSMSGT** - Sample message input, which is common to all check types.

You'll find these and more check samples on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

REXX check basics

You can use System REXX services to write a REXX check to gather installation information and look for problems, most likely by reading data set(s) and using the AXRCMD function to issue a system command and looking at its output, and then issuing the check results in messages. IBM Health Checker for z/OS may also write check exception messages as WTOs.

A REXX check runs in a System REXX address space, in an APF authorized environment defined by System REXX. .

You can write your REXX checks for two environments: TSO and non-TSO. Writing a check for a TSO environment gives you a dynamic TSO environment to work with and is very secure, because it ensures that the check routine runs by itself in a single address space. Example HZSSXCHK in SYS1.SAMPLIB shows code for both a TSO and a non-TSO environment check.

See System REXX in *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about writing and running REXX execs on z/OS.

We recommend that you keep the REXX check very simple. At a high level, **your REXX check will:**

1. Invoke the HZSLSTRT function to indicate that the exec has started running and place some check information from the HZSPQE data area into REXX variables.
2. Look at the HZS_PQE_ENTRY_CODE REXX variable set by IBM Health Checker for z/OS from the check definition to identify the REXX check being called when an exec contains more than one REXX check.
3. Start processing the REXX check.
4. If desired, look for the function code set by IBM Health Checker for z/OS (in HZS_PQE_FUNCTION_CODE). If the function code is INITRUN for a first iteration of a REXX check, the REXX check sets the HZS_PQE_CHKWORK field to nulls and the REXX check should do any necessary set up.
5. The REXX check should validate input parameters, if any, for the REXX check when the system indicates that parameter data has changed. Use the HZSLSTRT REXX function output variable, HZS_PQE_LOOKATPARMS, to see whether check parameters have changed since the last time the REXX check ran. (Check parameters are contained in HZSLSTRT output variable HZS_PQE_PARMAREA.) When the HZS_PQE_LOOKATPARMS variable is set on, it indicates that check parameters have been changed since the last time the REXX check ran. Use the HZSLFMSG REXX function input variables to report parameter errors found by the REXX check. See “HZSLFMSG function” on page 240.
6. Now for the guts of the REXX check - check for potential problems on a system.
7. Issue messages or handle parameter and other errors the REXX check encounters using the HZSLFMSG function. HZSLFMSG is the interface to the HZSFMSG macro - see “HZSFMSG macro — Issue a formatted check message” on page 307. HZSLFMSG also sets or modifies the status for the REXX check.
8. Invoke the HZSLSTOP function to indicate the REXX check has completed running.

REXX checks only run when System REXX is up and running: If System REXX is not available, your REXX checks will not run because these checks run in a System

REXX address space. To add your REXX check, see "Defining a REXX check to IBM Health Checker for z/OS" on page 177.

Defining the environment for a REXX check: A REXX check runs in a System REXX address space in an environment defined and controlled by System REXX. IBM Health Checker for z/OS runs your REXX check using the AXREXX service. REXX checks run under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*. See System REXX in *z/OS MVS Programming: Authorized Assembler Services Guide* for information.

The system loads the message table for your REXX check into the IBM Health Checker for z/OS address space.

Information that every REXX check starts out with: When IBM Health Checker for z/OS calls the REXX check, it sets the following HZSLSTART function variables for the REXX check to use:

- HZS_HANDLE, which identifies the remote REXX check in order to synchronize processing between the REXX check and IBM Health Checker for z/OS. This is important because a REXX check is a remote check - it runs in a System REXX address space. The system uses this handle as input within the HZSLSTRT, HZSLFMSG, and HZSLSTOP functions. The REXX check should never alter this field and probably will never even need to reference it except when encapsulating calls to those functions in procedures. In this case, make sure you use the REXX EXPOSE statement to allow the use of HZS_HANDLE inside the procedure. This will avoid ABEND 290 RSN=858 (HZS_HANDLE was not valid)
- HZS_PQE_ENTRY_CODE, which identifies the check being called, for a REXX check containing more than one check.
- HZS_PQE_FUNCTION_CODE, which indicates whether the REXX check is being called for the first time (INITRUN) or for a subsequent iteration (RUN).

Limit a REXX check to looking at one setting or one potential problem. Limiting the scope of a REXX check will make it easier for the installation using the REXX check to:

- Resolve any exceptions that the REXX check finds by either fixing the exception, overriding the setting, or deactivating the REXX check.
- Set appropriate override values for REXX check defaults such as severity or interval.

Do not set a return code in your REXX check: IBM Health Checker for z/OS ignores any return code set by your REXX check. When you use the HZSLFMSG function, the system will return information in the RESULT and HZSLFMSG_RSN variables.

Use the 2K check work area: Use the 2K check work area (HZS_PQE_CHKWORK variable made available by the HZSLSTRT function) to hold data that you want to retain through check iterations for the life of the REXX check. Prior to the INITRUN function code call, the system sets the 2K work area to null. The HZS_PQE_CHKWORK variable is the only HZSLSTRT variable your REXX check should write to. The system saves the HZS_PQE_CHKWORK contents when the REXX check invokes the HZSLSTOP System REXX function, and then sets the area to null when any of the following occur

- The REXX check is to run for the first time
- The check is REFRESHed

- The check becomes either INACTIVE or DISABLED for any reason besides invalid parameters

If your REXX check does obtain additional resources, allocation of a data set, for example, the REXX check must release these resources before it completes. A REXX check is not called for cleanup or delete, as a local check is, so that when the REXX check runs again there is no guarantee it will execute in the address space or under the same task. The REXX check must also release resources when a non-exception condition, such as a time-out or cancel, occurs.

Using the IBM Health Checker for z/OS System REXX functions: Use the System REXX functions listed below in your REXX check. Note that a check is marked in error if ANY of the HZSL*xxxx* functions fail with a return code 8 or higher. See the individual HZSL*xxxx* function return codes in Chapter 11, "IBM Health Checker for z/OS System REXX Functions," on page 235 to determine the cause of an error.

- Invoke **HZSLSTRT** to indicate that the REXX check has started to run. This function sets REXX variables containing the HZSPQE information for the REXX check, such as check definition values. This function is used at the very start of the REXX check. Do not alter any HZSLSTRT variables except for the HZS_PQE_CHKWORK work area. Some of the most important HZSLSTRT variables you use in a REXX check include:

Table 19. Important HZSPQE information used in a REXX check from HZSLSTRT variables

Field name	Meaning
HZS_PQE_FUNCTION_CODE	Contains the function code for the REXX check. The REXX check receives control in response to either the RUN or INITRUN function code. The system sets this field on entry to the REXX check.
HZS_PQE_ENTRY_CODE	Contains the identifier (entry code) assigned for the REXX check in the check definition. The entry code is used when a REXX exec contains multiple checks. The system sets this field on entry to the REXX check.
HZS_HANDLE	Identifies the remote REXX check in order to synchronize processing between the REXX check and IBM Health Checker for z/OS. This is important because a REXX check is a remote check - it runs in a System REXX address space. The REXX check uses this handle as input to the HZSLSTRT, HZSLFMSG, and HZSLSTOP functions. The system sets this field on entry to the REXX check.
HZS_PQE_LOOKATPARMS	A bit indicating that the parameters have changed. If this bit is on, the REXX check should read the HZS_PQE_PAREMAREA and HZS_PQE_PARMLEN variables.
HZS_PQE_VERBOSE	A byte indicating whether the REXX check is in verbose mode.
HZS_PQE_DEBUG	A byte indicating whether the REXX check is in debug mode.
HZS_PQE_PARMAREA	The area containing the user parameters. Quotes surrounding the PARMS value in an operator command or HZSPRMxx statement are not included.
HZS_PQE_CHKWORK	2K check work area used and mapped by the REXX check as needed. The system zeros the 2K check work area before calling the REXX check with function code RUN. A REXX check can both write and read from this field, and the system will retain this information for subsequent calls to the check. Changes made to any other variables are not saved between function calls.
HZS_PQE_DOM_CHECK	Indicates how the DOM(SYSTEM|CHECK) parameter was set when the check was added to IBM Health Checker for z/OS. If the value 1, DOM(CHECK) was specified for the check. If the value is 0, DOM(SYSTEM) was specified for the check. For information on how DOM=CHECK and HZSFMSG REQUEST=DOM/ HZSLFMSG_REQUEST=DOM works, see "Controlling check exception message WTOs and their automation consequences" on page 126.

See "HZSLSTRT function" on page 236 for all of the REXX variables returned.

- Invoke **HZSLFMSG** to:
  - Issue REXX check messages and IBM Health Checker for z/OS messages. You will invoke this function multiple times in your REXX check. See "Issuing messages from your REXX check with the HZSLFMSG function" on page 180
  - Report a problem with the check - use HZSLFMSG to report the problem and change the check state. You can also stop the REXX check in case of an error found, such as bad parameters or an inappropriate environment for the check.
- Invoke **HZSLSTOP** to indicate that the REXX check has completed an iteration. The REXX check invokes this function at the end of the REXX check. This function saves HZS_PQE_CHKWORK for the next REXX check iteration.

All of the REXX functions return a return code (RESULT variable) and reason code (HZSL*nnnn*_RSN variable). These functions also include many other useful input and output variables. See Chapter 11, "IBM Health Checker for z/OS System REXX Functions," on page 235 for complete information on these functions.

Give grouped REXX checks individual entry codes: Multiple REXX checks can use a single REXX exec. When you do this, each individual REXX check still gets its own HZSPQE area, and you must define a unique entry code for each individual check. This ensures that the REXXIN and REXXOUT data sets for each REXX check are unique - the system uses the entry code in the data set name suffix. Code your REXX check to look for the entry code passed in the HZSLSTART function HZS_PQE_ENTRY_CODE variable, and pass control to processing for the REXX check indicated. You define the entry code for each REXX check with the ENTRYCODE parameter in the check definition on the HZSADDCK call or HZSPRMxx parmlib member.Note that the IBM Health Checker for z/OS will not verify the uniqueness of the entry codes you define for your REXX checks.

The following example shows how a REXX check uses entry codes to route control to individual checks:

```
/*********************************************************************/
/* Check the entry code to determine which check to process          */
/*********************************************************************/
 IF HZS_PQE_ENTRY_CODE = 1 THEN
   DO
     Call Process_HZS_SAMPLE_REXXIN_CHECK
   END
 IF HZS_PQE_ENTRY_CODE = 2 THEN
   DO
     Call Process_HZS_SAMPLE_REXXTSO_CHECK
   END
 EXIT
```

If you are using HZSADDCHECK exit routines to add your REXX checks to the system, you should also use a single exit routine to add related checks to the system. See "Defining a REXX check to IBM Health Checker for z/OS" on page 177.

Do not attempt to communicate between individual REXX checks. Even though you may have placed all of your REXX checks in the same exec, do not rely on communication between them. Each REXX check is intended to stand by itself and has a unique severity, reason, parameters, HZSPQE data area, and entry code.

Using input data sets in a TSO-environment REXX check

A REXX check running in a TSO environment (REXXTSO(YES)) can use TSO services. A REXXTSO(YES) check can allocate and read from or write to any data set that it can access When there is a lot of input parameter data, we recommend that the check parameter be the name of the data set and the exec would allocate and read from that data set to access its parameters. For example, lets say a TSO REXX check is defined with the PARMS parameter, as follows:

```
PARMS('DSN(IBMUSER.HZSSXCHK.DATA)')
```

Based on the data set specified in PARMS, the REXX check uses data set **IBMUSER.HZSSXCHK.DATA** as its input data set.

In order to get consistent results from your REXX checks, IBM suggests that the exec has exclusive access to the input data set. If the system cannot allocate or use a requested input data set, the REXX check does not run successfully.

Using REXXIN data sets

An exec running in a **non-TSO environment** can use the REXXIN data set to read data from. You must specify REXXTSO(NO) and REXXIN(YES) in the check definition in order to use a REXXIN data set. Typically, a check would use a REXXIN data set when it has a lot of input parameter data.

TSO environment REXX checks can use input data sets, see “Using input data sets in a TSO-environment REXX check.”

In order to get consistent results from your REXX checks, IBM suggests that the exec has exclusive access to the REXXIN data set. If the system cannot allocate or use a requested REXXIN data set, the REXX check does not run successfully.

REXXIN data set naming conventions

If you specify REXXIN(YES), the system allocates and names your REXXIN input data set using the following REXX check definition information:

1. REXXHLQ(*hlq*)
2. EXEC(*execname*)
3. REXXIN(YES)
4. ENTRYCODE(*entrycode*)

For example, let's say a non-TSO REXX check is defined with the following parameters:

```
EXEC(HZSSXCHK)
REXXHLQ(IBMUSER)
REXXIN(YES)
ENTRYCODE(1)
```

The REXXIN data set name that the system uses is **IBMUSER.HZSSXCHK.REXXIN.E1**. If you did not define an entry code for this REXX check, the REXXIN data set name would be **IBMUSER.HZSSXCHK.REXXIN**

Using REXXOUT data sets

Both TSO environment (REXXTSO(YES)) and non-TSO (REXXTSO(NO)) environment REXX checks can use REXXOUT data sets to diagnose REXX check problems. The REXXOUT data set is provided when the check is in debug mode and is intended to capture data used to debug the check. When a REXXOUT data set is provided, System REXX writes data to the REXXOUT data set every time:

- You code the SAY or TRACE keyword in your REXX exec. For example, if your REXX check finds an error in parameters or when issuing a message (HZSLFMSG function), you might want to capture data such as HZSLFMSG return and reason codes, system diagnostic information, abend reason codes and details of user errors.
- When your REXX check receives a TSO error message
- When your REXX check receives a System REXX message

If your REXX check is not running in debug mode, this output is lost. To place a REXX check in DEBUG mode, use the following command example:

```
F hzsproc,UPDATE,CHECK=(checkowner,checkname),DEBUG=ON
```

From SDSF, you can also place a REXX check in debug mode by over-typing the DEBUG field to ON.

REXX check exception, information, and report messages are written to the message buffer rather than the REXXOUT data set.

The system will allocate the REXXOUT data set for you based on the naming conventions for your environment, if it is not already allocated when the REXX check runs. However, you must ensure that IBM Health Checker for z/OS address space has the authority to allocate the data set. If the system cannot exclusively allocate or use the REXXOUT data set, the REXX check will not run successfully.

REXXOUT data set naming conventions

For both TSO (REXXTSO(YES)) and non-TSO (REXXTSO(NO)) environment REXX checks, the system allocates REXXOUT data sets for use using the following:

1. REXXHLQ(*hlq*) from the check definition
2. EXEC(*execname*) from the check definition
3. REXXOUT
4. ENTRYCODE(*entrycode*) from the REXX check definition, if defined

For example, let's say a REXX check is defined with the following parameters:

```
EXEC(HZSSXCHK)
REXXHLQ(IBMUSER)
ENTRYCODE(1)
```

The REXXOUT data set name that the system uses is **IBMUSER.HZSSXCHK.REXXOUT.E1**. If you did not define an entry code for this REXX check, the REXXOUT data set name would be **IBMUSER.HZSSXCHK.REXXOUT**.

Examples: Capturing error data in REXXOUT

The following examples show code that captures error data in REXXOUT. Note that before writing the error detail to REXXOUT, the REXX checks first determine whether the check is in debug mode by looking at the HZS_PQE_DEBUG variable.

Example 1 - Using HZSLFMSG to capture bad parameter data in REXXOUT: The following example shows a TSO REXX check which requires a REXXIN data set, the name of which is specified PARMS parameter. If the REXX check finds that the parameter in PARMS is invalid, it uses the SAY keyword to capture error information in a REXXOUT data set allocated by the system when the check is in debug mode:

```
Process_HZS_SAMPLE_REXXTSO_CHECK:
/*********************************************************************/
/* Process parameters for HZS_SAMPLE_REXXTSO_CHECK                   */
/*********************************************************************/
/*                                                                   */
/* For our example,                                                  */
/* - assume that the required PARMAREA string is DSN(value) where    */
/*   value is the name of a sequential data set that contains data   */
/*   to be processed by this check. We use TSO services to do the    */
/*   validation.                                                     */
/*                                                                   */
/*********************************************************************/
 ADDRESS TSO "Alloc "HZS_PQE_PARMAREA" SEQ OLD"
 IF RC ^= 0 THEN
    DO
      HZSLFMSG_REQUEST = "STOP"
      HZSLFMSG_REASON = "BADPARM"
      HZSLFMSG_RC = HZSLFMSG()
      IF HZS_PQE_DEBUG = 1 THEN
       DO                /* Report debug detail in REXXOUT      */
         SAY "PARMS: ||"HZS_PQE_PARMAREA"||"
         SAY "HZSLFMSG RC"  HZSLFMSG_RC
         SAY "HZSLFMSG RSN" HZSLFMSG_RSN
         SAY "SYSTEMDIAG"   HZSLFMSG_SYSTEMDIAG
       END
       EXIT              /* The check is not performed          */
    END
```

In this example, we write the return and reason codes from HZSLFMSG and system diagnostic information to REXXOUT to help debug the parameter problem.See “HZSLFMSG function” on page 240 for complete information about HZSLFMSG input and output variables.

Example 2 - Capturing HZSLFMSG message function error data in REXXOUT: The following example from a non-TSO REXX check shows how to capture error data when the message function, HZSLFMSG, completes with a RESULT of 8:

```
/*********************************************************************/
/*                                                                   */
/* When the message service detects a user error, HZSLFMSG result    */
/* will be 8.                                                        */
/*                                                                   */
/* HZSLSFMSG_RSN = 000008xx    A user error was detected             */
/*                                                                   */
/* HZSLSFMSG_RSN = 0000089F    See HZSLFMSG_USERRSN.                 */
/* HZSLSFMSG_USERRSN           The reason for the user  error.       */
/*                             See ABEND REASON CODES in HZSLFMSG    */
/*                                                                   */
/* HZSLFMSG_ABENDRESULT contains diagnostic detail about user        */
/* errors                                                            */
/*                                                                   */
/* Check looks for debug mode on, and if on, writes SAY messages     */
/* with debug detail in REXXOUT data set.                            */
/*                                                                   */
/*********************************************************************/
         IF HZS_PQE_DEBUG = 1 THEN
          DO                        /* place debug detail in REXXOUT */
            SAY "PARMS: ||"HZS_PQE_PARMAREA"||"
```

```
  SAY "HZSLFMSG RC"  HZSLFMSG_RC
  SAY "HZSLFMSG RSN" HZSLFMSG_RSN
  SAY "SYSTEMDIAG"   HZSLFMSG_SYSTEMDIAG
  SAY "USER RSN" HZSLFMSG_UserRsn
  SAY "USER RESULT" HZSLFMSG_AbendResult
END
```

In this example, we write the return and reason codes from HZSLFMSG, system diagnostic information, the user error detail, and abend reason code to REXXOUT to help debug the HZSLFMSG error. See "HZSLFMSG function" on page 240 for complete information about HZSLFMSG input and output variables.

See the information for abend X'290' in *z/OS MVS System Codes* for a description of the abend reason codes for IBM Health Checker for z/OS.

Example 3: Capturing TRACE data in REXXOUT: The following REXXOUT output data was created by placing the TRACE ALL REXX instruction in SYS1.SAMPLIB check HZSSXCHK, and running the checks with DEBUG(ON):

```
   905 *-*      ADDRESS TSO "Alloc DSN("DataSetName") OLD"
       >>>        "Alloc DSN('IBMUSER.HZSSXCHK.DATA') OLD"
IKJ56228I DATA SET IBMUSER.HZSSXCHK.DATA NOT IN CATALOG OR CATALOG CAN NOT BE AC
IKJ56701I MISSING DATA SET NAME+
IKJ56701I MISSING NAME OF DATA SET TO BE ALLOCATED
       +++ RC(12) +++
   963 *-*    ERROR:
   964 *-*    FAILURE:
   965 *-*    NOVALUE:
   966 *-*    HALT:
   967 *-*    ERR1 = "An Error has occurred on line: "Sigl
   968 *-*    ERR2 =  sourceline(sigl)
   969 *-*    Say Err1
An Error has occurred on line: 905
   970 *-*    Say "Line "Sigl" text: "Err2
Line 905 text:     ADDRESS TSO "Alloc DSN("DataSetName") OLD"
   971 *-*    ADDRESS TSO "FREE DSN("DataSetName")"
       >>>      "FREE DSN('IBMUSER.HZSSXCHK.DATA')"
IKJ56247I DATA SET IBMUSER.HZSSXCHK.DATA NOT FREED, IS NOT ALLOCATED
       +++ RC(12) +++
   972 *-*    HZSLFMSG_REQUEST = "STOP"         /* Disable the check
   973 *-*    HZSLFMSG_REASON = "ERROR"
   974 *-*    HZSLFMSG_DIAG   =  Right(RC,16,0) /* report the decimal rc in the
e and the check                                      display detail
   977 *-*    HZSLFMSG_RC = HZSLFMSG()
   978 *-*    IF HZS_PQE_DEBUG = 1
       *-*     THEN
   979 *-*     DO                                /* Report debug detail in REXXOU
   980 *-*      SAY "PARMS: "HZS_PQE_PARMAREA
PARMS: DSN(IBMUSER.HZSSXCHK.DATA)
   981 *-*      SAY "HZSLFMSG RC"  HZSLFMSG_RC
HZSLFMSG RC 0
   982 *-*      SAY "HZSLFMSG RSN" HZSLFMSG_RSN
HZSLFMSG RSN 0
   983 *-*      SAY "SYSTEMDIAG"   HZSLFMSG_SYSTEMDIAG
SYSTEMDIAG N/A
   984 *-*     END
   985 *-*    EXIT               /* The check is not performed          */
```

Defining a REXX check to IBM Health Checker for z/OS

After you've written your REXX check, use the ADD | ADDREPLACE CHECK parameter in an HZSPRMxx parameter to define check defaults and add the check. Do this as follows:

1. Create a parmlib member.

2. Use the ADD | ADDREPLACE CHECK parameter to define the new System REXX check definition. For example:

```
ADDREPLACE CHECK(IBMSAMPLE,HZS_SAMPLE_REXXTSO_CHECK)
        EXEC(HZSSXCHK)
        REXXHLQ(IBMUSER)
        REXXTSO(YES)
        MSGTBL(HZSSMSGT)
        ENTRYCODE(2)
        PARMS('DSN(MY.PARMLIB)')
        SEVERITY(LOW)
        INTERVAL(0:05)
        EINTERVAL(SYSTEM)
        DATE(20061219)
        REASON('A sample check to demonstrate an ',
              'exec check using TSO services.')
```

   See the ADD or ADDREPLACE CHECK parameter in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

3. Use the ADD,PARMLIB command to add the new parmlib member containing the REXX check definition. For example:

```
F hzsproc,ADD,PARMLIB=xx
```

While IBM suggests using HZSPRMxx to define REXX checks, you can also define your REXX check by writing an authorized HZSADDCHECK exit routine r running in the IBM Health Checker for z/OS address space. See Chapter 9, "Writing an HZSADDCHECK exit routine," on page 191.

Note that whether you use HZSPRMxx or an HZSADDCHECK exit routine to define your check, you cannot change the high level qualifier for the REXXIN and REXXOUT data sets once you have defined them.

You can specify the following parameters for REXX checks in either the ADDREPLACE CHECK parameter in HZSPRMxx or their equivalents in the HZSADDCK macro:

- EXEC(*execname*) - This parameter, required for a REXX check defined in the HZSPRMxx parmlib member, specifies the name of the REXX exec containing the REXX check or checks. This parameter tells the system that you are defining a REXX check. For an assembler check, you would specify the CHECKROUTINE(*checkname*).

  If you define your REXX check with the HZSADDCK macro in an HZSADDCHECK exit routine r, the equivalent of EXEC(*execname*) is the REXX=YES,EXEC=execname parameters.
- REXXHLQ(*hlq*) - This parameter, required for a REXX check, specifies the high level qualifier for any input or output data set for the check.
- REXXTIMELIMIT(*timelimit*) - This optional input parameter specifies the number of seconds a check iteration is allowed to run before the system ends it. A value of 0, which is the default, specifies that there is no time limit for the check.
- REXXTSO(YES | NO) - This parameter, optional for a REXX check, specifies whether the check runs in a TSO environment or a non-TSO environment. The default is REXXTSO(YES).
  - REXXIN(YES | NO) - This parameter, optional for a REXX check, specifies whether or not a non-TSO check requires an sequential input data set. The name of the REXXIN data set will consist of the high level qualifier specified in the HLQ parameter, the exec name specified in the EXEC parameter, and an optional entry code specified in the ENTRYCODE parameter.

    You can only specify REXXIN(YES) if you also specify REXXTSO(NO).

If you modify the definition for your REXX check, the changes will take effect the next time the check runs.

Gotcha - Don't make a typo when defining your REXX check! When you define your REXX check in a HZSPRMxx parmlib member using the ADD|ADDREPLACE CHECK parameters, do it carefully, because it is a nuisance to delete check definitions created using parmlib members, because you can't delete the check definition, even if you delete all the checks. And creating multiple definitions for the same REXX check may cause an error when the check is added or refreshed.

If you do make a mistake, you can do one of the following to resolve the problem:

- Issue the following command, which will first delete all existing check definitions and then add the definitions found in the specified parmlib members:
  `F hzsproc,REPLACE,PARMLIB=(suffix1,suffix2,...suffixn),CHECKS`
- If you make a mistake when defining a REXX check in an HZSADDCHECK exit routine r, you must delete the check (by creating a policy statement that deletes the check) and then delete the erroneous exit using SETPROG. You can then add the corrected HZSADDCHECK exit routine r again.
- Stop and start IBM Health Checker for z/OS to delete the check definition.

Why does IBM Health Checker for z/OS make it so hard to delete a check definition? Because if you delete your check definition, you lose all the history of the check and may find it more difficult to re-define it.

Installation requirement for running compiled REXX checks: In order to run compiled REXX checks, installations must have either the SEAGALT or SEAGLPA data set available in the system search order.

- SEAGALT is provided in z/OS V1R9 and higher
- SEAGLPA is provided in the RIBM Library for REXX on System z product

REXX execs that are not compiled do not require the SEAGALT or SEAGLPA libraries. For more information, see:

- *IBM Compiler and Library for REXX on System z: User's Guide and Reference* for information about SEAGALT and SEAGLPA and writing compiled REXX code
- *z/OS MVS Programming: Authorized Assembler Services Guide* for information on z/OS System REXX.

Using ENF event code 67 to listen for check status changes

If your check is authorized, it can use the ENFREQ LISTEN service to detect check status changes. On the ENFREQ service, specify theX'20000000' status change event qualifier and the listener user exit routine that is to receive control after the specified event occurs. The listener user exit specified receives control when IBM Health Checker for z/OS comes up and notifies the check routine of the status change.

To listen for ENF event code 67, you must specify the qualifying events on the BITQUAL parameter, which specifies a 32-byte field, a hexadecimal constant, or a register containing the address of a 32-byte field containing a bit-mapped qualifier that further defines the event. The qualifier is mapped by mapping macro HZSZENF. The BITQUAL value for the status change event is Enf067_BitQual_StatusChange in the HZSZENF mapping macro. This might mean on eof the following:

- The check completed with a different result than the last time it ran. For example, the check ran successfully after the previous check run issued an exception or vice versa.
- The check was deactivated or deleted

The check then might want to issue the HZSQUERY macro to get information about the check.

This event may not be presented if IBM Health Checker for z/OS is terminating (indicated by a X'40000000' ENF 067 event for NotAvailable - see "Using ENF event code 67 to listen for IBM Health Checker for z/OS availability" on page 139).

If the check routine decides it is no longer interested in knowing if IBM Health Checker for z/OS is up or not, it can issue the ENFREQ REQUEST=DELETE request to delete the listen request.

For information about ENFREQ and listener exits, see:

- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
- Listening for system events in *z/OS MVS Programming: Authorized Assembler Services Guide*

Issuing messages from your REXX check with the HZSLFMSG function

This section covers issuing messages from your REXX check. REXX check messages are important because they report the results of the check to an installation. See "Planning your check messages" on page 201.

Each REXX check should issue at least:

- One or more messages for any exception found to the setting the check is looking for.
- A message indicating that no exceptions were found, when appropriate.

You can issue messages for your REXX check in two ways:

- If you are creating a message table for your check, you issue the message using HZSLFMSG, but the actual text and explanation for your check messages are defined in your message table, see "Issuing messages in your local check routine with the HZSFMSG macro" on page 117 and "HZSLFMSG function" on page 240.
- If you are writing a DIRECTMSG check that issues messages directly from the check routine, use HZSLFMSG_REQUEST='DIRECTMSG'. See "Input variables for HZSLFMSG_REQUEST='DIRECTMSG'" on page 242.

See "Issuing messages for your check - message table checks versus DIRECTMSG checks" on page 102 for more information.

Issue WTO's from your REXX check using only HZSLFMSG for an exception message. HZSLFMSG produces data for the message buffer to create reports and issue and exception message.

You'll use the HZSLFMSG function to:

- Issue one of the following requests:
  - **HZSLFMSG_REQUEST='CHECKMSG'**- Indicates that you want to issue a check specific message, such as an exception or report message. You use the HZSLFMSG interface to issue a message and define variables, but the actual text and explanation for your check messages are assembled by the

HZSMSGEN REXX exec from the message table. See “Issuing messages in your local check routine with the HZSFMSG macro” on page 117.

You can indicate the message number you want to issue with a HZSLFMSG_MESSAGENUMBER=*msgnum* input variable.

- **HZSLFMSG_REQUEST='DIRECTMSG'** indicates that you are issuing a check specific message, such as an exception or report message directly from the check routine, one that does not have a message table associated with it. The message text for this message is provided in the HZSLFMST_REQUEST='DIRECTMSG' input variables. See “Input variables for HZSLFMSG_REQUEST='DIRECTMSG'” on page 242.
- **HZSLFMSG_REQUEST='HZSMSG'** - Indicates that you want to issue an IBM Health Checker for z/OS message. IBM Health Checker for z/OS provides the message text for an HZSMSG request. See “Input variables for HZSLFMSG_REQUEST='HZSMSG'” on page 245.
- **HZSLFMSG_REQUEST='STOP'** - Indicates that the system should stop calling this check . The message text is provided by IBM Health Checker for z/OS. See “Input variables for HZSLFMSG_REQUEST='STOP'” on page 250

- Define the number of variables and the variables themselves for a message with the HZSLFMSG_INSERT input variable.
- The HZSLFMSG_RC output variable reports the return code for the HZSLFMSG function.

Example - Issuing a message for a REXX check with a message table: The following example shows how a REXX check uses the HZSLFMSG function to issue an exception message for a check with a message table associated with it. Note that REXX variable values are processed as character text; the input values for decimal and hexadecimal variables must be expressed in hexadecimal.

```
/************************************* ***************************/
/* Build and write exception message                              */
/*                                                                */
/* In the message source, message number 1, has 5 variables       */
/*                                                                */
/*  symbol     output     input                                   */
/*  name       format     format                                  */
/*  ------     ------     ------                                  */
/*  num-avail   hex       a fullword hex value is expected        */
/*  num-inuse   decimal   a fullword hex value is expected        */
/*  num-avail   hex       a fullword hex value is expected        */
/*  num-inuse   decimal   a fullword hex value is expected        */
/*  summary     char      text (char)                             */
/*                                                                */
/******************************************************************/
 HZSLFMSG_REQUEST = "CHECKMSG"          /* A message table  request  */
 HZSLFMSG_MESSAGENUMBER = 1             /* write message 1           */
 HZSLFMSG_INSERT.0 = 5                  /* 5 input values are provided */
 HZSLFMSG_INSERT.1 = '0000000A'x        /* a fullword hex value      */
 HZSLFMSG_INSERT.2 = '0000000A'x        /* a fullword hex value      */
 HZSLFMSG_INSERT.3 = '00000020'x        /* a fullword hex value      */
 HZSLFMSG_INSERT.4 = '00000020'x        /* a fullword hex value      */
 HZSLFMSG_INSERT.5 = 'My summary text'  /* a character string        */
 HZSLFMSG_RC = HZSLFMSG()
 IF HZS_PQE_DEBUG = 1 THEN
    DO
       SAY "HZSLFMSG RC"  HZSLFMSG_RC
       SAY "HZSLFMSG RSN" HZSLFMSG_RSN
       SAY "SYSTEMDIAG"   HZSLFMSG_SYSTEMDIAG
       IF HZSLFMSG_RC = 8 THEN
          DO
```

```
          SAY "USER RSN"    HZSLFMSG_UserRsn
          SAY "USER RESULT" HZSLFMSG_AbendResult
        END
    END
```

In this example:

- HZSLFMSG_INSERT.*x* is a message insert text. The text provided provided in the insert should be compatible with the class attribute of the associated message variable in the message table. A class attribute of hex, decimal or timestamp in the message table will treat the insert data as a hexadecimal string.
- Variable HZSLFMSG_INSERT.1 expects to receive hexadecimal data. In the message table, variable 1 has a class attribute of hex:

  ```
  &lt;mv class="hex">variable 1
  ```

Note that decimal text also converts hexadecimal values to decimal text. For example, lets say that variable in the message table has a class attribute of:

```
 &lt;mv class=&odq;decimal">variable 1&lt;/mv>
```

In that case, the REXX check might use the following HZSLFMSG input variable:

```
HZSLFMSG_INSERT.1 = &csqg;0A'&csqg;X   /* The decimal value 10 is displayed */
```

The example implies everything needs to be a fullword. A hex value is required. Since this is a rexx and not assembler it makes more sense to make it look like a rexx variable. Hex values must be bytes. So refer to the number of bytes. '1234' when display as hex would appear as 'F1F2F3F4' '1234' '0A'x when displayed as decimal variable with a fieldsize of 4 would appear as a left aligned 10

Message variables issued by REXX checks should be text (character) inserts, except when the value is a true hexadecimal or decimal value, or when the value has been converted to a hex value. Text variables do not require additional translation.

- '000000000A'x when displayed as hex variable would appear as '00000000 0A'
- '1234' when display as hex would appear as 'F1F2F3F4'
- '0A'x when displayed as decimal variable with a fieldsize of 4 would appear as a left aligned 10

If an HZSLFMSG function call is incorrect, the system issues system abend X'290' with a unique reason code and creates a logrec error record. The abend and reason code are included in the check display output. The system checks the following for each HZSLFMSG call:

- That the message is in the message table
- That the number of inserts provided on the call exactly matches the number required to complete the message
- That each variable definition is between 1-256 characters long

The reason codes for system abend X'290' describe the message error. See *z/OS MVS System Codes*.

Reporting check exceptions

When a check detects a system condition or setting that runs counter to the values that the check is looking for, the check should issue an exception message to report the exception. For an exception message, the system displays both the message text and the entire message explanation in the message buffer. The message should include a detailed explanation of the error **and** the appropriate action that the installation should take to resolve the condition. If you are writing a check that

checks for a setting that conflicts with the default for the setting, you should include in your check output information about **why** the check user is getting an exception message for a default setting.

Along with an exception message, IBM Health Checker for z/OS will issue a line showing the severity and the return code for the check. The check will continue to run at the defined intervals, reporting the exception each time until the exception condition is resolved.

The following example shows an exception message issued to the message buffer:

```
CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
START TIME: 05/25/2005 09:42:56.690844
CHECK DATE: 20040703  CHECK SEVERITY: HIGH

* High Severity Exception *

IRRH204E The RACF_SENSITIVE_RESOURCES check has found one or
more potential errors in the security controls on this system.

  Explanation:  The RACF security configuration check has found one or
    more potential errors with the system protection mechanisms.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator and the system auditor.

  System Programmer Response:  Examine the report that was produced by
    the RACF check. Any data set which has an "E" in the "S" (Status)
    column has excessive authority allowed to the data set. That
    authority may come from a universal access (UACC) or ID(*) access
    list entry which is too permissive, or if the profile is in WARNING
    mode. If there is no profile, then PROTECTALL(FAIL) is not in
    effect. Any data set which has a "V" in the "S" (Status) field is
    not on the indicated volume. Remove these data sets from the list
    or allocate the data sets on the volume.

    Asterisks ("****") in the UACC, WARN, and ID(*) columns indicate
    that there is no RACF profile protecting the data set. Data sets
    which do not have a RACF profile are flagged as exceptions, unless
    SETROPTS PROTECTALL(FAIL) is in effect for the system.

    If a valid user ID was specified as a parameter to the check, that
    user's authority to the data set is checked. If the user has an
    excessive authority to the data set, that is indicated in the USER
    column. For example, if the user has ALTER authority to an
    APF-authorized data set, the USER column contains "<Read" to
    indicate that the user has more than READ authority to the data set.

  Problem Determination:  See the RACF System Programmer's Guide and
    the RACF Auditor's Guide for information on the proper controls for
    your system.

  Source:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Reference Documentation:
    RACF System Programmer's Guide
    RACF Auditor's Guide
```

```
  Automation:  None.

  Check Reason:  Sensitive resources should be protected.

END TIME: 05/25/2005 09:43:13.717882  STATUS: EXCEPTION-HIGH
    APF-authorized data set, the USER column contains "
```

The **Check Reason:** field display the default reason in an exception message without installation parameter overrides.

See "Issuing a REXX check exception message" for an example of how to issue an exception message from a REXX check.

Example - Issuing a DIRECTMSG message for a REXX check: For a check that has no message table associated with it, you can issue a check message directly from the check routine, as shown in the example below. REXX sample check SYS1.SAMPLIB(HZSSXCHN) also shows DIRECTMSG calls.

```
/* Set up exception message input for HZSLFMSG */
/* Required input variables: */
HZSLFMSG_REQUEST='DIRECTMSG'
HZSLFMSG_REASON='CHECKEXCEPTION'
HZSLFMSG_DIRECTMSG_ID='UTHH003E'
HZSLFMSG_DIRECTMSG_TEXT='Brief exception summary'
/* Optional input variables: */
HZSLFMSG_DIRECTMSG.EXPL='The exception explanation for UTHR003E'
HZSLFMSG_DIRECTMSG.AUTOMATION='Automation text for UTHR003E'
HZSLFMSG_DIRECTMSG.SOURCE='Source text for UTHR003E'
/* Call HZSLFMSG */
HZSLFMSG_RC = HZSLFMSG()

/* Set up report message input for HZSLFMSG */
HZSLFMSG_REQUEST='DIRECTMSG'
HZSLFMSG_REASON='CHECKREPORT'
HZSLFMSG_DIRECTMSG_TEXT='Single line report message'
/* Call HZSLFMSG */
HZSLFMSG_RC = HZSLFMSG()
```

Writing a check with dynamic severity levels

You can create your check routine to issue check exception messages with a dynamically varying severity level, giving users more control over how exception messages are issued and handled. For example, you might use the dynamic severity function for checks that inspect a system setting value and compare it against a threshold. As the value approaches the high threshold, the check can vary the severity of the exception, depending on how close to the threshold the value is.

For information on how users work with checks enabled with dynamic severity, see “Customizing check exceptions with dynamically varying severity” on page 31

How to enable a check for dynamic severity: In order to allow the customer to use the dynamic severity function do the following:

1. Define check parameters that let the check know what severity exception to issue. For example, if your check looks for parameter 'p', define the following parameters:
   - p_HIGH
   - p_MED
   - p_LOW

- p_NONE

In addition, define the corresponding short forms of the parameter, for example for our case, you would define p_H, p_M, p_L, and p_N parameters. And, because it's always likely that customers could forget the exact parameter specifications, try to build in the obvious mistakes. For example, a customer might try p_HI when the correct parameter is p_HIGH, so we recommend that you also support p_HI and p_NO (as a short version of p_NONE).

We have the following tips for you in defining dynamic severity check parameters: Note that the parameter names you create cannot exceed the 16 character parameter length limit for the ADD/UPDATE CHECK interface!

- Of course, the customer should not specify multiple forms for a given severity, but it is not worth while enforcing that in your check routine. Instead, program your check to accept the longest parameter. In other words, if a customer specifies both p_HIGH and p_H parameters for a check, your check should use the value from p_HIGH.
- If you are writing a new check that exploits dynamic severity, we recommend that you use the suffixed parameters (p_HIGH, p_MED, p_LOW and so on) and do not code a non-suffixed, non-dynamic version of the 'p' parameter.

  On the other hand, if you are upgrading an existing check to use dynamic severity, keep the plain non-suffixed parameter and add the _HIGH, _MED, _LOW, _NONE variants as well. Then make sure to code your check to use the rule that if any of the dynamic severity variants are present, then the non-dynamic severity variant is ignored. That way, you don't have to worry about interpreting cases where the customer specifies both a non-dynamic 'p' version of the parameter and a dynamic p_HIGH version.

  If you are adding dynamic severity parameters for an old check, and none of the existing parameters allow an underscore either within the parameter name or value, you can code your check to assume that the customer is using a dynamic severity specification if you find an underscore within the parameter string.
- Code your check so that the customer does not need to specify a severity for all parameters. They should be able to specify just the ones that they want.

2. Code your dynamic severity check to issue exception messages with the SEVERITY parameter on the HZSFMSG service or using REXX function HZSLFMSG_SEVERITY, as follows:
   - If no dynamic severity parameter is provided and the criterion for an exception based on the parameter is met, issue a severity-system message
   - Else if a p_HIGH parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-high message
   - Else if a p_MED parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-medium message
   - Else if a p_LOW parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-low message
   - Else if a p_NONE parameter is provided and the criterion for an exception based on that parameter is met, issue a severity-none message

   Note that the severity specified on HZSFMSG or HZSLFMSG_SEVERITY overrides the default severity defined for the check when it was added.
3. Add the check with ALLOWDYNSEV=YES specified. See the ADD or ADDREPLACE CHECK paramters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68 or "HZSADDCK macro — HZS add a check" on page 260.

How dynamic severity and the SEVERITY in the check definition interact: As we mention above, a check using dynamic severity overrides the severity specified in the check definition. So that was easy. But just to keep things interesting, note that the implicit WTO handling of exception messages that is derived from the severity in either the check definition or the dynamic check severity being used can be overridden by a WTOTYPE specified in the check definition. See WTOTYPE in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Controlling check exception message WTOs and their automation consequences

By default, IBM Health Checker for z/OS issues DOM requests to delete any check exception message WTOs left behind from previous check iterations. It does this DOMing right before the start of the new check iteration. That means that each time the check generates an exception, it also sends a new exception WTO, which also kicks off any automation actions you've set up for your installation.

So, what if you want more control over check exception WTOs and their automation consequences? For example, let's say you have a check that runs every hour. Now let's say that your check begins generating identical exceptions that you've automated on to prompt a beeper call to your favorite system programmer. You have not yet resolved the exception issue, and the installation policy is to not disable checks generating exceptions. That's just good practice, right? And yet your check might generate a lot of WTOs and beeper calls to that poor system programmer while the issue gets resolved.

That's where DOM control comes in! Starting with z/OS V1R13, IBM Health Checker for z/OS you can use the following functions that help you control whether you want to suppress WTOs and any automation actions they trigger for a check that is generating exceptions:

1. Add your check to the product using the DOM(CHECK) parameter on the HZSPRMxx and MODIFY *hzsproc* command. See ADD or ADDREPLACE CHECK parameters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.
2. From your check you decide when to DOM WTOs from previous check runs using the HZSFMSG REQUEST=DOM macro ("HZSFMSG macro — Issue a formatted check message" on page 307) or the REXX HZSLFMSG_REQUEST='DOM' function "HZSLFMSG function" on page 240.

Realizing the benefits of this function is all in the timing:

- If your check (added with DOM(CHECK)) is generating multiple identical unresolved exceptions, your check can wait to DOM the exception WTO (with HZSFMSG REQUEST=DOM or the REXX HZSLFMSG_REQUEST='DOM' function) until the exception condition is resolved. This way, your check is still running, but the exception WTOs from previous iterations of the check do not get DOMed. That means that exception messages from this check are just recorded in the message buffer and not sent as WTOs that set off automation actions.
- If your check is running successfully and is not generating an exception in a check iteration or is generating different check exceptions between iterations, your check should issue HZSFMSG REQUEST=DOM or the REXX HZSLFMSG_REQUEST='DOM' function to DOM WTOs from previous iterations. That way any subsequent exception will be sent as a WTO and will kick off any defined automation actions.

On the other hand, if you always want to delete WTOs for your check, and never wish to suppress duplicate exception WTOs, you will want to specify or default to DOM(SYSTEM) when you add your check, and let the system take care of DOMing of check WTOs for you!

The well-behaved REXX check - recommendations and recovery considerations

Follow the rules for REXX execs: A well behaved REXX check will adhere to all the rules for writing a REXX exec. See:

- *z/OS MVS Programming: Authorized Assembler Services Guide*
- *z/OS TSO/E REXX User's Guide*
- *z/OS TSO/E REXX Reference*. See the topic on conditions and condition traps for recovery information.

Release any system resources obtained: A REXX check should release any resources it obtains, such as a data set, for example, before the REXX check stops running. The REXX check must also include logic that releases resources when an unexpected non-exception condition, such as a time-out or CANCEL, occurs. For information about how System REXX manages unexpected conditions, see:

- System REXX in *z/OS MVS Programming: Authorized Assembler Services Guide*
- The section on conditions and condition traps in *z/OS TSO/E REXX Reference*.

The following example shows how our SYS1.SAMPLIB check, HZSSXCHK, frees resources for an unexpected condition:

```
/*********************************************************************/
/*                                                                   */
/* HZS_SAMPLE_REXXTSO_CHECK unexpected conditions:                   */
/* SYNTAX, ERROR, FAILURE, NOVALUE and HALT are specified by the     */
/* SIGNAL function and receive control when an unexpected event      */
/* occurs.                                                           */
/*                                                                   */
/*  - Report the line in error                                       */
/*  - Free the input data set if it is allocated                     */
/*  - DISABLE the check and exit                                     */
/*                                                                   */
/*********************************************************************/
 SYNTAX:
 ERROR:
 FAILURE:
 NOVALUE:
 HALT:
ERR1 = "An Error has occurred on line: "Sigl
ERR2 =  sourceline(sigl)
Say Err1
Say "Line "Sigl" text: "Err2
ADDRESS TSO "FREE DSN("DataSetName")"
HZSLFMSG_REQUEST = "STOP"         /* Disable the check               */
HZSLFMSG_REASON = "ERROR"
HZSLFMSG_DIAG   =  Right(RC,16,0) /* report the decimal rc in the
                                     HZS1002E message and the check
                                     display detail                  */
HZSLFMSG_RC = HZSLFMSG()
IF HZS_PQE_DEBUG = 1 THEN
 DO                               /* Report debug detail in REXXOUT  */
   SAY "PARMS: "HZS_PQE_PARMAREA
   SAY "HZSLFMSG RC"  HZSLFMSG_RC
   SAY "HZSLFMSG RSN" HZSLFMSG_RSN
```

```
  SAY "SYSTEMDIAG"   HZSLFMSG_SYSTEMDIAG
END
EXIT               /* The check is not performed          */
```

Have your REXX check stop itself when the environment is inappropriate: If your check encounters an environmental condition that will prevent the check from returning useful results, your check should stop itself and not run again until environmental conditions change and your code requests it to run. Your check should do the following to respond to an inappropriate environment:

1. Issue the HZSLFMSG function to stop itself:

   ```
   HZSLFMSG_REQUEST = "STOP"
   HZSLFMSG_REASON = "ENVNA"
   HZSLFMSG_RC = HZSLFMSG()
   ```

2. Issue an information message to describe why the check is not running. For example, you might issue the following message to let check users know that the environment is not appropriate for the check, and when the check will run again:

   ```
   The server is down.
   When the server is available, the check will run again.
   ```

3. Make sure that your product or check includes code that can detect a change in the environment and start running the check again when appropriate. To start running the check, issue the following HZSCHECK service:

   ```
   HZSCHECK REQUEST=RUN,CHECKOWNER=checkowner,CHECKNAME=checkname
   ```

   If the environment is still not appropriate when your code runs the check, it can always stop itself again.

Save time, save trouble - test your check with these commands: When you have written your check, test it with the following commands to find some of the most common problems people make in writing checks:

```
F hzsproc,UPDATE,CHECK(check_owner,check_name),DEBUG=ON
F hzsproc,UPDATE,CHECK(check_owner,check_name),PARM=parameter,REASON=reason,DATE=date
F hzsproc,DELETE,CHECK(check_owner,check_name),FORCE=YES
F hzsproc,DISPLAY,CHECK(check_owner,check_name),DETAIL
```

Trap internal errors, so the exec routine is not terminated by System REXX

Debugging REXX checks

Naturally, we hope you'll never need this section and that all your checks will run perfectly the very first time. However, if you do run into trouble, the following tips can help:

Look at the documentation for System REXX errors: System REXX may put out some clues to the problem you are having with your checks. Look at the following documentation as appropriate:

- See System REXX abend code X'050' information in *z/OS MVS System Codes*.
- See the System REXX messages in AXR messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.
- See AXREXX Return and reason codes in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

Look for clues to any REXX check problems in the system console log, the logrec data set, and the message buffer.

Make sure your REXX check writes debug information to REXXOUT when running in debug mode: When your REXX check runs in debug mode, the system will write information that can help in check debugging to a REXXOUT data set, if allocated. Information includes TSO error messages, System REXX error messages, and any information you write with the SAY keyword. See “Using REXXOUT data sets” on page 175.

Turn on debug mode: Writing code to capture great debug information in REXXOUT won't help if you don't put the REXX check in debug mode. When a REXX check runs in debug mode the system invokes the REXX check with a REXXOUT dataset. When a REXX check is not in debug mode, the system invokes the REXX check with no REXXOUT dataset, and the debug mode output is not saved

You can turn on debug mode for a REXX check using the DEBUG parameter in the MODIFY *hzsproc* command, in HZSPRMxx, or by overtyping the DEBUG field in SDSF to ON.

Unexpected data in your REXXOUT data set? If your check is running in debug mode, make sure the REXX check has exclusive access to the REXXOUT output data set. See “Using REXXOUT data sets” on page 175.

Chapter 9. Writing an HZSADDCHECK exit routine

For a local or REXX exec check, you can optionally add your check to the system using an authorized HZSADDCHECK exit routine running in the IBM Health Checker for z/OS address space. The HZSADDCHECK exit routine describes the information about your local or REXX exec check or checks. The HZSADDCHECK exit routine invokes the HZSADDCK macro to:

- Identify the check, providing values such as the check owner, check name, check routine name, and message table name.
- Specifies the default values for the check, such as the check interval, check parameter, and check severity.

You can write an HZSADDCHECK exit routine in either Metal C or assembler, regardless of which language your check routine is written in.

Note that you can also add a check to the system using the ADD | ADDREPLACE CHECK parameter in an HZSPRMxx parameter to define check defaults for a local or REXX exec check. This is the method of choice for REXX exec check. See the ADD or ADDREPLACE CHECK parameter in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

You cannot add remote checks to the system with a HZSADDCHECK exit routine. See "Issue the HZSADDCK macro to define a remote check to IBM Health Checker for z/OS" on page 140.

To reduce system overhead and simplify maintenance, we suggest that you create one HZSADDCHECK exit routine for all the checks for your component or product.

Sample HZSADDCHECK exit routines

For a Metal C sample HZSADDCHECK exit routine, look for hzscadd.c in /usr/lpp/bcp/samples.

For an assembler sample HZSADDCHECK exit routine, look for HZSSADCK in SYS1.SAMPLIB.

You'll also find these and more check samples on the IBM Health Checker for z/OS Web page:
`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

Metal C or assembler?

As mentioned above, you can write an HZSADDCHECK exit routine in either Metal C or assembler. The concepts in this section apply to either language. Metal C lets you create a IBM Health Checker for z/OS compatible load module that follows MVS linkage conventions, just as you would using assembler. You can also easily use assembler macros, such as HZSFMSG or any other assembler macro, using the _asm keyword. If you are writing in Metal C, IBM Health Checker for z/OS provides generic C header files in SYS1.SIEAHDRV containing:

- Mappings of IBM Health Checker for z/OS structures and control blocks
- Commonly used Health Checker related constants

When the HZSADDCHECK exit calls the exit routines to add checks to the system, the system processes the default values from the HZSADDCK macro call, and applies any installation updates to the defaults.

Use the following guidelines in defining defaults for your check in the HZSADDCHECK exit routine:

- Use the **"HZSADDCK macro — HZS add a check" on page 260** in your HZSADDCHECK exit routine to describe your check.
- **The CHECKOWNER field should reflect both the company and component or product name:** For quick identification of checks, we suggest that the owner field include a company identifier and component or product name. For example, CHECKOWNER name IBMGRS reflects both the company and component that owns the check.
- **Define a meaningful CHECKNAME for your check:** Create a meaningful, descriptive name for your check. Your CHECKNAME should start with a component or product prefix so that you can easily identify where a check comes from. In addition, using the prefix ensures that all the checks for a particular component or product will be grouped together in an SDSF check display, if supported on your system. For example, IBM's virtual storage management (VSM) checks all start with VSM. (See Chapter 13, "IBM Health Checker for z/OS checks," on page 381.)
- **Using the DATE parameters:** The HZSADDCK DATE parameter specifies when the setting or value being checked was defined. This will alert customers to check the installation updates for this check. An installation update also has an associated date, and when the installation update date is older than the DATE parameter specified on HZSADDCK, the system:
  - Does not apply the update
  - Issues a message to inform the installation of the circumstance.

  If you change your check, you should update the HZSADDCK DATE parameter only if you want to make sure that the installation takes a look at your check again to make sure any installation updates are still appropriate.
- **Assign a severity to your check based on the problems your check is looking for** and how critical they are. The severity you choose will determine how the system handles the exception messages that your check routine issues with the HZSFMSG service:
  - SEVERITY(HIGH) indicates that the check routine is checking for high-severity problems in an installation. All exception messages that the check issues with the HZSFMSG service will be issued to the console as critical eventual action messages.
  - SEVERITY(MEDIUM) indicates that the check is looking for problems that will degrade the performance of the system. All exception messages the check issues with HZSFMSG will be issued to the console as eventual action messages.
  - SEVERITY(LOW) indicates that the check is looking for problems that will not impact the system immediately, but that should be investigated. All exception messages the check issues with HZSFMSG will be issued to the console as informational messages.

  Installations can update the SEVERITY value in the HZSADDCHECK exit routine using either the SEVERITY or WTOTYPE parameter in an installation update.
- **Selecting an INTERVAL and EINTERVAL for your check:** Keep the following in mind when selecting an interval for a check:

  - The INTERVAL parameter specifies how often the check will run. But you can also specify an exception interval (EINTERVAL), which lets you specify a more frequent interval for the check to run if it has raised an exception.
  - A check INTERVAL must be 1 minute or longer.
  - The specified INTERVAL or EINTERVAL time starts ticking away when a check finishes running.
- **Specify parameters for your REXX exec check:** For a REXX exec check, there are some special HZSADDCK keywords:
  - EXEC(*execname*) - This parameter, required for a REXX check defined in the HZSPRMxx parmlib member, specifies the name of the REXX exec containing the REXX check or checks. This parameter tells the system that you are defining a REXX check. For an assembler check, you would specify the CHECKROUTINE(*checkname*).

    If you define your REXX check with the HZSADDCK macro in an HZSADDCHECK exit routine, the equivalent of EXEC(*execname*) is the REXX=YES,EXEC=execname parameters.
  - REXXHLQ(*hlq*) - This parameter, required for a REXX check, specifies the high level qualifier for any input or output data set for the check.
  - REXXTIMELIMIT(*timelimit*) - This optional input parameter specifies the number of seconds a check iteration is allowed to run before the system ends it. A value of 0, which is the default, specifies that there is no time limit for the check.
  - REXXTSO(YES | NO) - This parameter, optional for a REXX check, specifies whether the check runs in a TSO environment or a non-TSO environment. The default is REXXTSO(YES).
    - REXXIN(YES | NO) - This parameter, optional for a REXX check, specifies whether or not a non-TSO check requires an sequential input data set. The name of the REXXIN data set will consist of the high level qualifier specified in the HLQ parameter, the exec name specified in the EXEC parameter, and an optional entry code specified in the ENTRYCODE parameter.

      You can only specify REXXIN(YES) if you also specify REXXTSO(NO).
- **Specify whether your check requires UNIX System Services:** Use the USS keyword to specify whether your check requires UNIX System Services. Any check that uses UNIX System Services such as DUB should specify USS=YES. If you specify USS=YES for a check, the system will run the check only when UNIX System Services are available.
- **Specify an ENTRYCODE for your check if there are multiple checks in a check routine:** Use the ENTRYCODE parameter to specify a unique entry code for a specific check if multiple checks invoke the same check routine or REXX check. The routine or REXX exec must contain logic to determine which check the system is calling by checking the entrycode. The entrycode is passed to the check routine in the field Pqe_EntryCode in the HZSPQE mapping macro.
- **Making your HZSADDCHECK exit routine reentrant:** Your HZSADDCHECK exit routine will be reentrant, so you must use the LIST and EXECUTE forms of the HZSADDCK macro and any other z/OS macros with parameter lists.
- **HZSADDCHECK exit routine return codes less than eight indicate success:** We hate to say to ignore return codes, but in general there's really no need to worry about HZSADDCHECK exit routine return codes. For any problem requiring attention, the system issues an abend code. See “Debugging HZSADDCHECK exit routine abends” on page 197.

Programming considerations for the HZSADDCHECK exit routine

Environment

IBM Health Checker for z/OS calls the HZSADDCHECK exit routine in primary mode from the IBM Health Checker for z/OS address space.

- **Address space:** IBM Health Checker for z/OS
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=SASN=HASN
- **AMODE:** 31
- **ASC mode:** Primary
- **Key:** System defined. The system will give control to the exit routine in the same key in which it gives control to the check routine.
- **State:** Supervisor
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Control parameters:** Control parameters are in the IBM Health Checker for z/OS address space

Note that HZSADDCHECK exit routines are loaded in common. The exit routine should be a single csect load module.

The message table is loaded in Health Check private, it should be a single csect load module.

Input Registers

When an HZSADDCHECK exit routine receives control, the contents of the registers are as follows:

Register
Contents

Register 0 - 12
Not applicable

Register 13
Points to the address of a 72 byte save area

Register 14 - 15
Not applicable

When an HZSADDCHECK exit routine receives control, the contents of the access registers (ARs) are as follows:

Register
Contents

Register 0 - 12
Not applicable

Register 13
Points to the address of a 72 byte save area

Register 14 - 15
Not applicable

Output Registers

When a HZSADDCHECK exit routine r returns control, the contents of the registers must be:

Register
Contents

Register 0 - 1
The exit routine does not have to place any information in this register, and does not have to restore its contents to what they were when the exit routine received control

Register 2 - 13
Unchanged

Register 14 - 15
The exit routine does not have to place any information in this register, and does not have to restore its contents to what they were when the exit routine received control.

When a HZSADDCHECK exit routine r returns control, the contents of the access registers (ARs) must be:

Register
Contents

Register 0 - 1
The exit routine does not have to place any information in this register, and does not have to restore its contents to what they were when the exit routine received control

Register 2 - 13
Unchanged

Register 14 - 15
The exit routine does not have to place any information in this register, and does not have to restore its contents to what they were when the exit routine received control

Defining multiple local or REXX checks in a single HZSADDCHECK exit routine

To reduce system overhead and simplify maintenance, you can and should define multiple uniquely-named checks in a single HZSADDCHECK exit routine. Defining multiple checks in one HZSADDCHECK exit routine will streamline the identification and registration process for a component or product, so that you need only one HZSADDCHECK exit routine and one check routine for your checks.

If you put multiple checks in one check routine (which is recommended), use the **ENTRYCODE** parameter on HZSADDCK to assign an entry code to each check. For a non-REXX check, the entry code is passed to the check routine in the PQE_ENTRY_CODE field in the HZSPQE mapping macro. For a REXX check, it is passed to the check exec in REXX variable HZS_PQE_ENTRY_CODE.

Note that the IBM Health Checker for z/OS will not verify the uniqueness of the entry codes you define for your checks.

Dynamically adding local or REXX exec checks to IBM Health Checker for z/OS

Once you've written the check routine and the HZSADDCHECK exit routine r for your checks, you must then add the checks to IBM Health Checker for z/OS so that the system can run the check. To do this, you must add the HZSADDCHECK exit routine r to the HZSADDCHECK exit and then have the system call the exit to run the exit routine. There are two approaches to this step:

- When your check is ready for production, you will add the code to your product or component to activate your checks when your product starts. See "Creating product code that automatically registers checks at initialization" on page 197.
- For **testing purposes**, you can add the HZSADDCHECK exit routine to the system dynamically with either operator commands or in a program, as we will show in this section. See:
  - "Using operator commands to add checks to the system dynamically"
  - "Using a routine to add checks to the system dynamically" on page 197

  You can also simply define the check defaults and values directly in an HZSPRMxx parmlib member and bypass the HZSADDCHECK exit routine entirely. See the ADD or ADDREPLACE CHECK parameters in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Once you have added your check to the system using one of these methods, you can use a command such as the following to verify that it is there:

```
F hzsproc,DISPLAY CHECK(checkowner,checkname),DETAIL
```

Once a check has been added to the system, it will remain active and will run at the specified interval until:

- A user explicitly deactivates or deletes the check, issuing a MODIFY command for example. See "Making dynamic, temporary changes to checks" on page 44.
- IBM Health Checker for z/OS disables a check because of check routine or environmental problems. See "IBM Health Checker for z/OS controlled states" on page 36.

Using operator commands to add checks to the system dynamically

1. Make the check routine, HZSADDCHECK exit routine r, and message table available to either:
   - The LNKLST set being used by the IBM Health Checker for z/OS address space
   - The current LNKLST if the IBM Health Checker for z/OS address space is not currently active

   If the check routine, HZSADDCHECK exit routine r, and message table belong in SYS1.LINKLIB, one way you can make them available to LNKLST is to:
   a. Copy them into SYS1.LINKLIB
   b. Update LLA to indicate that the new parts are available by adding a statement to a CSVLLAxx parmlib member containing the name of the parts for your check. For example, you might update a CSVLLAxx parmlib member with the following statement:

      ```
      LIBRARIES(SYS1.LINKLIB) MEMBERS(checkroutine,hzsaddcheckexitroutine,messagemod)
      ```

c. Issue the MODIFY LLA,UPDATE=*xx* command to have the system use the updated CSVLLAxx parmlib member.

2. Issue the SETPROG command to add the HZSADDCHECK exit routine r to the HZSADDCHECK exit. The following example shows how we add the exit routine, HCEXIT, to the HZSADDCHECK exit:

   ```
   SETPROG EXIT,ADD,EXITNAME=HZSADDCHECK,MODNAME=HCEXIT
   ```

   Note that instead of using SETPROG, you can place the analogous EXIT statement in a PROGxx parmlib member and issue the SET PROG=*xx* command.

3. Issue the MODIFY command to add all new checks to the system:

   ```
   F hzsproc,ADDNEW
   ```

Using a routine to add checks to the system dynamically

You can create a routine to add checks dynamically to the system in either Metal C or assembler.

The following examples show the logic of a routines that:

- **1** Add exit routine, HCEXIT, to the HZSADDCHECK exit.
- **2** Issue the HZSCHECK service to call the HZSADDCHECK exit to run the exit routine and add checks.

```
 CSVDYNEX REQUEST=ADD,
           1
          EXITNAME=CL16'HZSADDCHECK',
          MODNAME=CL8'HCEXIT',
          MESSAGE=ERROR,
          RETCODE=RC,
          RSNCODE=RS

CHI   R15,8
JNL  NO_ADDNEW
 2
HZSCHECK REQUEST=ADDNEW,
          RetCode=RC,
          RsnCode=RS
```

Figure 12. Assembler example: Defining the HZSADDCHECK exit routine r and adding checks

Debugging HZSADDCHECK exit routine abends

If your HZSADDCHECK exit routine abends, the exit routine becomes inactive and the system issues message CSV430I identifying the exit routine and the abend so that you can debug the problem.

Creating product code that automatically registers checks at initialization

In Chapter 9, “Writing an HZSADDCHECK exit routine,” on page 191, we talked about registering checks for testing purposes. Ultimately, however, you'll probably want your component or product to automatically look for and activate any new checks when it initializes. We're doing this for some of our IBM products by making check registration part of the initialization processing for the product. Add code similar to the following assembler example to define the HZSADDCHECK exit routine to IBM Health Checker for z/OS and look for and activate new checks:

```
CSVDYNEX REQUEST=ADD
          EXITNAME=HZSADDCHECK,     /* HEALTH CHECKER name  */
          MODNAME=IBMHCADC,         /* check defintition
                                          exit routine        */
          MESSAGE=ERROR,
          RetCode=HCRetCode,
          RsnCode=HCRsnCode

IF HCRetCode = 0 Then               /* Tell Health Checker to */
HZSCHECK REQUEST=ADDNEW,           /* Look for new checks   */
          RetCode=HCRetCode,
          RsnCode=HCRsnCode
```

Creating product code that deletes checks as it goes down

If your component or product stays up for the life of the system (GRS, for example), you do not need to do any check deletion or clean up. However, if your product or component does occasionally come up and down you might want to delete the checks as you come down. Code the exit routine to issue the HZSADDCK macro to add the check or checks only if the product is up.

Chapter 10. Creating the message input for your check

Check messages are the output of your check routine - they communicate the results uncovered by a check to the user. Your check messages should:

- Report any exceptions to the values and settings the check is looking for and convey recommendations for actions to take to correct the exception. Depending on what the check is designed to do, an exception message may report risks to system performance, function, availability, or integrity. A check should also report dynamic changes since the last IPL that pose a potential problem, and which might make the next IPL slower or error-prone.
- Report that the check found no exceptions, if appropriate.
- If the setting the check is looking for conflicts with existing default settings, explain why the check user is getting an exception message for a default setting.
- Create report messages that displays data that the check collects.

This chapter covers how to create a message table for your check. However, you can also omit the message table and issue messages directly from the check routine as follows:

- For local or remote checks, use HZSFMSG REQUEST=DIRECTMSG, providing the message text in the HZSFMSG REQUEST=DIRECTMSG request. See "HZSFMSG macro — Issue a formatted check message" on page 307.
- For REXX checks, use HZSLFMSG_REQUEST='DIRECTMSG', providing the message text right in the HZSLFMSG_REQUEST='DIRECTMSG' input variables. See "Input variables for HZSLFMSG_REQUEST='DIRECTMSG'" on page 242.

For more information, see our fabulous Redpaper! For lots of details and experience based information about creating messages for your check, see *Exploiting the Health Checker for z/OS infrastructure* (REDP-4590-00).

To create a message table for your check, you must do the following:

1. **Plan your check messages** to be easy to understand, use, and automate. See "Planning your check messages" on page 201.
2. **Create a message table** that contains both message texts **and explanations** for checks. See "Creating the message table" on page 205.
3. **Optionally create a setup data set** customized for your checks. The setup data set contains entries for symbols, frequently used text strings used in messages, and for the books you reference in your message explanation. (Every message must contain a reference to a book for more information.) See "Defining your own symbols for check messages" on page 229
4. **Generate the messages** into a compilable assembler CSECT using a JCL procedure using the HZSMSGEN REXX exec. The HZSADDCHECK exit routine passes the name of the message table for a check to IBM Health Checker for z/OS. See "Generating the compilable assembler CSECT for the message table" on page 231.
5. **Compile the message CSECT to create the message table module**, which you will ship with the compiled check routine and HZSADDCHECK exit routine, if you have one for the check. Make sure that you link edit the message table as reentrant. In addition, Make sure that the message table is in a single separate module, rather than mainline code, to make maintenance and corrections easier.

For local and REXX checks, the system loads the message table into IBM Health Checker for z/OS private storage. Remote checks must load their message table into their own address space's storage.

6. **From your check routine:**
   - **Remote checks must load the message table into their own storage.**
   - **Define the variables for your messages**. In your check routine, you will define and store message variable data into the HZSMGB data area. See “Defining the variables for your messages” on page 120. You can have up to 20 message variables per message, each used one time only.
   - **Use the HZSFMSG macro to issue messages** . See “Issuing messages in your local check routine with the HZSFMSG macro” on page 117.

The following figure shows how all the parts fit together in the process of creating a message table:

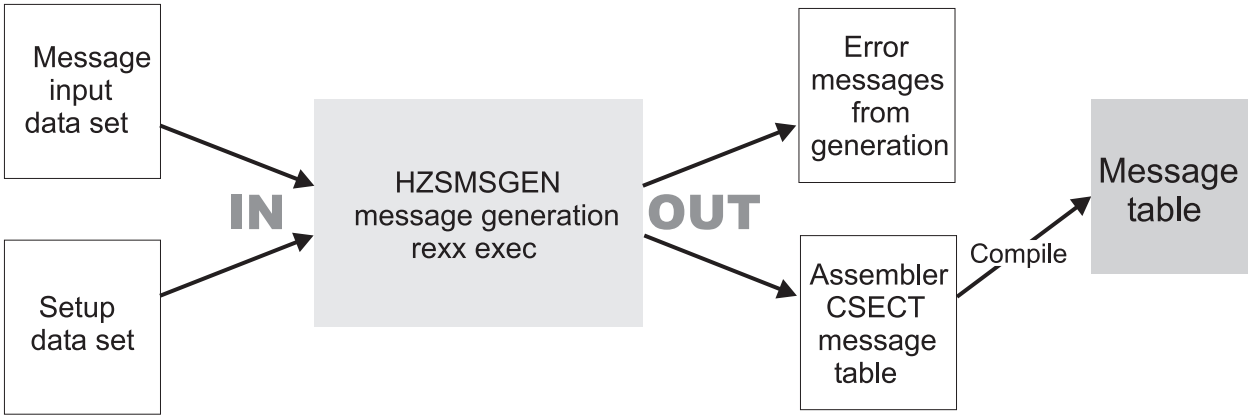


Figure 13. Inputs and outputs for creating a complete message table

How messages and message variables are issued at check runtime

When a check runs, it issues messages using the HZSFMSG macro to relate the results of the check. Most of the data for the messages comes from the generated and compiled message table. However, if a message issues dynamic variables (<mv></mv> tags), the variables work as follows:

- The values for the variables must be defined in the HZSMGB data area for the check, which contains an array of pointers to variables.
- The address for the HZSMGB data area for a check is specified on the MGBADDR parameter of the HZSFMSG macro. For example:

  ```
  HZSFMSG REQUEST=CHECKMSG,MGBADDR=MGB_PTR
  ```

- The message table describes the attributes of the variable, which determine how the variable is formatted. See "Variables for message text" in “Message text (<msgtext>) and message variable (<mv>) tags” on page 218.
- At runtime, when the HZSFMSG macro is issued, the IBM Health Checker for z/OS gets the text of the message variable from the address pointed to in the MGB_MsgInsertAddr field in data area HZSMGB.

Figure 14 on page 201 shows how messages with variables get resolved at check runtime.

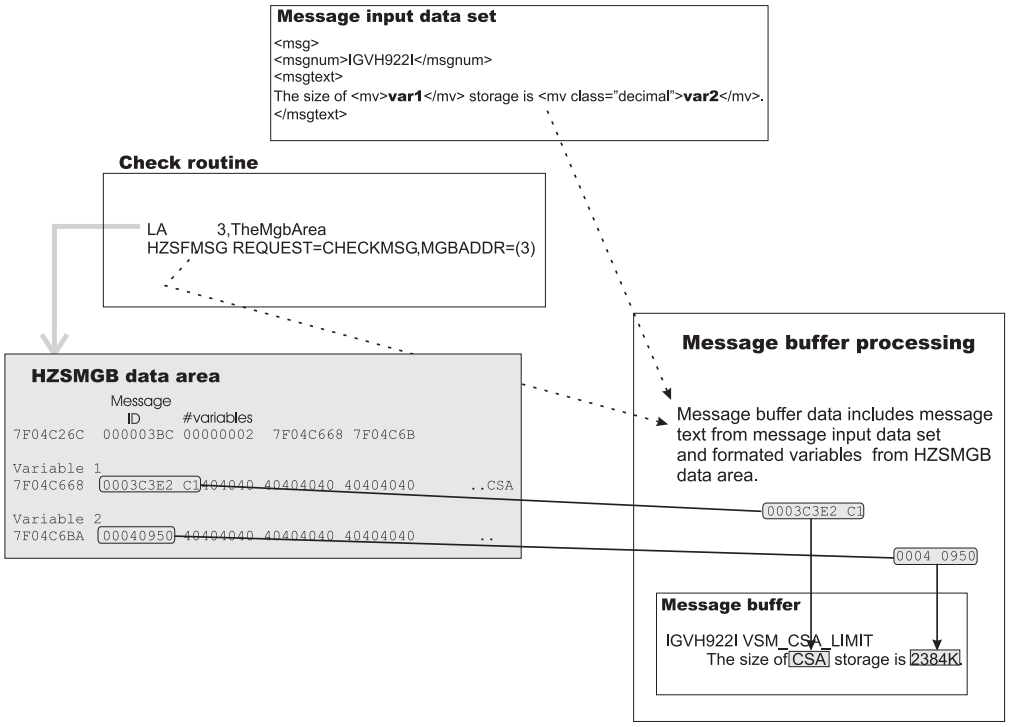


Figure 14. Message and variable resolution at runtime

Planning your check messages

If you've used IBM Health Checker for z/OS, you know that the messages issued by the check are the most important part of the check, because they notify installations of potential problems and suggested fixes for those problems.

You can issue several types of messages in your check routine. Use the following sections to plan for the types of messages your check will issue:

- "Planning your exception messages" on page 202
- "Planning your information messages" on page 202
- "Planning your report messages" on page 203
- "Planning your debug messages" on page 203

In addition, the system issues **predefined environment and parameter error** when a check issues the HZSFMSG REQUEST=STOP service after finding a parameter or environmental error that prevents the check from running. You do not have to define these messages in your message table - when you issue the HZSFMSG REQUEST=STOP service, the system issues an IBM Health Checker for z/OS HZS100*x*E error message. See "The well-behaved local check routine - recommendations and recovery considerations" on page 127 and "Recommendations and recovery considerations for remote checks" on page 161 for information.

The system also issues **parameter parser error messages** - see the HZSFMSG REASON=PARS*nnnn* parameters in "HZSFMSG macro — Issue a formatted check message" on page 307.

You must decide the following when planning your messages:

- "Decide what release your check will run on" on page 203
- "Decide whether to translate your check exception messages into other national languages" on page 204
- "Rely on IBM Health Checker for z/OS to issue basic check information for you" on page 204

Planning your exception messages

Your check will issue exception messages when a check detects a deviation from a suggested setting. See "Understanding exception messages" on page 28 for how an exception message looks to users.

- The **message text** of an exception message is a WTO and should be designed to alert an installation to a condition that requires the attention of a system programmer. The audience for this exception WTO is the operator, so it should simply include enough information to identify the system resource that requires attention. For example, the following exception message text explains just enough to let operators know what kind of a problem the check has uncovered and who they might need to contact:

  ```
  IRRH204E The RACF_SENSITIVE_RESOURCES check has found one or
  more potential errors in the security controls on this system.
  ```

  Both for quick identification and to facilitate automation, IBM Health Checker for z/OS precedes the message text WTO with a HZS000xx message that displays the check name and exception severity.
- The **message details** for an exception message, with its multiple categories such as explanation, operator response, and system programmer response, is issued to the message buffer. The explanation should provide details about what the check was looking for, the exception condition it found, and the impact that the condition might have on the system. The audience for the explanation is the system programmer, who will appreciate very specific input on how to correct the problem in the system programmer response. For example, your exception system programmer response might include correct command syntax. You might also include a pointer to documentation about a suggested solution, although ideally you can outline a complete solution right in the system programmer response in the check's message buffer.

  See "Message Table" on page 222 for the categories you'll need to include in your exception message details.

See "Exception message example" on page 205.

Planning your information messages

Your check can issue non-exception informational messages for the following reasons:

- Every exception message requires an informational message for the clean-run, no-exceptions-found case.
- Use an information message to report that the check cannot run because of parameter errors, or because it is inappropriate for the current installation environment. If you issue an informational message for a check that has stopped itself because of parameter errors, describe the correct syntax of the parameters.
- Use an informational message as a report title. The report title informational message should describe the report message, including the variables.

The explanation details that you code in your message table for informational messages should be as complete possible, because no one wants to have to look

the message up to figure out what to do. Go ahead, you've got the whole message buffer to explain your message! For example, you should include:

- The message explanation
- The product source of the message
- Any system action, even if it's just that the system continues processing
- The operator response, which is often to notify the system programmer.
- The system programmer response, with details on any problem with, for example, parameters in error.
- The detecting module for the message, information that might be helpful for your support people.

See "Information message example" on page 207.

Planning your report messages

A report message is a tabular form message issued to the message buffer, often to display supplementary information for an exception message. A report message give you more control over the formatted message output than any other check message type. Use a report message if your check has a lot of data to display about an exception to avoid issuing multiple exception WTO messages for a single check iteration. (WTOs can be a performance/resource issue.)

Your check should issue a report message **before** the exception message it supplements. In addition, your check should issue an informational title message before the report that includes the entire explanation for the report, including the meaning of variables, because a report message is not documented anywhere. There is no message number associated with a report message in the message buffer (except in debug mode viewed in SDSF).

The key rule for reports issued by checks is to **make sure your report can stand on its own**. In other words, for the sake of IBM Health Checker for z/OS users, make sure that your report is as clear and self-explanatory as possible.

See "Report message example" on page 208.

Planning your debug messages

Your check can issue debug messages to display extra information about the check to aid in testing and diagnosis when the check is in debug mode. See "The well-behaved local check routine - recommendations and recovery considerations" on page 127 for information about using debug mode. For a debug message, only the message text (<msgtext> field) is issued to the message buffer. See also "Debug message example" on page 212.

Decide what release your check will run on

The release your check will run on determines a couple of things about how you define your messages, particularly the message list, in the message table. So you must determine in advance whether:

- If your check runs only on z/OS V1R10 or higher level systems, **and** you want to use function that apply only to z/OS V1R10 systems or higher (see "Message list tag - <msglist>" on page 215 for R8 enhancments), specify your message list with a rules level of 3. Note that rules level 3 includes all the function of rules level 2.

  ```
  <msglist xreftext="csectname rules=3">
  ```

- If your check runs only on z/OS V1R8 or higher level systems, **and** you want to use function that apply only to z/OS V1R8 systems or higher (see "Message list tag - <msglist>" on page 215 for R8 enhancments), specify your message list with a rules level of 2:

  ```
  <msglist xreftext="csectname rules=2">
  ```

- If your check will run on z/OS V1R7 or lower level systems, you must specify a rules level of:

  ```
  <msglist xreftext="csectname rules=1">
  ```

  You can also use a rules level of 1 on a z/OS V1R8 or higher system, as long as you do not use V1R8 enhancements in your message list.

See "Message list tag - <msglist>" on page 215 for complete information.

Decide whether to translate your check exception messages into other national languages

If you want to translate your check exception messages into other national languages, there are a couple of things you will need to keep in mind as you code your exception message texts. For example in order to be translated, message texts must be 71 characters or less, so you must break up the WTO exception message text lines with <lines></lines> tags. For complete information about calculating line lengths for your exception message text and how to break up lines, see "Message text (<msgtext>) and message variable (<mv>) tags" on page 218. For information on using MMS, see Translating messages in *z/OS MVS Programming: Assembler Services Guide*.

IBM Health Checker for z/OS can generate the skeletons for your messages at the same time you generate the message table CSECT using the HZSMSGEN exec, see "Support for translating messages to other languages" on page 232.

Rely on IBM Health Checker for z/OS to issue basic check information for you

You should never need to issue basic information about your check such as check name, because IBM Health Checker for z/OS will automatically issue this kind of information about your check in both the message buffer and, for exception messages, in the WTO.

- In the message buffer, IBM Health Checker for z/OS issues information for all messages such as check owner and name, check date, start time, and end time. An exception message also includes additional information about values defined for the check, such as the check reason, check parameters in use (if any). See"Exception message example 1" in "Understanding exception messages" on page 28 and "Exception message example" on page 205.
-
- In a WTO for the exception message, IBM Health Checker for z/OS prefixes the exception with an HZS message stating the check owner and name:

  ```
  SYS1  HZS0002E CHECK(IBMXCF,XCF_SFM_ACTIVE):
  IXCH0514E The state of Sysplex Failure Management is NOT consistent
  with the IBMXCF recommendation.
  ```

Creating the message table

Sample message table

For a sample message table, see sample HZSSMSGT in either SYS1.SAMPLIB or on the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

In the message table, you'll provide both the message text and the explanation for each message.

- For exception messages, the entire message, including text and detail (<msgtext> and all the <msgitem> tags), will appear in the message buffer, viewable using either HZSPRINT or SDSF. However, only the message text (<msgtext> tag) is included in the WTO. In the message text, convey the potential problem uncovered. In the detailed explanation, convey the suggested solution to the problem.
- For non-exception messages, only the message text will appear in the message buffer.

The finished message table will be an interface which installations can automate.

This section covers the following:

- "Examples of message input"
- "Syntax of message input" on page 214
  - "Special formatting tags for the message table" on page 223
  - "How messages are formatted in the message buffer" on page 225
  - "Using symbols in the message table" on page 227

Examples of message input

Exception message example

The following shows an example of a complete check exception message formatted as it would be in the message buffer. The suffix of E indicates it's an exception message, and that the reported situation will require action.

```
CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
START TIME: 05/25/2005 09:42:56.690844
CHECK DATE: 20040703  CHECK SEVERITY: HIGH

* High Severity Exception *

IRRH204E The RACF_SENSITIVE_RESOURCES check has found one or
more potential errors in the security controls on this system.

  Explanation:  The RACF security configuration check has found one or
    more potential errors with the system protection mechanisms.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator and the system auditor.

  System Programmer Response:  Examine the report that was produced by
    the RACF check. Any data set which has an "E" in the "S" (Status)
    column has excessive authority allowed to the data set. That
```

```
    authority may come from a universal access (UACC) or ID(*) access
    list entry which is too permissive, or if the profile is in WARNING
    mode. If there is no profile, then PROTECTALL(FAIL) is not in
    effect. Any data set which has a "V" in the "S" (Status) field is
    not on the indicated volume. Remove these data sets from the list
    or allocate the data sets on the volume.

    Asterisks ("****") in the UACC, WARN, and ID(*) columns indicate
    that there is no RACF profile protecting the data set. Data sets
    which do not have a RACF profile are flagged as exceptions, unless
    SETROPTS PROTECTALL(FAIL) is in effect for the system.

    If a valid user ID was specified as a parameter to the check, that
    user's authority to the data set is checked. If the user has an
    excessive authority to the data set, that is indicated in the USER
    column. For example, if the user has ALTER authority to an
    APF-authorized data set, the USER column contains "<Read" to
    indicate that the user has more than READ authority to the data set.

  Problem Determination:  See the RACF System Programmer's Guide and
    the RACF Auditor's Guide for information on the proper controls for
    your system.

  Source:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Reference Documentation:
    RACF System Programmer's Guide
    RACF Auditor's Guide

  Automation:  None.

 Check Reason: Sensitive resources should be protected.

END TIME: 05/25/2005 09:43:13.717882  STATUS: EXCEPTION-HIGH
```

Note that fields such as **START TIME:**, **CHECK DATE:**, **Check Reason:** and **END TIME:** are not part of the message input specified by the check developer. The system issues these automatically, as appropriate. See “Extra fields issued to the message buffer for exception messages” on page 226 for more information.

You must code your message input with tags. The following example shows how the example message, IRRH204E, looks coded in the tag format. This example also shows the use of a symbol, **&hzsckname;**, for the check name - see “Using pre-defined system symbols” on page 227 for more information.

```
<msglist xreftext="csectname" rules="1">
<msg class="Exception">
<msgnum xreftext="204">IRRH204E</msgnum>
<msgtext>
The &hzsckname; check has found one or
<lines>
more potential errors in the security controls on this system.
</lines>
</msgtext>
<msgitem class="explanation"><p>
The RACF security configuration check has found one or more
potential errors with the system protection mechanisms.
</p></msgitem>
<msgitem class="sysact"><p>
The check continues processing. There is no effect on the system.
</p></msgitem>
<msgitem class="oresp"><p>
```

```
Report this problem to the system security administrator and the
system auditor.
</p></msgitem>
<msgitem class="spresp"><p>
Examine the report that was produced by the RACF check. Any data
set which has an "E" in the "S" (Status) column has excessive authority
allowed to the data set. That authority may come from a universal access
(UACC) or ID(*) access list entry which is too permissive, or if the
profile is in WARNING mode. If there is no profile, then
PROTECTALL(FAIL) is not in effect.
Any data set which has a "V" in the "S" (Status) field is not on
the indicated volume. Remove these data sets from the list or allocate
the data sets on the volume.
</p>
<p>Asterisks ("****") in the UACC, WARN, and ID(*) columns indicate that
there is no RACF profile protecting the data set. Data sets which
do not have a RACF profile are flagged as exceptions, unless
SETROPTS PROTECTALL(FAIL) is in effect for the system.
</p>
<p>If a valid user ID was specified as a parameter to the check, that
user's authority to the data set is checked. If the user has an
excessive authority to the data set, that is indicated in the USER
column. For example, if the user has ALTER authority to an
APF-authorized data set, the USER column contains
"<Read" to indicate
that the user has more than READ authority to the data set.
</p></msgitem>
<msgitem class="probd"><p>
See the RACF System Programmer's Guide and the RACF Auditor's
Guide for information on the proper controls for your system.
</p></msgitem>
<msgitem class="source"><p>
<lines>
RACF System Programmer's Guide
RACF Auditor's Guide
</lines>
</p></msgitem>
<msgitem class="refdoc"><p>
<lines>
RACF System Programmer's Guide
RACF Auditor's Guide
</lines>
</p></msgitem>
<msgitem class="automation"><p>
None.
</p></msgitem>
<msgitem class="module"><p>
IRRHCR00
</p></msgitem>
<msgitem class="rcode"><p>
</p></msgitem>
<msgitem class="dcode"><p>
</p></msgitem>
</msg>
     .
     .
     .
</msglist>
```

Note that tags <msgitem class="rcode"> and <msgitem class="dcode"> are coded in the message table, but are not displayed in the message buffer.

Information message example

The following example shows an information message text as it would appear in the message buffer:

```
CHECK(IBMCNZ,CNZ_CONSOLE_MASTERAUTH_CMDSYS)
START TIME: 06/01/2005 09:43:42.219863
CHECK DATE: 20040816  CHECK SEVERITY: LOW

CNZHS0002I At least one active console has MASTER authority and command
association to system JA0.

END TIME: 06/01/2005 09:43:42.225214  STATUS: SUCCESSFUL
```

The following example shows how the example message looks coded in the tag format. Note that the same message explanation tags are required in an information message as are in an exception message, although they do not show up in the message buffer and they do not appear in this example. This example also shows the use of a symbol, **&hzssysname;**, for the system name - see "Using pre-defined system symbols" on page 227 for more information.

```
<msglist xreftext="csectname" rules="1">
<msg class=information>
<msgnum xreftext=201>CNZHS0002I</msgnum>
<msgtext>
At least one active console has MASTER authority and command
association to system &hzssysname;.
</msgtext>
<msgitem class="explanation"><p>
n/a</p>
</msgitem>
<msgitem class="sysact"><p>
n/a</p>
</msgitem>
<msgitem class="oresp"><p>
n/a</p>
</msgitem>
<msgitem class="spresp"><p>
n/a</p>
</msgitem>
<msgitem class="probd"><p>
n/a</p>
</msgitem>
<msgitem class="source"><p>
n/a</p>
</msgitem>
<msgitem class="refdoc"><p>
n/a</p>
</msgitem>
<msgitem class="automation"><p>
n/a</p>
</msgitem>
<msgitem class="module"><p>
n/a</p>
</msgitem>
<msgitem class="rcode"><p>
n/a</p>
</msgitem>
<msgitem class="dcode"><p>
n/a</p>
</msgitem>
</msg>
     .
     .
     .
</msglist>
```

Report message example

- **A report message:** The following example shows a report message text as it would appear in the message buffer.

```
CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
START TIME: 06/01/2005 12:50:57.749916
CHECK DATE: 20040703  CHECK SEVERITY: HIGH

                          APF Dataset Report

S Data Set Name                             Vol    UACC Warn ID*  User
- ------------------------------------ ------ ---- ---- ---- ----
E SYS1.LINKLIB                              PETPB1 Updt No   ****
E SYS1.SVCLIB                               PETPB1 Updt No   ****
E SYS1.SIEALNKE                             PETPB1 Updt No   ****
                               .
                               .
                               .
  TCPIP.V3R2M0.SEZALINK                     D83AE8 Read No   ****
```

This example shows that:

- The report actually consists of four messages, IRRH255R through IRRH258R. However, unless you run the check in debug mode and use SDSF to display messages, the message identifier is not displayed for report messages.
- The first three messages, IRRH255R through IRRH257R contain the report title lines, with the report name supplied by a variable.
- The fourth message, IRRH258R, contains the report data as a series of variables.
- The same message explanation tags are required in a report message as they are in an exception message, although they do not show up in the message buffer.

The following example shows how we coded the message:

```
<msglist xreftext="csectname" rules="1">
<!-- ============================================================ -->
<!--  Message:    IRRH255R                                        -->
<!-- ============================================================ -->
<msg class="report">
<msgnum xreftext="255">IRRH255R</msgnum>
<msgtext>
<lines class="center">

<mv>report-name</mv>Report

</lines>
</msgtext>
<msgitem class="explanation"><p>
Header line for the RACF_SENSITIVE_RESOURCES check.
</p></msgitem>
<msgitem class="sysact"><p>
Processing continues.
</p></msgitem>
<msgitem class="oresp"><p>
None.
</p></msgitem>
<msgitem class="spresp"><p>
None.
</p></msgitem>
<msgitem class="probd"><p>
None.
</p></msgitem>
<msgitem class="source"><p>
None.
```

```
</p></msgitem>
<msgitem class="refdoc"><p>
None.
</p></msgitem>
<msgitem class="automation"><p>
None.
</p></msgitem>
<msgitem class="module"><p>
IRRHCR00
</p></msgitem>
<msgitem class="rcode"><p>
</p></msgitem>
<msgitem class="dcode"><p>
</p></msgitem>
</msg>
<!-- ============================================================ -->
<!--  Message:    IRRH256R                                        -->
<!-- ============================================================ -->
<msg class="report">
<msgnum xreftext="256">IRRH256R</msgnum>
<msgtext>
S Data Set Name                           Vol    UACC Warn ID*  User
</msgtext>
<msgitem class="explanation"><p>
Header line for the RACF_SENSITIVE_RESOURCES check
</p></msgitem>
<msgitem class="sysact"><p>
Processing continues.
</p></msgitem>
<msgitem class="oresp"><p>
None.
</p></msgitem>
<msgitem class="spresp"><p>
None.
</p></msgitem>
<msgitem class="probd"><p>
None.
</p></msgitem>
<msgitem class="source"><p>
None.
</p></msgitem>
<msgitem class="refdoc"><p>
None.
</p></msgitem>
<msgitem class="automation"><p>
None.
</p></msgitem>
<msgitem class="module"><p>
IRRHCR00
</p></msgitem>
<msgitem class="rcode"><p>
</p></msgitem>
<msgitem class="dcode"><p>
</p></msgitem>
</msg>
<!-- ============================================================ -->
<!--  Message:    IRRH257R                                        -->
<!-- ============================================================ -->
<msg class="report">
<msgnum xreftext="257">IRRH257R</msgnum>
```

```
<msgtext>
- ------------------------------------ ------ ---- ---- ---- ----
</msgtext>
<msgitem class="explanation"><p>
Data set header line for the RACF_SENSITIVE_RESOURCES check
</p></msgitem>
<msgitem class="sysact"><p>
Processing continues.
</p></msgitem>
<msgitem class="oresp"><p>
None.
</p></msgitem>
<msgitem class="spresp"><p>
None.
</p></msgitem>
<msgitem class="probd"><p>
None.
</p></msgitem>
<msgitem class="source"><p>
None.
</p></msgitem>
<msgitem class="refdoc"><p>
None.
</p></msgitem>
<msgitem class="automation"><p>
None.
</p></msgitem>
<msgitem class="module"><p>
IRRHCR00
</p></msgitem>
<msgitem class="rcode"><p>
</p></msgitem>
<msgitem class="dcode"><p>
</p></msgitem>
</msg>
<!-- ============================================================ -->
<!--  Message:    IRRH258R                                        -->
<!-- ============================================================ -->
<msg class="report">
<msgnum xreftext="258">IRRH258R</msgnum>
<msgtext>
<mv>status</mv>
<mv>data-set-name</mv>
<mv>volume</mv>
<mv>UACC-access</mv>
<mv>idSplat-access</mv>
<mv>warning</mv>
<mv>userId-access</mv>
</msgtext>
<msgitem class="explanation"><p>
Data line for data set analysis report.
</p></msgitem>
<msgitem class="sysact"><p>
Processing continues.
</p></msgitem>
<msgitem class="oresp"><p>
None.
</p></msgitem>
<msgitem class="spresp"><p>
None.
```

```
</p></msgitem>
<msgitem class="probd"><p>
None.
</p></msgitem>
<msgitem class="source"><p>
None.
</p></msgitem>
<msgitem class="refdoc"><p>
None.
</p></msgitem>
<msgitem class="automation"><p>
None.
</p></msgitem>
<msgitem class="module"><p>
IRRHCR00
</p></msgitem>
<msgitem class="rcode"><p>
</p></msgitem>
<msgitem class="dcode"><p>
</p></msgitem>
</msg>
     .
     .
     .
</msglist>
```

- **A report message in debug mode:** If you are running in debug mode and using SDSF, the report message above displays with the message number on every line:

```
HZS1098I   CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
HZS1090I   START TIME: 06/01/2005 13:29:16.860783
HZS1095I   CHECK DATE: 20040703  CHECK SEVERITY: HIGH
IRRH255R
IRRH255R                              APF Dataset Report
IRRH255R
IRRH256R   S Data Set Name                            Vol    UACC Warn ID*  User
IRRH257R   - ----------------------------------------- ------ ---- ---- ---- ----
IRRH258R   E SYS1.LINKLIB                             PETPB1 Updt No   ****
IRRH258R   E SYS1.SVCLIB                              PETPB1 Updt No   ****
```

  The message number is **not** displayed on every line if you are looking at the report message in the message buffer or with the print command. That is only a feature of SDSF.

Debug message example

The following example shows a debug message text as it would appear in SDSF output when the system is running in debug mode - a debug message **only** appears when you are running in debug mode. (See “Debugging checks” on page 132 for how to turn on debug mode.)

```
HZS1098I   CHECK(IBMSDUMP,SDUMP_AUTO_ALLOCATION)
HZS1090I   START TIME: 06/01/2005 13:46:28.025111
HZS1095I   CHECK DATE: 20050118  CHECK SEVERITY: MEDIUM

IEAH700I   IEAH700I Build level 2005.045 15:49:09.60 FC=       2 EC=       2
```

Note that if you look at the debug message in the message buffer or with the print command, you will **not** see the message number on every line.

The following example shows how we coded the debug message:

```
<msglist xreftext="csectname" rules="1">
<msg class=debug>
<msgnum xreftext=700>IEAH700I</msgnum>
<msgtext>
<mv>debug</mv>
</msgtext>
<msgitem class="explanation"><p>Checker debug information.</p>
</msgitem>
<msgitem class="sysact"><p>n/a</p>
</msgitem>
<msgitem class="oresp"><p>n/a</p>
</msgitem>
<msgitem class="spresp"><p>n/a</p>
</msgitem>
<msgitem class="probd"><p>n/a</p>
</msgitem>
<msgitem class="source"><p>SDUMP (SCDMP)</p>
</msgitem>
<msgitem class="refdoc"><p>n/a</p>
</msgitem>
<msgitem class="automation"><p>n/a</p>
</msgitem>
<msgitem class="module"><p>IEAVTSHG</p>
</msgitem>
<msgitem class="rcode"><p>n/a</p>
</msgitem>
<msgitem class="dcode"><p>n/a</p>
</msgitem>
</msg>
     .
     .
     .
</msglist>
```

Message list tagging example

The following example shows how you would code the entire message list for your messages, including the copyright and <msglist> tags:

```
<lines props="copyright">
*    THE SOURCE CODE FOR THIS PROGRAM IS NOT PUBLISHED OR OTHERWISE
*    DIVESTED OF ITS TRADE SECRETS, IRRESPECTIVE OF WHAT HAS BEEN
*    DEPOSITED WITH THE U.S. COPYRIGHT OFFICE.
*    OCO SOURCE MATERIALS
*    5650-ZOS (C) COPYRIGHT IBM CORP. 2005
</lines>
<msglist xreftext="hzshmtbl">
<!-- ============================================================ -->
<!--  Message:    IRRH201I                                        -->
<!-- ============================================================ -->
<msg class="information">
<msgnum xreftext="201">IRRH201I</msgnum>
<msgtext>The &hzsckname;  check cannot be executed in a
GRS=NONE environment.
</msgtext>
<msgitem class="explanation"><p>
The RACF check &hzsckname; is not applicable to a
GRS=NONE environment.
</p></msgitem>
<msgitem class="sysact"><p>
The check stops processing. There is no effect on the system.
</p></msgitem>
<msgitem class="oresp"><p>
Report this problem to the system programmer.
</p></msgitem>
<msgitem class="spresp"><p>
Disable the  &hzsckname; RACF check.
</p></msgitem>
```

```
<msgitem class="probd"><p>
</p></msgitem>
<msgitem class="source"><p>
RACF System Programmer's Guide
</p></msgitem>
<msgitem class="refdoc"><p>
<lines>
RACF System Programmer's Guide
MVS Planning: Global Resource Serialization
</lines>
</p></msgitem>
<msgitem class="automation"><p>
None.
</p></msgitem>
<msgitem class="module"><p>
IRRHCR00
</p></msgitem>
<msgitem class="rcode"><p>
</p></msgitem>
<msgitem class="dcode"><p>
</p></msgitem>
</msg>

      .
      .
      .
</msglist>
```

Syntax of message input

You code the message input with tags. The following topics describe the syntax for coding an IBM Health Checker for z/OS message in the message table:

- "Message input tags"
- "Special formatting tags for the message table" on page 223
- "How messages are formatted in the message buffer" on page 225
- "Using symbols in the message table" on page 227

Restrictions:

The following characters are tag characters, and you can only use them in tags in your message table. Note, however, that IBM Health Checker for z/OS does support symbols in the message table.

- Do not use < (less-than) and > (greater-than) - You can use < and > as substitutes.
- Do not use & (ampersand) - You can use & as a substitute.

Message input tags

Copyright information

<lines id="*checkownername***" props="copyright" >** * *copyright information* * **</lines>**

The message table for every check **must contain** a copyright statement. For example, a copyright statement might look as follows:

```
<lines id="IBMRACF" props="copyright">
*    THE SOURCE CODE FOR THIS PROGRAM IS NOT PUBLISHED OR OTHERWISE
*    DIVESTED OF ITS TRADE SECRETS, IRRESPECTIVE OF WHAT HAS BEEN
*    DEPOSITED WITH THE U.S. COPYRIGHT OFFICE.
*    OCO SOURCE MATERIALS
*    5650-ZOS (C) COPYRIGHT IBM CORP. 2005
</lines>
```

id="*checkownername***"**
Specify the check owner, such as IBMRACF, for the messages in this message list.

props="copyright"
Specify props="copyright" to indicate that this is your copyright statement.

Message list tag - <msglist>

<msglist xreftext="*csectname* **rules=***ruleslevel"***></msglist>**:

The message list tag. You can only have **one message list per message table**. If you include any data after the end message list tag, </msglist>, the message generation program will issue error message HZSM0009 and issue a return code of 8. See "Generating the compilable assembler CSECT for the message table" on page 231.

csectname
Specify the name of the generated message table CSECT. *csectname* must be 1-8 alphanumeric characters. xreftext="*csectname*" is required.

ruleslevel
Use the rules attribute to indicate whether or not your check message input table uses message elements that apply only to z/OS V1R8 and above systems. If the message table does not conform to the rules level selected, the HZSMSGEN exec will not be able to generate the message table.

rules="3"
Indicates that the check can run on systems at the z/OS V1R10 level or above and use R10 and above functions, including all the rules="2" and rules="1" functions. The z/OS V1R10 enhancement includes:

- When a message class of report is used, and the message output requires more than one line, leading blanks are suppressed at the beginning of the new line.

If you specify rules="3" on a message list for a check that runs on a system below the z/OS V1R10 level, the check abends when the system tries to add the check to the system.

rules="2"
Indicates that the check can run on systems at the z/OS V1R8 level or above and use R8 and above functions:

- You can use maxlen and fieldsize attributes for variables - see "Variables for message text" in "Message text (<msgtext>) and message variable (<mv>) tags" on page 218.
- In rules="2" message list, you cannot enter leading blank characters after the <p> and <msgtext> tags. If you need a blank in a rules="2" messagelist, use symbol &rbl;, which inserts a required blank into the formatted output. This is also useful for keeping words together on one line. For example, to keep the words `System A` on one line and together, code the following:

```
System&rbl;A
```

If you specify rules="2" on a message list for a check that runs on a system below the z/OS V1R8 level, the check abends when the system tries to add the check to the system.

In order to convert a message list from rules="1" to rules="2", you must:

- Specify rules="2" on the message list tag
- Use a required blank symbol, &rbl; if you require a leading blank after a <p> tag or <msgtext> tag.

rules="1"
Indicates that the check can run on any system where IBM Health Checker for z/OS is available, depending on what releases the individual check supports - see Chapter 13, “IBM Health Checker for z/OS checks,” on page 381. At rules level 1, a check message table cannot use z/OS V1R8 enhancements such as required blank symbols or maxlen and fieldsize attributes for variables. For a rules="1" message list, the system will ignore leading blank characters after the paragraph tag, <p>, and message text tag, <msgtext>.

Message instance tag - <msg>

<msg class="*msgtype***"></msg>**
The <msg> element defines a message in a message list. The class="*msgtype*" attribute, which describes the type of message, is required on the <msg> tag. The class describes z/OS Health Checker message behavior. You can have the following values for class="*msgtype*":

- **Exception:** Messages notifying the installation that action is required because a check routine found an exception to a suggested setting. The message text, intended for the operator, is issued in a WTO. The message text and full explanation are issued to the message buffer, mainly for the system programmer.
- **Information:** Messages conveying general non-exception information, for example the completion of the check without exceptions, or as the first line of a report. Only the message text is issued to the message buffer.
- **Report:** Single lines of report data issued to the message buffer without a message number (except in debug mode). Only the message text is issued to the message buffer. The report message type gives you the most control over the formatted output. A single report line should be 72 characters of formatted output or less. Because report messages do not display with a message number, a report message with a line of report data should be preceded by an information message containing the report title.
- **Debug:** Messages issued to the message buffer when the check is placed in debug mode to aid in testing and diagnosis. Only the message text is issued to the message buffer.

The following table summarizes information for the different message types:

Table 20. A summary of message types for IBM Health Checker for z/OS

Message type <msg class=*msgtype*>	**Message number suffix**	**Where is message text issued?**
Exception <msg class=Exception>	E	• The message buffer, including the message text and all details. • WTO message text **only** to console, operlog, or syslog

Table 20. A summary of message types for IBM Health Checker for z/OS (continued)

Message type <msg class=*msgtype*>	Message number suffix	Where is message text issued?
Information <msg class=Information>	I	Message buffer
Report <msg class=report>	N/A - Message number is **not displayed** unless you are running in debug mode and use SDSF to display the message.	Message buffer
Debug <msg class=debug>	N/A - Message number is **not displayed** unless you are running in debug mode and use SDSF to display the message.	Message buffer or system hardcopy log

Each <msg> element **must** contain the following elements:

- <msgnum> - See "Message number tag - <msgnum>."
- <msgtext> - See "Message text (<msgtext>) and message variable (<mv>) tags" on page 218.
- <msgitem> - See "Message item tag - <msgitem>" on page 222.

Restriction: You must code the following tags in consecutive lines without comments or blank lines between them:

```
<msg  class="msgtype"></msg>
<msgnum  xreftext="nnnn">ccccHmmmms</msgnum>
<msgtext>....
```

Message number tag - <msgnum>

<msgnum xreftext="*msgnumber*">*ccccHmmmms*</msgnum>**

The message is identified in two ways on the <msgnum> tag - both inputs are **required** for all messages:

xreftext="*messagenumber***"**

messagenumber is the decimal value used in the HZSFMSG macro to uniquely identify the message. *messagenumber* is a decimal value between 1 and 2147483647. The *messagenumber* you define will be reflected in the MGB_MessageNumber field in the HZSMGB data area.

*cccc***H***mmmms*

The text value that appears as the message identifier. In the message identifier:

*cccc***H** *cccc* is the component identifier, such as ISG for global resource serialization. The required H represents IBM Health Checker for z/OS.

mmmm

The message number. For example, in message identifier ISGH101E, 101 is the message number.

s The severity code for the message. *s* is one of the following:

- For an **exception** message, use E.
- For **information** and **debug** messages, use I.
- For **report** messages, you can use any suffix you like, or no suffix at all - users will not see these message numbers unless they're running IBM Health Checker for z/OS in debug mode and viewing the message buffer with SDSF. Because report messages do not display with a message number, a report message with a line of report data should be preceded by an information message with the report title and the explanation for the report.

Message text (<msgtext>) and message variable (<mv>) tags

<msgtext></msgtext>

Required element that contains the data issued to a message buffer when IBM Health Checker for z/OS issues messages for a check.

For an exception message, the system uses data in the <msgtext> tags to create a WTO. In many installations, the exception message WTOs (which are issued to the operator's console) require national language support (NLS) translation using MVS message service (MMS). If you are going to translate your messages, each message text line must be 71 characters or less. See "Support for translating messages to other languages" on page 232 for guidelines on how to code the message text for translating messages.

You can have message variables in a message text - see "Message text (<msgtext>) and message variable (<mv>) tags."

You **cannot** use paragraph ( ) tags within the <msgtext> . If you need to start a new line, <lines></lines>:

```
<msgtext>
I need a new line, but I can't use a paragraph tag.
<lines></lines>
But I can get a new line with the lines tag.
</msgtext>
```

For information on how message text (<msgtext>) is formatted in the message buffer, see "How messages are formatted in the message buffer" on page 225.

<mv class="*variable_class*" xreftext="maxlen(*nnn* | fieldsize(*nnn*")></mv>

Specify <mv></mv> to define the variables in your message text (<msgtext></msgtext>). You define each variable with a class="*variable _class*", to specify the type of variable and xreftext="maxlen(*nnn)* | fieldsize(*nnn*" to define the length of a variable after it is formatted.

class="*variable_class*"

Specify <mv class="*variable_class*"> to define different types of variables. A message variable allows the check routine to issue the HZSFMSG macro with dynamic variables in the message text. You can have up to 20 message variables in a message, each used one time only. When you develop a check, you'll provide the data for the message variables in the HZSMGB data area in the MGB_MsgInsert field. See "How messages and message variables are issued at check runtime" on page 200.

The default for an <mv> variable (if you do not specify a class) is text, which indicates that the variable is EBCDIC text with a maximum length of 256 bytes.

Use a meaningful variable name. IBM Health Checker for z/OS processes message variables positionally; it does not parse the content. The following example shows a message text message with two variables:

```
<msgtext>LNKLST
<mv class=compress xreftext=maxlen(16)>lnklst name</mv>
data set name :
<mv class=compress xreftext=maxlen(44)>dsname</mv><lines></lines>
has more extents than when it was
activated.
</msgtext>
```

The output of the message markup showing the dynamically resolved variables might look as follows:

```
LNKLST LNKLST00 data set name: DATASET1.DATA.ABC has more extents than
when it was activated.
```

Note that although the variables are coded on separate lines, they are all on the same line in the output. The system inserts a blank between each variable because of the end of the line.

You can specify the following different types of variables:

<mv class="compress">
Specifies that data in the MGB_MsgInsertField of the HZSMGB data area is text. The system removes leading and trailing blanks within the variable in the message output. class="compress" is the default behavior for exception, information, and debug messages.

<mv class="nocompress>
Specifies that data in the MGB_MsgInsertField is text. The system will not remove leading and trailing blanks or suppress blanks within the variable in the message output - the length of the variable will be same as the length provided in the MGB_MsgInsertField for the variable.

Class="nocompress" is the default for report messages because it gives you more control over the formatted output. The system does not remove blanks in nocompress variables..

You cannot specify xreftext="fieldsize(*nnn*)" for a nocompress variable.

<mv class="condcompress>
Specifies that data in the MGB_MsgInsertField field for the variable is text. The system leaves or removes leading and trailing blanks, depending on the message type. For report messages, the system does not remove blanks in the variable. For exception, information, and debug messages, the system does remove blanks within the variable. <mv class="condcompress> is the default.

<mv class="decimal">
Specifies that the binary input described in the MGB_MsgInsert field for the variable be converted to

decimal. Leading zeros are suppressed, and the field size is set to the first significant digit. The largest generated output length for decimal variable values up to 2147483647 (X'7FFFFFF') is 10 bytes. Values greater than 2147483647 are expressed with multiplier notation and can be a maximum of six characters. The following table shows the value of multipliers:

Multiplier	Abbreviation	Value
Kilo	K	1,024
Mega	M	1,048,576
Giga	G	1,073,741,824
Tera	T	1,099,511,627,776
Peta	P	1,125,899,906,842,624

<mv class="hex">
Specifies that the binary input described in the MSB_MsgInsert field for the variable (up to 100 characters) be translated to a hexadecimal value expressed in EBCDIC characters. Hexadecimal output is formatted with an underscore after every eighth character and the underscores must be taken into account when calculating the length of the output message text.

<mv class="gmttime">
Specifies that the value from the MGB_MsgInsert field is not to be adjusted for local time. The input data for this variable is an eight character field containing a 64-bit time of day value in the format MM.DD.YY HH:MM:SS.*ffffff*.

<mv class="LocalTime">
Specifies that the value from the MGB_MsgInsert field is to be adjusted to local time. The input data for this variable is an eight character field containing a 64-bit time of day value in the format MM.DD.YY HH:MM:SS.*ffffff*.

Maximum length values for variables are as follows:

Table 21. Variable input and output lengths and alignment:

Variable type	Input length	Output length
Class="compress", "nocompress", or "condcompress"	1-256	256
Class="hex"	1-100	256
Class="decmial"	1-8	10
Class="gmttime", "localtime"	8	26

Note that if a variable has a zero input length at the time when the message is issued, the field does not show up in the output because all the nulls and blank are eliminated from the output.

xreftext=" maxlen(*nnn*) | fieldsize(*nnn*)"
Specify <mv class="*variable_class*" xreftext="maxlen(*nnn* | fieldsize(*nnn*" to

define the length a variable after it is formatted. You can only specify maxlen and fieldsize on a rules="2" message list.

maxlen(*nnn*)

Maxlen allows you to specify the maximum length of formatted output a variable should produce. Maxlen applies only to systems at z/OS V1R8 or higher and the message list must specify rules="2". Maxlen is particularly useful for calculating variable size for NLS message translation, which requires message line text length of 71 characters or less. See “Support for translating messages to other languages” on page 232.

If variable resolution produces output greater length than the value of maxlen, the HZSFMSG invocation abends with abend code X'290', reason code X'4016'.

The following table shows which variables allow maxlen:

Table 22. Which variables allow maxlen?

Variable type	Maxlen allowed?	Maxlen value allowed
Class="compress", "condcompress"	Yes	256 or less
Class="nocompress"	Yes	256 or less
Class="hex"	Yes	224 or less, because the output is limited to 100 bytes.
Class="decmial"	Yes	10 or less
Class="gmttime", "localtime"	No	–

fieldsize(*nnn*)

Fieldsize allows you to specify the exact length that the field should always be in the formatted output. The output is expanded to the specified length and padded with blanks. This is useful for creating columns in report messages. When you specify fieldsize, variables are aligned as follows:

- Decimal and hex variables are right aligned
- Hexadecimal and character variables that specify fieldsize are left aligned.
- Compress and condcompress variables are left aligned
- gmltime and localtime variables are truncated on the right

The following example shows a report message that uses fieldsize for variables:

```
<msg class=report>
<msgnum xreftext=0001>ReportL01</msgnum>
<msgtext><mv xreftext="fieldsize(6)">Volser</mv>
<mv xreftext="fieldsize(44)">data set name</mv>
<mv class="decimal" xreftext="fieldsize(6)">data set extents</mv>
</msgtext>
```

If variable resolution produces output greater length than the value of fieldsize, the check abends with abend code X'290', reason code X'4116'.

Message item tag - <msgitem>

<msgitem class="*itemclass*"></msgitem>

The <msgitem class="*itemclass*"> tag contains the message explanation information that is usually included in a message reference document:

- For **class="exception"** messages, the information specified in the <msgitem class="*itemclass*"> tags is also included in the message buffer, where it is available for users to read and automate from. The first line of the message explanation described with <msgitem> tags begins in position 3. Lines following the first line of the explanation begin in position 5. When a <msgitem> section ends, the system generates new paragraph to put a blank line between each <msgitem>.
- For **class="information", "report", and "debug"** messages, information specified in the <msgitem> tags will only be included in the documentation for the message in message book for the check component. The <msgitem> information is not included in the output in the message buffer or elsewhere for non-exception messages.

You can optionally enclose the *itemclass* in quotes, either double or single. Use <mv></mv> to list and explain all variables that appear in the message. Use the following formatting elements to control the presentation of text

- (paragraph) - text in a message item must be within paragraph tags ( ).
- <lines> (lines) - allows you to control lines of text. Use <lines></lines> to generate a blank line.

See “Special formatting tags for the message table” on page 223.

You will have multiple <msgitem class="*itemclass*"> tags for a message (see “Examples of message input” on page 205 for an example). All of the classes are required. If you do not have specific information for a class, you can often use 'n/a'.

Message Table

Table 23. Description of <msgitem> classes required for all message explanations

Class	Description
"explanation"	Explains the message. Can include message variables, paragraphs and other formatted text - see “Special formatting tags for the message table” on page 223. For an exception message, the explanation should describe the exception condition found and its impact to the system. For the informational report title message, the explanation must include the meaning of the variables used in the message text that are not self-explanatory within the explanation. For example, if you use <mv>*widget*</mv> within the message text, you must then explain what the variable in the explanation, as follows: *widget* An important device you need to buy for your computer. *widget* will be one of the following: **widgeta** Type a widget **widgetb** Type b widget
"sysact"	The system action describes what the system, in particular the component that owns the check, is doing as a result of the condition that caused the message to be issued. The system action must be specific - you cannot enter a system action of 'n/a' or 'None'. The system always does something, even if it just continues processing.

Table 23. Description of <msgitem> classes required for all message explanations (continued)

Class	Description
"oresp"	Operator response describes the actions an operator should take in response to the message. • For exception messages, this should direct the operator to the right person to evaluate the exception. for example, "Contact the system programmer:" • For other messages, which do not appear on the operators console, 'n/a' is the correct operator's response. The operator response can also be 'n/a'.
"spresp"	System programmer response describes the actions, if any, the system programmer should take to isolate and correct an error, including diagnostic steps and reporting the problem to the IBM support center. Include sample syntax and references for changing system parameters or issuing commands. Include a reference to the problem determination category if you're using the probd class for additional information for the system programmer. "Spresp" can also be 'n/a'.
"probd"	Problem determination - communicates additional information or actions that a system programmer, system administrator, security administrator, or database administrator may need to know to further diagnose a problem discussed in the system programmer response, including trace or dump information. Provide cross references (including links to other books) to procedures - different dumps use different allocations, procedures, and resources. Probd can also be 'None'.
"source"	The name of the component, subsystem, or product issuing the message.
"automation"	Automation - Use this section to discuss automation concerns related to the check results. Specify 'n/a' if you have no automation information for a message.
"refdoc"	Use the reference documentation class to reference books that provide additional detail regarding suggested settings or interpreting results. Include the book title, chapter heading, and section heading. Specify 'n/a' if you have no reference information.
"module"	Module identifies the name of the detecting module, the component name, or N/A. This class is not included in the message buffer for an exception message.
"rcode"	Routing code. Specify N/A for this field, because Health Checker for z/OS does not use a routing code, so this value is always zero unless the installation overrides the value in the HZSPRMxx parmlib member values for the check. Rcode is not included in the message buffer for an exception message.
"dcode"	Descriptor code. For an exception message, document the default descriptor code based on the severity of the check: • **High** severity checks use a descriptor code of 11 • **Medium** severity checks use a descriptor code of 3. • **Low** severity checks use a descriptor code of 12. See “Defining your own symbols for check messages” on page 229.The installation can override the severity and descriptor code in the HZSPRMxx parmlib member. Dcode is not included in the message buffer for an exception message. For a non-exception message specify N/A for this field.

Special formatting tags for the message table

<mv ></mv>

For use in the message explanation, <msgitem> tags, specifies that the text within the <mv></mv> tags are variables. The text within the tags will generally format in italics. See “Message text (<msgtext>) and message variable (<mv>) tags” on page 218 for complete information about variables.

<!-- comment --> tags

The data placed between these tags is treated as a comment. Comment are not supported within a <msgtext> or <msgitem> section. Comments put in these sections will result in a syntax error and the HZSMSGEN REXX exec cannot generate the messages into a compilable assembler CSECT. You cannot place comments or blank lines inside the body of individual

messages, between the <msg> and </msg> tags. This will cause unpredictable results. You should only place comments and blank lines:

- Before the copyright statement for the message table (<lines id="*ownername*" props="copyright" > * *copyright information* * </lines>).
- Between the copyright statement and the message list tag, <msglist>
- Between the <msglist> tag and the message tag, <msg>
- Between individual messages, which would be between the message end tag, </msg>, and the next message start tag, <msg>.

Comments must go on a separate line with no other data.

See "Defining your own symbols for check messages" on page 229 for putting comments in an entity declaration.

<lines></lines>

The <lines> tag lets you control lines of text by keeping short lines of text from flowing together or by generating blank lines. You can use the <lines> tags to format text in the message text (<msgtext>) or explanation (<msgitem> tags) for any type of message. The <lines class="center"> tag lets you both control and center lines of text. Use <lines tags with the following considerations:

- For exception messages, use <lines></lines> tags to define a new line **before** you reach the WTO limit of 71 characters. Make sure you include the message number in your count.
- The <lines> or <lines class="center"> beginning tag generates a new line. Use <lines></lines> to create a blank line for messages in the message buffers. Blank lines are suppressed for WTOs, so in the WTO message text for an exception message <lines></lines> will just start a new line.
- If you specify too long a line of text within the <lines> tag to fit on the line, the data wraps to a new line.
- The end of a line is broken on a word boundary and causes the next word to begin a new line.
- You can put variables (<mv> tag) and symbols within <lines tags.
- You cannot use tags within the <lines markup.

<lines> example 1: The following example show a valid use of <lines> tags to keep a group of short lines from flowing together:

```
<lines>
Short lines of data
that format exactly as I type them
in the generated output.
</lines>
```

<lines> example 2: You cannot use tags within the message text <msgtext> tags. If you need to start a new line, use <lines></lines> instead. For example, you might want to break a line in an exception message text before you reach the WTO limit of 71 characters.

```
<msgtext>
I need a new line, but I can't use a paragraph tag.
<lines></lines>
But I can get a new line with the lines tag.
</msgtext>
```

For a WTO message text, <lines></lines> tagging starts a new line:

```
I need a new line, but I can't use a paragraph tag.
But I can get a new line with the lines tag.
```

In the message buffer, <lines></lines> gives you a blank line:

```
I need a new line, but I can't use a paragraph tag.

But I can get a new line with the lines tag.
```

<lines class="center"> example: Use <lines class="center"> to center your lines of text:

```
<lines class="center">
Short lines of data
that format exactly as I type them
in the generated output
except centered
</lines>
```

You will get the following output:

```
       Short lines of data
that format exactly as I type them
     in the generated output
         except centered
```

** **

A paragraph contains text in paragraph form. Use paragraph tags to format paragraph text as follows:

- Paragraph tags are required to enclose the text in all <msgitem> tags, for any type of message.
- You cannot use paragraph tags in message text <msgtext>. Instead, use <lines></lines> to create a blank line.
- The beginning tag starts a new line and data begins at the start of the next line.
- If your text in a paragraph hits the end of the line boundary, the line splits on a word boundary. Leading and trailing blanks may be lost when the line is split. See "How messages are formatted in the message buffer."

Example: The following example shows valid use of paragraph tags:

```
This is a line of text.
<p>Although the text flow of paragraph is the default behavior,
more rigid rules are observed.
</p>
<p>
Blank
lines are suppressed.
</p>
```

This example will format as follows:

```
 This is a line of text.

Although the text flow of paragraph is the default behavior, more rigid
rules are observed.

Blank lines are suppressed.
```

How messages are formatted in the message buffer

For exception messages, both text (<msgtext>) and explanation (<msgitem class="*itemclass*"> except rcode and dcode) are issued to the message buffer. For information, report, and debug messages, only the text (<msgtext>) is issued to the message buffer or log.

Default formatting for messages in the message buffer is as follows:

- Message processing for all messages and all the parts of a message use paragraph flow unless <lines></lines> tags are used to break up lines.
- When a message is reformatted to fit in the message buffer, the system splits the line on a word boundary. Blanks may be lost when the line is split. The system suppresses blank lines.
- The system strips leading and trailing blanks from variables, except variables in report messages or nocompress variables (<msgitem class="nocompress">.
- A character string with a length greater than the output line length will continue on the following line without blanks added. This is referred to as wrap mode.

The table below shows how the different types of messages are formatted in the message buffer:

Table 24. How messages are formatted in the message buffer

Message type	Formatting in the message buffer

Exception	A complete message, including both the message text **and** explanation are issued to the message buffer. The message text is limited to 14 lines, 70 characters long (4 for indentation). In the check details, you can have lines 71 characters long. Check exception messages are imbedded in a IBM Health Checker for z/OS HZS prefix message with a prefix of HZS and format as follows:
	• The first line of the exception message contains the HZS message identifiers, either HZS003A, HZS002E, or HZS001I, depending on the severity of the message as well as the check owner and check name. For example, the following output would be issued to the operator console:
	`HZS002E (IBMCSV,LNKLST_SPACE)`
	`CSVH951E  LNKLST CZ INCLUDES DATA SETS WITH SECONDARY SPACE`
	`DEFINED.`
	• The first line of the message explanation in the message buffer begins in position 3.
	• The second line of the message begins with the message identifier assigned to the message in the <msgnum> tag.
	• The second and following lines of the message begin in position 5 and are 66 characters long.
	• When the <msgtext> section ends, the system generates a new line.

Information	The message text (<msgtext>) specified is formatted in paragraph format in the message buffer. • The system splits the line on a word boundary and begins the new line with a non-blank character. The system suppresses blank lines. • When data exceeds the output line length, it will wrap to the next line. • A line ends on a word boundary, and new lines begin in the 5th position on the following line.
Report	The message text (<msgtext>) is a single line of data issued to the message buffer. • Each line of text begins in position 1. • When data exceeds the output line length, it will wrap to the next line. • The system will not suppress leading and trailing blanks in the message text when you add dynamic variables using the <mv> element. To compress leading and trailing blanks in a variable in a report message, use a <mv class="compress> tag.
Debug	The message text (<msgtext>) is issued to the message buffer or system hardcopy log when you place the check in debug mode. • The first line of text in a debug message begins in position 1, unless you format the lines differently, using <lines>, for example. • Lines end on a word boundary and new lines begin in the 5th position on the following line. • When data exceeds the output line length, it will wrap to the next line. • The system suppresses leading and trailing blanks in the message text of information messages unless formatting tags override the default text flow. • You can add dynamic variables using the <mv> element.

Extra fields issued to the message buffer for exception messages

For exception messages issued to the message buffer, IBM Health Checker for z/OS issues additional information automatically, including:

- **Owner IBM*comp* reason**: This field displays the reason for running the check. The reason displayed is specified by the check developer in the HZSADDCHECK exit routine r. For example, for check GRS_MODE, the reason defined in the HZSADDCHECK exit routine r is as follows:

  ```
  GRS should run in STAR mode to improve performance
  ```

  See Chapter 9, "Writing an HZSADDCHECK exit routine," on page 191.
- **Installation reason**: This field is displayed if the installation overwrites the HZSADDCHECK exit routine r reason with a new reason value.
- **Check Parameters**: This field displays the parameters (defined in the HZSADDCHECK exit routine r) that are passed to the check routine when it runs.

Using symbols in the message table

You can specify symbols in your message table that resolve to meaningful values. There are several types of symbols that you can use:

- Pre-defined system symbols set by IBM Health Checker for z/OS for use in an exception message and that resolve at check runtime. (A very few resolve when you generate your message table CSECT.) See "Using pre-defined system symbols."
- Symbols defined in the source message table or setup file that resolve when you generate the CSECT for the message table. See "Defining your own symbols for check messages" on page 229.

Using pre-defined system symbols

You can use the following predefined system symbols in an exception message. These are set by IBM Health Checker for z/OS. For example, the following message markup uses symbols (delimited by an ampersand and a semi-colon) for the check and system names:

```
<msgtext>&hzsckname; Health Checker Report for z/OS
on system &hzssysname;
</msgtext>
```

IBM Health Checker for z/OS sets these symbols at registration and run time. That means you do not have to define these symbols in a setup data set or in your message table in order to use them. The system resolves the following pre-defined symbols that resolve when the check runs:

Table 25. A summary of pre-defined symbols that resolve when the check runs

Predefined symbol	Maximum number of characters	Symbol resolves to
&hzs;	38	IBM Health Checker for z/OS
&hzsproc;	8	The name of the start up procedure for IBM Health Checker for z/OS
&hzssysname;	8	System name
&hzssysplex;	8	Sysplex name
&hzsreason;	256	User or component reason from the HZSADDCHECK exit routine r.
&hzsexitrtn;	8	The name of the HZSADDCHECK exit routine r.
&hzsparmsource ;	16	Resolves to 'installation' or 'owner' to indicate whether the default PARMS from the HZSADDCHECK exit routine r are in effect, or user overrides are in effect.

Table 25. A summary of pre-defined symbols that resolve when the check runs (continued)

Predefined symbol	Maximum number of characters	Symbol resolves to
&hzssev;	6	Resolves to the severity set defined at runtime from either the HZSPRMxx parmlib member or the HZSADDCHECK exit routine r (HIGH | MEDIUM | LOW)
&hzsparms;	126	Active check parameters
&hzsckname;	32	The check name, as defined in the HZSADDCHECK exit routine r
&hzsowner;	16	The check owner, which is the component or subsystem as defined in the HZSADDCHECK exit routine r. For example, &owner; might resolve to OEM, IBMGRS, IBMRSM, IBMXCF, or IBMUSS.
&hzsdate;	10	The current system date in the form: `dd mmm yyyy`
&hzsgmttime;	16	The current system GMT time is displayed in the form: `mm/dd/yyyy hh:mm:ss.tttttt`
&hzslocaltime;	16	The current system time is adjusted to local time and displayed in the form : `mm/dd/yyyy hh:mm:ss.tttttt`

The following table shows other pre-defined symbols that resolve when you generate the CSECT for the message table:

Table 26. A summary of pre-defined symbols that resolve when you generate the CSECT for the message table

Predefined symbol	Maximum number of characters	Symbol resolves to
&rbl;		The ever useful required blank character for use in a rules="2" message list. In rules="2" message list, you cannot enter leading blank characters after the <p> and <msgtext> tags, so you must use symbol &rbl;, which inserts a required blank into the formatted output, and keeps the words on the same line. A required blank will not be removed during formatting. This is also useful for keeping words together on one line. For example, to keep the words `System A` on one line and together, code the following: `System&rbl;A` You can also specify a required blank in a message variable using the character X'44'. The system resolves this hex character as a blank when the output data is formatted.
>		Greater than symbol, >
<		Less than symbol, <
&		Ampersand symbol, &
&hzsnl;		A tag to begin a new line
&hzsbl;		A tag to insert a blank line.

If you want to use other symbols besides the predefined system symbols in your message input, you must define them in the source message table (before the <msglist> tag) or in the message setup data set, see "Defining your own symbols for check messages" on page 229.

Defining your own symbols for check messages

Besides using the predefined system symbols, you can also define symbols specific to the check messages in the message table or setup file. We'll call these local symbols. Keep the following in mind as you plan whether to define local symbols and which ones to define:

- You should only create a local symbol for a known constant that you use multiple times in the message table.
- National language support (NLS) variables are not created for check specific symbols, and they do not require special support at execution time.
- Local symbols can use symbols within symbols. In other words, the symbol substitution text can include other local or system symbols.

Like system symbols, you specify the entity for your local symbol as a name delimited by an ampersand (&) and a semi-colon. For example, you could specify and use local symbols as follows:

- Create your own symbols for any text you use multiple times in the message table. When you generate the message table, the symbols will resolve to your text value. For example:

  ```
  If I insert an entity, or symbol, &newsym;.
  ```

  This would resolve to:

  ```
  If I insert an entity, or symbol, the symbol resolves to this exciting text.
  ```

- Define your own symbols to make it easier to put accurate book titles in the **required** <msgitem class="*refdoc*"> tag for check messages. For example:

  ```
  For more information about the recommended settings, see &ieaa100t;.
  ```

  This resolves to:

  ```
  For more information about the recommended settings, see
  z/OS MVS Auth Assm Services Reference ALE-DYN.
  ```

You can define symbols for your check using <!ENTITY> tags in either:

- The **source message table**, before the <msglist> tag. Here's an example of defining symbols in the message table itself:

  ```
  <!ENTITY PROD1 "Product ABC">
  <!ENTITY PROD2 "Product DEF">
  <!ENTITY NA "N/A">
  <msglist xreftext="PRODABC Rules=2">
                      .
                      .
                      .
  ```

- A **setup data set**, which is a separate data set containing the symbol definitions for your check. This is a handy way to make symbols available for multiple checks. Here's an example of a setup data set with both a copyright statement and some symbols we find very useful for multiple users and multiple checks:

```
<!ENTITY cunu100t "z/OS Support for Unicode: Using Conversion Service">
<!ENTITY ieaa100t "z/OS MVS Auth Assm Services Reference ALE-DYN">
<!ENTITY ieaa200t "z/OS MVS Auth Assm Services Reference ENF-IXG">
<!ENTITY ieaa300t "z/OS MVS Auth Assm Services Reference LLA-SDU">
<!ENTITY ieae200t "z/OS MVS Initialization and Tuning Reference">
                   .
                   .
                   .
<!ENTITY act " The system continues processing.">
<!ENTITY bugmsg " This message only appears when you are running in debug mode.">
<!ENTITY repsysp "Report this problem to the system programmer.">
<!ENTITY lvl2    "Search problem reporting data bases for a fix for the problem.">
<!ENTITY diagdoc "Provide the messages, the logrec data set record, the SYSLOG output, and dump if one was taken."
-- The following symbols are defined for routing codes. -- >
<!ENTITY hisevdc "11 is the default set by this check.">
<!ENTITY medsevdc "3 is the default set by this check.">
<!ENTITY losevdc "12 is the default set by this check.">
<!ENTITY success "This check ran successfully and found no exceptions.">
```

Figure 15. Example of a setup data set that defines symbols used in the message table

Using either of these methods, the symbols are resolved in the CSECT when you generate the message table.

Syntax for defining your symbols - the <!ENTITY> tag: The following shows the syntax of the <!ENTITY> tag you use to define your own symbols in the message table or setup data set:

<!ENTITY *entity-name* **"***replacement text***" -- comment -- >**

An ENTITY identifies an entity declaration, which just means it's how you define a symbol. An entity statement breaks down as follows:

entity-name

Specifies the name you select for your symbol. When you specify the entity name in your message table, it will resolve to the replacement text when you generate the message table.

"*replacement text***"**

A single string specifying the text that you want your entity or symbol name to resolve to.

Local symbols can use symbols within the replacement text. For example, you could define a symbol as follows:

```
<!ENTITY good2 "&amp;good1; is a good symbol, these are two good symbols">
```

Then in the message table, you code the following sentence:

```
If &good2; we can use in a message.
```

Will resolve into a complete sentence when the check runs:

```
If &good1; is a good symbol, these are two good symbols we can use in a message.
```

Neat huh? Don't get yourself tied in knots though!

-- comment -- >

Specifies a comment within an entity declaration. You can insert a comment anywhere in an entity declaration between the < > delimiters. Identify the start and end of the comment with two hyphens (--). The following example shows a comment in an entity declaration:

```
<!ENTITY newsym  "the symbol resolves to this text" -- this is a comment -->
```

Generating the compilable assembler CSECT for the message table

The message table is loaded in Health Check private, it should be a single csect load module.

You can generate the messages from the message table into a compilable assembler CSECT using the message generation exec HZSMSGEN. The input and output DD names for HZSMSGGEN can be allocated to a data set, PDS member or z/OS UNIX file.

The JCL for HZSMSGEN is contained in member HZSMSGNJ of SYS1.SAMPLIB.

HZSMSGEN can also generate the national language support (NLS) message skeletons for message translation of check message text, if desired. See "Support for translating messages to other languages" on page 232.

To use the message generation JCL to generate check messages into a CSECT, do the following:

1. Get the HZSMSGNJ message generation JCL from SYS1.SAMPLIB. Also in SYS1.SAMPLIB, you will find the following files referenced in the HZSMSGNJ JCL:
   - Member HZSSMSGT containing a sample message table.
   - Member HZSSSYMD containing a sample local symbol.
2. Customize your copy of HZSMSGNJ as indicated in the prolog.

   Note that specifying NLSCHECK(Y) does not specify that message skeletons be generated! NLSCHECK(Y) specifies that you want HZSMSGEN to enforce the message translation guidelines that will make it possible to generate message skeletons. (See "Support for translating messages to other languages" on page 232). If you want to generate message skeletons for message translation, you must uncomment the HZSNPSKE and HZSLNSKE DD statements:
   - Set HZSNLSKE to the name of the NLS skeleton output data set. This data set should be blocked variable with a record length of 259. After HZSMSGEN runs, this data set contains the prolog from HZSNPSKE and the completed message skeletons for the check messages.
   - Set HZSNPSKE to the name of the NLS skeleton prolog input data set. This data set must contain the version record required for MVS message service (MMS) translation and should contain a copyright statement. The following shows an example HZSNPSKE data set:

```
.VENUNHBB7730 5650-A010601
.*
.*********************************************************************
.*
.* COPYRIGHT  -
.*        5650-ZOS
.*        THIS MESSAGE INSTALL FILE IS "RESTRICTED MATERIALS OF IBM"
.*        (C) COPYRIGHT IBM CORP. 1988, 2005
.*        LICENSED MATERIALS  - PROPERTY OF IBM
.*
.* STATUS = HBB7730
.*
.*
.*
.* NOTE: VERSION RECORD (.V IN COLS. 1-2) MUST APPEAR FIRST IN THIS
.*       FILE. FOR UPDATES, REFER TO APPLICATION DEVELOPMENT GUIDE:
.*       ASSEMBLER LANGUAGE PROGRAMS.
.*
.* CHANGE-ACTIVITY:
```

```
.* $L0=HCHECK   HBB7730,050731,PD00ZJ: HCR8
.*
.*****************************************************************
```

The version record must be the first non-comment record in each install message file identified by the '.v' in columns 1 and 2 of the HZSNLSKE data set. See Creating a version record in *z/OS MVS Programming: Assembler Services Guide* for the complete format of the version record.

3. To run the message generation JCL, Issue the following command from TSO:

   ```
   SUB 'your.dataset.name(HZMSGNJ)'
   ```

4. HZSMSGEN writes a message generation report to either the data set or UNIX System Services file specified in HZSMDSN, if specified, or to the SYSTSPRT file. The following shows an HZSMSGEN report for both CSECT and NLS skeleton generation:

   ```
   IBM Health Checker For Z/OS HBB7730 NLSCHECK(Y) SOURCE(ERROR)

   10 May 2006

   RulesLevel 2 (HBB7730 and up) was selected for processing

   System execution level: z/OS 01.08.00 HBB7730 TSO/E 3060

   Source data set 'SYS1.SAMPLIB(HZSSMSGT)'

   Setup entity data set: 'SYS1.SAMPLIB(HZSSSYMD)'

   Assembler source MSGTBL: 'userid.your.dsname(csect)'

   NLS skeleton prologue: 'input.nls.version.record'

   NLS skeleton source:'output.nls(skeleton)'


   HZSM0133 The assembler source for the message table was created

   Return Code: 0

   HZSM0133 The NLS message skeleton source was created

   Return Code: 0
   ```

Support for translating messages to other languages

In many installations, the text that appears in WTOs may need to be translated to other languages using MVS Message Service (MMS). You can use the HZSMSGEN exec to create the MMS source file, which is required input when translating message to other languages. For information about using MMS for message translation, see Translating messages in *z/OS MVS Programming: Assembler Services Guide*. This section also includes details on how the system uses the NLS skeletons.

Guidelines for coding translatable exception message text lines

If you want to generate skeletons for message translations for your check exception WTO messages, it will impact the way you code the message text for your messages in the message table. For example, if you want to generate NLS skeletons for your messages, you must break up message text in the message table into lines of 71 characters or less. The line length is calculated based on the total length of the message text, and the maximum length that each insert is defined. When you

use HZSMSGEN, you can specify NLSCHECK(Y) to specify that the system enforce the NLS length guideline. HZSMSGEN enforces these restrictions when messages are created.

To make sure that you can generate your exception messages successfully, and that messages will translate successfully at runtime, use the following guidelines to help you calculate the length of each message line to make sure that each line is 71 characters or less:

- On the first line of the message text, remember that the message identifier, or number, can require up to 11 characters.
- You must use "<lines></lines>" on page 224 to define a new line **before** you reach the WTO limit of 71 characters.
- Specify <mv class="*variable_class*" xreftext="maxlen(*nnn*)>" for all the variables in your exception messages to define the maximum length possible for each variable. This will make it much easier for you to calculate where you need to insert a <lines></lines> tag to break up a message text to avoid exceeding the 71 character limit. If you do not specify maxlen, you must allow for the maximum space allowed for the type of variable when calculating where you want to break your line with <lines></lines>. See Table 21 on page 220 for specifics on variable lengths.
- For a predefined system symbol, which is resolved at check run time, you must allow for the maximum space allowed for the element when calculating the number of characters it will take up. See Table 25 on page 227.
- Both system symbols and variables can be longer than 71 characters themselves. This is OK, as long as the lengthy item is followed by a new line indicator (<lines></lines> tags together) or is the very last thing in the message text.

When you specify NLSCHECK(Y) and run the HZSMSGEN exec with he **HZSNPSKE** and **HZSNLSKE** DD statements uncommented, your message skeletons are generated in the output data set specified on the HZSNLSKE DD statement.

- Set variable HZSNPSKE to the name of a sequential data set or a member of a PDS that contains the NLS prologue. The product version record is required by MMS and must be included in NLS prologue to produce a compilable NLS skeleton.
- Set variable HZSNLSKE to the name of a sequential data set or a member of a PDS to be used as output. It will contain the NLS message skeleton when NLSCHECK(Y) is specified and message generation completes with a return code of 0. This data set must have a variable record length of 259.

```
//HZSMSGEN JOB
//*
//        SET  SYSPROC=SYS1.SBLSCLI0(HZSMSGEN)
//        SET  HZSMDSN=SYS1.SAMPLIB(HZSSMSGT)
//        SET  HZSADSN=&SYSUID..TEMP.ASM;
//        SET  HZSSDSN=SYS1.SAMPLIB(HZSSSYMD)
//        SET  HZSNPSKE=&SYSUID..DUMMY.NLS.PROLOGUE;
//        SET  HZSNLSKE=&SYSUID..TEMP.SKEL;
//*
//HZSMSG  EXEC PGM=IKJEFT01,REGION=32M,
//        PARM='%HZSMSGEN NLSCHECK(N) SOURCE(ERROR)'
//SYSTSPRT DD  SYSOUT=*,DCB=(LRECL=132,BLKSIZE=132,RECFM=FB)
//SYSPROC  DD  DISP=SHR,DSN=&SYSPROC;
//SYSTSIN  DD  DUMMY
//HZSMDSN  DD  DISP=SHR,DSN=&HZSMDSN;
//HZSSDSN  DD  DISP=SHR,DSN=&HZSSDSN;
//HZSADSN  DD  DSN=&HZSADSN;,DISP=(NEW,KEEP),
//  SPACE=(TRK,(10,10)),UNIT=SYSDA,DCB=(LRECL=80,BLKSIZE=0,RECFM=FB)
```

```
//HZSNPSKE DD  DSN=&HZSNPSKE;,DISP=SHR
//HZSNLSKE DD  DSN=&HZSNLSKE;,DISP=(NEW,KEEP),
//* SPACE=(TRK,(10,10)),UNIT=SYSDA,DCB=(LRECL=259,BLKSIZE=0,RECFM=VB)
/*
```

When you use the HZSMSGEN exec to generate skeletons for the following messages:

```
CSVH0970E New extents were detected in LNKLST set(s).
CSVH0980E Some LNKLST sets include data set(s) allocated
 with secondary space defined.
```

You will get the following skeletons generated:

```
CSVH0970E          New extents were detected in LNKLST set(s).
CSVH0980E 01001    Some LNKLST sets include data set(s) allocated with
CSVH0980E 01002    secondary space defined.
```

You can customize the timestamp, date, or day generated by timestamp symbols or variables in your messages, by customizing the format for the symbols in the system configuration SYS1.PARMLIB CNL members. See CNLcccxx (Time and date format for translated messages) in *z/OS MVS Initialization and Tuning Reference* for additional information.

For predefined system symbols &hzsgmttime; and &hzslocaltime; the format used in the skeletons is as follows:

```
 &DATE;=DATEMDY4. &TIME;=TIMEHMSCD6
```

Parmlib members CNLENU00 and CNLJPN00 now include symbol TIMEHMSCD6.

Because the length of class=gmltime and localtime variables might vary (for example, if you specify field size, <mv class="gmltime" xreftext=" fieldsize(11)"), the format used in the skeletons will also vary by length as follows. Note that using other lengths than those shown produces results that will not match a date/time NLS skeleton.

Fieldsize	Formatted date/time	NLS skeleton
26	mm/dd/yyyy.hh.mm.ss.tttttt	&DATE=DATEMDY4. &TIME=TIMEHMSCD6.
25	mm/dd/yyyy.hh.mm.ss.ttttt	&DATE=DATEMDY4. &TIME=TIMEHMSCD5.
24	mm/dd/yyyy.hh.mm.ss.tttt	&DATE=DATEMDY4. &TIME=TIMEHMSCD4.
23	mm/dd/yyyy.hh mm.ss.ttt	&DATE=DATEMDY4. &TIME=TIMEHMSCD3.
22	mm/dd/yyyy.hh.mm.ss.tt	&DATE=DATEMDY4. &TIME=TIMEHMSCD2.
21	mm/dd/yyyy.hh.mm.ss.t	&DATE=DATEMDY4. &TIME=TIMEHMSCD1.
19	mm/dd/yyyy.hh.mm.ss	&DATE=DATEMDY4. &TIME=TIMEHMSC.
16	mm/dd/yyyy.hh.mm	&DATE=DATEMDY4. &TIME=TIMEHMC.
13	mm/dd/yyyy.hh	&DATE=DATEMDY4. &hh.
10	mm/dd/yyyy	&DATE=DATEMDY4.

Chapter 11. IBM Health Checker for z/OS System REXX Functions

IBM Health Checker for z/OS includes the following System REXX functions:

- "HZSLSTRT function" on page 236 - The REXX check exec invokes HZSLSTRT to notify IBM Health Checker for z/OS it is running. This call initializes multiple variables defined in the HZSPQE macro.
- "HZSLFMSG function" on page 240 - The REXX check invokes HZSLFMSG to issue messages.
- "HZSLSTOP function" on page 255 - The REXX check invokes HZSLSTOP to notify IBM Health Checker for z/OS of check completion. This request will save the user work area, HZS_PQE_ChkWork.

HZSLSTRT function

Purpose: REXX function indicating that the check has started running. This is the interface to the assembler HZSCHECK REQUEST=OPSTART macro.

Invocation: CALL HZSLSTRT

Input variables

The following REXX variable is input to HZSLSTRT:

Table 27. HZSLSTRT input variable

Variable name	Description
HZS_HANDLE	IBM Health Checker for z/OS sets this variable to the correct value when it calls a REXX check. Your REXX check must not modify HZS_HANDLE. The HZS_HANDLE is used to synchronize the check and IBM Health Checker for z/OS, because they do not run in the same address space. If the HZSLSTRT function is used within a REXX procedure, make sure you used for example the EXPOSE procedure option to make HZS_HANDLE accessible to the procedure code.

Output variables

The following REXX variable are returned by HZSLSTRT:

Table 28. HZSLSTRT output variables

Variable name	Description
HZSLSTRT_RSN	The reason code explaining a RESULT value of 8 or more. The reason codes are as follows: **00000858** **Meaning:** HZS_HANDLE was not valid. **Action:** Make sure that the HZSLSTRT serice is only called from a REXX exec called by IBM Health Checker for z/OS. The check must not modify output variable HZS_HANDLE. If this function is used within a procedure, make sure you used the EXPOSE option to make HZS_HANDLE accessible to the procedure code **xxxx08xx** **Meaning:** HZSCHECK REQUEST=OPSTART returned the HzscheckRC_InvParm reason code equate symbol. **Action:** Refer to the action under the individual reason code for the HZSCHECK macro.Meaning: Action: **00000C16** **Meaning:** The HZSLSTRT function has been invoked by a non-SYSREXX caller. **Action:** Ensure that HZSLSTRT is only called from a REXX health check routine. **xxxx0Cxx** **Meaning:** HZSCHECK REQUEST=OPSTART returned the HzscheckRC_EnvError reason code equate symbol. **Action:** Refer to the action under the individual reason code for the HZSCHECK macro. **00001003** **Meaning:** A service used by HZSLSTRT failed. **Action:** Retry the service. If HZSLSTRT continues to fail, obtain the value of the REXX variable HZSLSTRT_SYSTEMDIAG, and contact IBM service.
HZSLSTRT_SYSTEMDIAG	Diagnostic data returned by the failed service that HZSLSTRT uses.
HZS_PQE_ENTRY_CODE	Contains the numeric identifier (entry code) assigned for the REXX check in the check definition. The entry code is used when a REXX exec contains multiple checks. The system sets this field on entry to the REXX check.
HZS_PQE_DOM_CHECK	Indicates how the DOM(SYSTEM | CHECK) parameter was set when the check was added to IBM Health Checker for z/OS for the current check: **1** The check was added with DOM(CHECK). **0** The check was added with or defaulted to DOM(SYSTEM). For information on how DOM=CHECK works, see “Controlling check exception message WTOs and their automation consequences” on page 126.
HZS_PQE_AllowDynSev	Indicates how the ALLOWDYNSEV(YES | NO) parameter was set when the check was added to IBM Health Checker for z/OS for the current check: **1** The check was added with ALLOWDYNSEV(YES). **0** The check was added with or defaulted to ALLOWDYNSEV(NO). For information on how ALLOWDYNSEV(YES) works, see “Writing a check with dynamic severity levels” on page 124.

Table 28. HZSLSTRT output variables (continued)

Variable name	Description
HZS_PQE_FUNCTION_CODE	Contains the function code in text form for the REXX check. The REXX check receives control in response to either the "RUN" or "INITRUN"function code. The system sets this field on entry to the REXX check.
HZS_PQE_VERSION	The version of the HZSPQE that is used to represent this check.
HZS_PQE_CHECK_COUNT	Number of times this check has been called since the check was initialized.
HZS_PQE_CUM_CHECK_COUNT	The cumulative check iteration number since initialized. This differs from HZS_PQE_CHECK_COUNT in that it is not reset when a refresh occurs. It is updated just before calling the check function, but the check itself might not have confirmed that it got control.
HZS_PQE_ENVIRONMENT _XCFLOCAL	Indicates whether the system is in XCF local mode: **1** Boolean TRUE - System is XCF local mode. **0** Boolean FALSE - System is not XCF local mode.
HZS_PQE_ENVIRONMENT _XCFMONOPLEX	Indicates whether the system is in XCF monoplex mode: **1** Boolean TRUE - System is XCF monoplex mode. **0** Boolean FALSE - System is not XCF monoplex l mode.
HZS_PQE_CHECKOWNER	The product, component, or element that owns the check.
HZS_PQE_CHECKNAME	Check name.
HZS_PQE_GLOBAL _CHECK	Indicates whether the check is defined as global: **1** Boolean TRUE - check is defined as global. **0** Boolean FALSE - check is not defined as global.
HZS_PQE_DEBUG	Indicates whether the check is running in debug mode: **1** Boolean TRUE - check is running in debug mode. **0** Boolean FALSE - check is running in debug mode,
HZS_PQE_LOOKATPARMS	Indicates whether the check should look at the parameter values, either because check parameter values have changed since the last time this check ran, or because it is the first time the check has run after it was in a DISABLED or INACTIVATED state.: **1** Boolean TRUE - The check should look at the parameter values. **0** Boolean FALSE - The check does not need to look at the parameter values.
HZS_PQE_VERBOSE	Indicates whether the check is running in verbose mode: **1** Boolean TRUE - The check is running in verbose mod. **0** Boolean FALSE - The check is not is running in verbose mod.
HZS_PQE_REASON	Current value of the check reason text.
HZS_PQE_PARMAREA	Current check parameter(s). If LENGTH(HZS_PQE_PARMAREA)=0, then no parameters are currently defined for this check.
HZS_PQE_CHKWORK	Current value of the PQE_CHKWORK area saved by the HZSLSTOP service the last time the check ran. Only a maximum of 2048 characters HZS_PQE_CHKWORK will be saved and restored. HZS_PQE_CHCKWORK is reset before the check is run for the following reasons: • When the check is to run for the first time. • When the check is REFRESHed. • When the check becomes either INACTIVE or DISABLED for any reason besides invalid parameters.

HZSLSTRT return codes

The return code for the HZSLSTRT function. The possible return codes are as follows:

0 **Meaning:** HZSLSTRT was invoked while the exec was running under System REXX.

Action: RESULT will be set to the return code of the service.

8 **Meaning:** The HZSLSTRT function did not complete because of an error.

Action: Refer to action under the individual reason code returned in HZSLSTRT_RSN

12 (X'C')
Meaning: HZSLSTRT was not invoked from a System REXX environment.

Action: Make sure the HZSLSTRT serice is only called from an exec that has gotten control as a REXX check called by the IBM Health Checker for z/OS.

16 (X'10')
Meaning: HZSLSTRT did not completed because of an component error.

Action: Refer to action under the individual reason code returned in HZSLSTRT_RSN

20 (X'14')
Meaning: HZSLSTRT encountered problems when storing expected REXX output variables, such as the HZS_PQE_ variables.

Action: Refer to action under the individual reason code returned in HZSLSTRT_RSN

HZSLFMSG function

Purpose: REXX write messages for the check. This is the interface to the assembler HZSFMSG macro. See "HZSFMSG macro — Issue a formatted check message" on page 307.

Invocation: CALL HZSLFMSG

Input variables

The following REXX variables are input to HZSLFMSG:

Table 29. HZSLFMSG input variables

Variable name	Description
HZS_HANDLE	IBM Health Checker for z/OS sets this variable to the correct value when it calls a REXX check. Your REXX check must not modify HZS_HANDLE. The HZS_HANDLE is used to synchronize the check and IBM Health Checker for z/OS, because they do not run in the same address space. If the HZSLFMSG function is used within a REXX procedure, make sure you used for example the EXPOSE procedure option to make HZS_HANDLE accessible to the procedure code.
HZSLFMSG_REQUEST= { 'CHECKMSG' | 'DIRECTMSG' | 'HZSMSG' | 'STOP' | 'DOM'}	• CHECKMSG indicates that the message text is provided in the message table identified by the MSGTBL parameter when the check was added to IBM Health Checker for z/OS. • DIRECTMSG indicates that the messages for this check are issued directly from the check routine - this check does not have a message table associated with it. The message text for this message is provided in the HZSLFMSG_REQUEST='DIRECTMSG' input variables. • HZSMSG indicates that the message text is provided by IBM Health Checker for z/OS. • STOP indicates that the system is to stop calling this check. The message text is provided by IBM Health Checker for z/OS. • DOM indicates that all the check exception WTOs from previous iterations of this check be DOMed. HZSLFMSG_REQUEST=DOM for a check added as DOM(CHECK) gives you control over when exception messages WTOs for a check are DOMed. For information on using this function, see "Controlling check exception message WTOs and their automation consequences" on page 126. HZSLFMSG_REQUEST='DOM' is only allowed: – From within a check routine – Before any check exception messages have been sent in the current check iteration – For checks added with parameter DOM(CHECK). There are no HZSLFMSG_REQUEST=DOM input variables.

Input variables for HZSLFMSG_REQUEST='CHECKMSG'

HZSLFMSG_REQUEST='CHECKMSG' indicates that the message text is provided in the message table identified by the MSGTBL parameter when the check was added to IBM Health Checker for z/OS.

The following REXX variables are required input when HZSLFMSG_REQUEST='CHECKMSG' is specified:

Table 30. HZSLFMSG_REQUEST='CHECKMSG' input variables

Variable name	Description
HZSLFMSG_ MESSAGENUMBER	The message number for the message being issued. This is the value specified in "XREFTEXT=MessageNumber" within the <msgnum> tag of the message source used to create the message table identified by the MSGTBL parameter when the check was added. Must be in the range between 1 and 999999999.
HZSLFMSG_INSERT	REXX stem variable identifying the character variable message inserts.
HZSLFMSG_INSERT.0	The number of inserts or variables provided. This value must match the number of inserts defined in the message and must be in the range between 0 and 20.

HZSLFMSG_INSERT.x	The message insert text. The text provided in the insert should be compatible with the class attribute of the associated message variable in the message table. A class attribute of hex, decimal or timestamp in the message table will treat the insert data as a hexadecimal string.
	
	In the following example, variable HZSLFMSG_INSERT.1 expects to receive hexadecimal data:
	• Variable 1 in the message table has a class attribute of hex: <mv class="hex">*variable 1*</mv>
	• The REXX check might use the following HZSLFMSG input variables:
	`HZSLFMSG_INSERT.1 = '01234567'X  /* A hex character string  */`
	`HZSLFMSG_INSERT.1 = x2c(020B140E)  /* Text that is converted to hexadecimal */`
	
	Note that decimal text also converts hex values to decimal text. For example, lets say that variable in the message table has a class attribute of:
	`<mv class="decimal">`*`variable 1`*`</mv>`
	
	The REXX check use the following HZSLFMSG input variable:
	`HZSLFMSG_INSERT.1 = '0A'X  ->  10  /*  The decimal value 10 is displayed  */`
	
	In general, the REXX values you use will be text and usually do not require additional translation.

Table 30. HZSLFMSG_REQUEST='CHECKMSG' input variables (continued)

Variable name	Description
HZSLFMSG_SEVERITY	HZSLFMSG_SEVERITY={SYSTEM | LOW | MED | HI | NONE} specifies the severity for the check, for checks that are set up to specify check severity dynamically. HZSLFMSG_SEVERITY is only allowed for: • Checks added with parameter ALLOWDYNSEV(YES) to allow the check to specify check severity dynamically • Check exception messages (not for information or report messages) Note that the check is responsible to clear (with a DROP statement) or reset this variable between calls to HZSLFMSG, to avoid having a consecutive HZSLFMSG call pick up the severity specified previously. • SYSTEM, which is the default, indicates that check exceptions be issued with the severity defined for the check when the check was added or updated. • LOW indicates that this check exception message is sent as a low severity message • MED indicates that this check exception message is sent as a medium severity message. • HI indicates that the check exception message is sent as a high severity message. • NONE indicates that the check has no severity. The severity specified on HZSLFMSG_SEVERITY overrides the default severity defined for the check when it was added. The check writer can use specific criteria and check parameters to specify when to use the different severities. See "Writing a check with dynamic severity levels" on page 124. See also HZSPRMxx parameters SEVERITY and WTOTYPE in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Input variables for HZSLFMSG_REQUEST='DIRECTMSG'

HZSLFMSG_REQUEST='DIRECTMSG' indicates that the message text is provided in the DIRECTMSG input variables, rather than in a message table. See "Issuing messages for your check - message table checks versus DIRECTMSG checks" on page 102. Each of these input variables can contain up to 65535 characters of data, except for HZSLFMSG_DIRECTMSG_ID, which can contain up to 10 characters.

Note that the optional variables are named with a dot, to allow easy REXX DROP of variable content between HZSLFMSG calls. For example, variables such as HZSLFMSG_DIRECTMSG**.**EXPL and HZSLFMGS_DIRECTMSG**.**AUTOMATION contain a dot rather than an underscore character. This allows you to use the following statement to clear all the optional variables at one time:

```
DROP HZSLFMSG_DIRECTMSG.
```

Using symbols in DIRECTMSG message texts: You can use the same pre-defined symbols for commonly used phrases in your DIRECTMSG message text as you can in a message table. See "Using pre-defined system symbols" on page 227 for a list of symbols.

Note that using a plain ampersand character (&) is not recommended outside of the name of a supported predefined symbol. Use the symbol & instead. If you use a plain ampersand in your DIRECTMSG text, HZSLFMSG issues warning, RC=4 RSN=1A, but will still issue the DIRECTMSG message.

Required HZSLFMSG_REQUEST='DIRECTMSG' variables: The following REXX variables are required input when HZSLFMSG_REQUEST='DIRECTMSG' is specified:

Table 31. HZSLFMSG_REQUEST='DIRECTMSG' required input variables

Variable name	Description
HZSLFMSG_REASON= {'CHECKEXCEPTION' | 'CHECKINFO' | 'CHECKREPORT'}	Specifies the type of message you are issuing with DIRECTMSG: **CHECKEXCEPTION** specifies that you want to issue an exception message to notify the installation that action is required because a check routine found an exception to a setting. The message text, intended for the operator, is issued in a WTO. The message text and full explanation are issued to the message buffer, mainly for the system programmer. REASON='CHECKEXEPTION' requires the following variables: • HZSLFMSG_DIRECTMSG_ID • HZSLFMSG_DIRECTMSG_TEXT You can also specify the following optional variables on REASON='CHECKEXEPTION': • HZSLFMSG_DIRECTMSG.EXPL • HZSLFMSG_DIRECTMSG.SYSACT • HZSLFMSG_DIRECTMSG.ORESP • HZSLFMSG_DIRECTMSG.SPRESP • HZSLFMSG_DIRECTMSG.PROBD • HZSLFMSG_DIRECTMSG.SOURCE • HZSLFMSG_DIRECTMSG.REFDOC • HZSLFMSG_DIRECTMSG.AUTOMATION • HZSLFMSG_SEVERITY={SYSTEM | LOW | MED | HI | NONE} **CHECKINFO** specifies that you want to issue a general non-exception informational message, such as a message about the completion of the check without exceptions or the first line of a report. The message number and text are issued to the message buffer. REASON='CHECKINFO' requires the following variables: • HZSLFMSG_DIRECTMSG_ID • HZSLFMSG_DIRECTMSG_TEXT There are no optional variables for use with CHECKINFO. **CHECKREPORT** specifies that you want to issue a report message. A report message consists of a single line of report data issued to the message buffer without a message number (except in debug mode). The report message type gives you the most control over the formatted output. A single report line can be up to 65535 characters of formatted output. REASON='CHECKREPORT' requires the HZSLFMSG_DIRECTMSG_TEXT variable. There are no optional variables for use with CHECKREPORT.
HZSLFMSG_DIRECTMSG_ID	The message number for the text message being issued. You can specify HZSLFMSG_DIRECTMSG_ID on a REASON=CHECKEXCEPTION or CHECKINFO request.
HZSLFMSG_DIRECTMSG_TEXT	Use this variable to specify the text issued in a WTO or to the message buffer when the check issues a message using DIRECTMSG. You can specify HZSLFMSG_DIRECTMSG_TEXT on a REASON=CHECKEXCEPTION, CHECKINFO or CHECKREPORT request. For information on how message text is formatted in the message buffer, see "How messages are formatted in the message buffer" on page 225.

Optional HZSLFMSG_REQUEST='DIRECTMSG' variables: The following REXX variables are **optional** and can only be specified on the HZSLFMSG_REASON='CHECKEXCEPTION' request If you do not specify one of these variables, the system does not display this field in the message buffer.

Table 32. HZSLFMSG_REQUEST='DIRECTMSG' optional input variables for HZSLFMSG_REASON='CHECKEXCEPTION'

Variable name	Description
HZSLFMSG_DIRECTMSG.EXPL	Explanation for the exception message describing the exception condition.
HZSLFMSG_DIRECTMSG.SYSACT	Describes what the system or component that owns the check does as a result of the condition that caused the exception message to be issued.
HZSLFMSG_DIRECTMSG.ORESP	Describes the actions an operator should take in response to the exception message.
HZSLFMSG_DIRECTMSG.SPRESP	Describes the actions, if any, the system programmer should take to isolate and correct an error.
HZSLFMSG_DIRECTMSG.PROBD	Describes problem determination information.
HZSLFMSG_DIRECTMSG.SOURCE	The name of the component, subsystem, or product issuing the exception message. For IBM checks, this is used to direct service calls.
HZSLFMSG_DIRECTMSG.REFDOC	Specifies reference information for the exception found by the check.
HZSLFMSG_DIRECTMSG. AUTOMATION	Describes automation concerns related to the check results.
HZSLFMSG_SEVERITY={SYSTEM | LOW | MED | HI | NONE}	Specifies the severity for the check, for checks that are set up to specify check severity dynamically. HZSLFMSG_SEVERITY is only allowed for: • Checks added with parameter ALLOWDYNSEV(YES) to allow the check to specify check severity dynamically • Check exception messages (not for information or report messages) Note that the check is responsible to clear (with a DROP statement) or reset this variable between calls to HZSLFMSG, to avoid having a consecutive HZSLFMSG call pick up the severity specified previously. • SYSTEM indicates that check exceptions be issued with the severity defined for the check when the check was added or updated. • LOW indicates that this check exception message is sent as a low severity message • MED indicates that this check exception message is sent as a medium severity message. • HI indicates that the check exception message is sent as a high severity message. • NONE indicates that the check has no severity. For complete information on how the system handles check exception messages based on the severity you specify, see SEVERITY and WTOTYPE in "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc*" on page 68.

Input variables for HZSLFMSG_REQUEST='HZSMSG'

HZSLFMSG_REQUEST=HZSMSG' indicates that the message text is provided by IBM Health Checker for z/OS.

The following REXX variables are required input when HZSLFMSG_REQUEST='HZSMSG' is specified:

Table 33. HZSLFMSG_REQUEST='HZSMSG' input variables

Variable name	Description
HZSLFMSG_REASON='ERROR'	Indicates that the message is being issued because of an error situation. The system is to issue HZS1002E. This message is also recorded in the check's message buffer. The state of the check is changed to error. The check remains active.
If you specify HZSFMSG_REASON='ERROR", you must also specify the following REXX input variables to identify the error:	

HZSLFMSG_DIAG	Is set to the data to be displayed as hex data in the message output to provide internal component diagnostic information for the error, which is included when check detail is displayed.
	
	The value in HZSLFMSG_DIAG must be either:
	• An 8 character value that will be displayed as hexadecimal value.
	• A 16 character hexadecimal value, that may contain valid hexadecimal characters: 0-9 and A-F only.
	
	Example: Lets say you want the following IBM Health Checker for z/OS message:
	`HZS1002E CHECK(HZJVTT78,HZXVTT78_A_PARMLIB_EXEC):`
	`  AN ERROR OCCURRED, DIAG: 00000000_01234567`
	
	You would define the following input variables:
	`HZSLFMSG_DIAG = '0000000001234567' /* hexadecimal characters */`
	`HZSLFMSG_DIAG = '0000000001234567'X /* hexadecimal data */`

HZSLFMSG_REASON='PARS1201'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1201E, *parm* IS REQUIRED BUT WAS NOT SPECIFIED. The following REXX variables are required input for HZSLFMSG_REASON='PARS1201': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert for HZS1201E. • HZSLFMSG_INSERT.0 = 1 - Indicates that there is one insert. • HZSLFMSG_INSERT.1=*parm* - 1 to 16 character name of the parameter in error.
HZSLFMSG_REASON='PARS1202'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1202E, *parm* WAS SPECIFIED BUT IS NOT ALLOWED. The following REXX variables are required input for HZSLFMSG_REASON='PARS1202': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert. • HZSLFMSG_INSERT.0 = 1 - Indicates that there is one insert. • HZSLFMSG_INSERT.1=*parm* - 1 to 16 character name of the parameter in error.

Table 33. HZSLFMSG_REQUEST='HZSMSG' input variables (continued)

Variable name	Description
HZSLFMSG_REASON='PARS1203'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1203E, PARAMETER *parm* VALUE *value* IS NOT VALID. The following REXX variables are required input when HZSLFMSG_REASON='PARS1203' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.
HZSLFMSG_REASON='PARS1204'	Indicates that the message is being issued for a parameter parsing error, issuing message HZS1204E, UNEXPECTED END OF PARAMETER STRING. The following REXX variables are required input for HZSLFMSG_REASON='PARS1204': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 0 - Indicates that there are no inserts.
HZSLFMSG_REASON='PARS1205'	Indicates that the message is being issued for a parameter parsing error, issuing message HZS1205E, A PARAMETER WAS EXPECTED BUT string WAS FOUND INSTEAD. The following REXX variables are required input for HZSLFMSG_REASON='PARS1205': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 1 - Indicates that there is one insert. • HZSLFMSG_INSERT.1 = *value* - 1 to 17 character string value in error.
HZSLFMSG_REASON='PARS1206'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1206E, A DELIMITER WAS EXPECTED BUT *string* WAS FOUND INSTEAD. The following REXX variables are required input for HZSLFMSG_REASON='PARS1206': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 1 - Indicates that there is one insert. • HZSLFMSG_INSERT.1 = *value* - 1 to 17 character string value in error.
HZSLFMSG_REASON='PARS1207'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1207E, PARAMETER *parm* HAS TOO MANY VALUES, *n*. The following REXX variables are required input when HZSLFMSG_REASON='PARS1207' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *n* - Number of values that were specified. The maximum value that can be specified is 999999999.

Table 33. HZSLFMSG_REQUEST='HZSMSG' input variables (continued)

Variable name	Description
HZSLFMSG_REASON='PARS1208'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1208E, PARAMETER *parm* HAS TOO FEW VALUES, I*n*. The following REXX variables are required input when HZSLFMSG_REASON='PARS1208' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *n* - Number of values that were specified. The maximum value that can be specified is 999999999.
HZSLFMSG_REASON='PARS1209'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1209E, PARAMETER parm IS NOT RECOGNIZED. The following REXX variables are required input for HZSLFMSG_REASON='PARS1209': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert. • HZSLFMSG_INSERT.0 = 1 - Indicates that there is one insert. • HZSLFMSG_INSERT.1=*parm* - 1 to 17 character name of the parameter in error.
HZSLFMSG_REASON='PARS1210'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1210E, PARAMETER *parm* IS MISSING ITS VALUE. The following REXX variables are required input for HZSLFMSG_REASON='PARS1210': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert. • HZSLFMSG_INSERT.0 = 1 - Indicates that there is one insert. • HZSLFMSG_INSERT.1=*parm* - 1 to 16 character name of the parameter in error.
HZSLFMSG_REASON='PARS1211'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1211E, PARAMETER *parm* VALUE *value* IS TOO LARGE. The following REXX variables are required input when HZSLFMSG_REASON='PARS1211' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.

Table 33. HZSLFMSG_REQUEST='HZSMSG' input variables (continued)

Variable name	Description
HZSLFMSG_REASON='PARS1212'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1212E, PARAMETER *parm* VALUE *value* IS TOO SMALL. The following REXX variables are required input when HZSLFMSG_REASON='PARS1212' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.
HZSLFMSG_REASON='PARS1213'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1213E, PARAMETER *parm* VALUE *value* IS TOO LONG. The following REXX variables are required input when HZSLFMSG_REASON='PARS1213' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.
HZSLFMSG_REASON='PARS1214'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1214E, PARAMETER *parm* VALUE *value* IS TOO SHORT. The following REXX variables are required input when HZSLFMSG_REASON='PARS1214' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.
HZSLFMSG_REASON='PARS1215'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1215E, PARAMETER *parm* VALUE *value* IS NOT DECIMAL. The following REXX variables are required input when HZSLFMSG_REASON='PARS1215' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.

Table 33. HZSLFMSG_REQUEST='HZSMSG' input variables (continued)

Variable name	Description
HZSLFMSG_REASON='PARS1216'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1216E, PARAMETER *parm* VALUE *value* IS NOT HEXADECIMAL. The following REXX variables are required input when HZSLFMSG_REASON='PARS1216' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.
HZSLFMSG_REASON='PARS1217'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1217E, PARAMETERS WERE SPECIFIED BUT NONE ARE NOT ALLOWED. The following REXX variables are required input for HZSLFMSG_REASON='PARS1217': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert. • HZSLFMSG_INSERT.0 =0 - Indicates that there are no inserts.
HZSLFMSG_REASON='PARS1218'	indicates that the message is being issued due to a parameter parsing error, issuing message HZS1218E, PARAMETER NUMBER *n* WAS NOT PROCESSED. The following REXX variables are required input when HZSLFMSG_REASON='PARS1218' is specified: • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert . • HZSLFMSG_INSERT.0 = 1 - Indicates that there are two inserts. • HZSLFMSG_INSERT.1 = *n* - Number of the parameter that was not processed, the maxamum value that can be specified is 999999999.
HZSLFMSG_REASON='PARS1219'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1219E, MIXING POSITIONAL AND KEYWORD FORMATS IS NOT ALLOWED. The following REXX variables are required input for HZSLFMSG_REASON='PARS1219': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert. • HZSLFMSG_INSERT.0 =0 - Indicates that there are no inserts.
HZSLFMSG_REASON='PARS1220'	Indicates that the message is being issued due to a parameter parsing error, issuing message HZS1220E, *parm1* IS NOT ALLOWED WITH *parm2*. The following REXX variables are required input for HZSLFMSG_REASON='PARS1220': • HZSLFMSG_INSERT - REXX stem variable that is used to identify the message insert. • HZSLFMSG_INSERT.0 = 2 - Indicates that there are two inserts • HZSLFMSG_INSERT.1 = *parm1* - 1-24 character name of the parameter • HZSLFMSG_INSERT.2 = *parm2* - 1-24 character name of the parameter

Input variables for HZSLFMSG_REQUEST='STOP'

HZSLFMSG_REQUEST='STOP' indicates that the system is to stop calling this check. The message text is provided by IBM Health Checker for z/OS.

The following REXX variables are required input when HZSLFMSG_REQUEST='STOP' is specified:

Table 34. HZSLFMSG_REQUEST='STOP' input variables

Variable name	Description
HZSLFMSG_REASON='BADPARM'	Indicates that the parameters are not valid. The system issues message HZS1001E. This message is also recorded in the check's message buffer. The state of the check is changed to parameter error. The check remains disabled until the PARMS are changed, presumably to address the error.
HZSLFMSG_REASON='ERROR'	Indicates that the message is being issued because of error. The system is to issue HZS1002E. The state of the check is changed to error. The check is disabled. The check will not be called again until the check is refreshed. The following REXX variable is required input for HZSLFMSG_REASON='ERROR'': • HZSLFMSG_DIAG - is set to the data to be displayed as hex data in the message output to provide diagnostic information for the failure that is being reported. There is no pre-defined format for this data; it may well be internal component diagnostic data. The value in HZSLFMSG_DIAG must be either: – An 8 character value that will be displayed as hexadecimal value. – A 16 character hexadecimal value, that may contain valid hexadecimal characters: 0-9 and A-F only.
HZSLFMSG_REASON='ENVNA'	Indicates that the check is not applicable in the current system environment. Message HZS1003E is written as hardcopy-only and is also written to the check's message buffer. The state of the check is changed to not applicable. The check is disabled. The check will not be called again until the reason for the condition is resolved and the check is refreshed (or its parameter is changed).

HZSLFMSG Output variables

The following REXX variable are returned by HZSLFMSG:

Table 35. HZSLFMSG output varaibles

Variable name	Description
HZSLFMSG_RSN	The reason code explaining a return code of 8 or more. The reason codes are as follows: **00000858** **Meaning:** HZS_HANDLE was not valid. **Action:** Make sure that the HZSLFMSG function is only called from a REXX exec called by IBM Health Checker for z/OS. The check exec must not modify output variable HZS_HANDLE. If this function is used within a procedure, make sure you used the EXPOSE option to make HZS_HANDLE accessible to the procedure code **00000890** **Meaning:** HZSLFMSG_REQUEST is not valid. **Action:** Make sure HZSLFMSG_REQUEST is set to a valid value. The valid values are 'CHECKMSG', 'HZSMSG' and 'STOP'. **00000891** **Meaning:** HZSLFMSG_DIAG is not valid. **Action:** Make sure the HZSLFMSG_DIAG is set to a valid value: • An 8 character value that will be displayed as a hexadecimal value. • A 16 character hexadecimal value, that may contain valid hexadecimal characters: 0-9 and A-F only. **00000892** **Meaning:** HZSLFMSG_REASON is not valid . **Action:** Make sure the HZSLFMSG_REASON is set to a valid value: • When HZSLFMSG_REQUEST='HZSMSG', the valid values of HZSLFMSG_REASON are: { 'ERROR' | 'PARS1201' | 'PARS1202' | 'PARS1203' | 'PARS1204' | 'PARS1205' | 'PARS1206' | 'PARS1207' | 'PARS1208' | 'PARS1209' | 'PARS1210' | 'PARS1211' | 'PARS1212' | 'PARS1213' | 'PARS1214' | 'PARS1215' | 'PARS1216' | 'PARS1217' | 'PARS1218' | 'PARS1219' } • When HZSLFMSG_REQUEST='STOP', the valid values of HZSLFMSG_REASON are: { 'BADPARM' | 'ERROR' | 'ENVNA' } **00000893** **Meaning:** HZSLFMSG_MESSAGENUMBER is not valid. **Action:** Make sure the HZSLFMSG_MESSAGENUMBER is set to a valid decimal number that identifies the desired message to be written. **00000894** **Meaning:** HZSLFMSG_INSERT 0 is not valid . **Action:** Make sure the stem variable HZSLFMSG_INSERT 0 is set to the number of message inserts defined for the message that is to be written. The minimum number of message inserts that can be defined for a message is zero (0). The maximum number of inserts that can be defined for a message is twenty (20).

Table 35. HZSLFMSG output varaibles (continued)

Variable name	Description
	00000895 **Meaning:** HZSLFMSG_INSERT.xx is not valid . **Action:** Make sure HZSLFMSG_INSERT.xx is valid. Each insert is limited to 256 characters. Numeric inserts for PARS12yy messages must be a decimal number between 0 and 999999999. The first 2 characters of HZSFMSG_RSN identifies which insert is not valid. **0000089F** **Meaning:** HZSLFMSG service issued a 290 ABEND . **Action:** Look at the data returned in HZSLFMSG_USERRSN and HZSLFMSG_ABENDRESULT to determine the problem: **HZSLFMSG_USERRSN** 290 ABEND reason code (see "ABEND Codes" on page 332). **HZSLFMSG_ABENDRESULT** ABEND result string returned by HZSFMSG service. **000008A0** **Meaning:** HZSLFMSG_DIRECTMSG_ID is not valid. **Action:** Make sure the required HZSLFMSG_DIRECTMSG_ID REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword NUM. **000008A1** **Meaning:** HZSLFMSG_DIRECTMSG_TEXT is not valid. **Action:** Make sure the required HZSLFMSG_DIRECTMSG_TEXT REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword TEXT. **000008A2** **Meaning:** HZSLFMSG_DIRECTMSG.EXPLN is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.EXPLN REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword EXPL. **000008A3** **Meaning:** HZSLFMSG_DIRECTMSG.SYSACT is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.SYSACT REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword SYSACT. **000008A4** **Meaning:** HZSLFMSG_DIRECTMSG.ORESP is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.ORESP REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword ORESP. **000008A5** **Meaning:** HZSLFMSG_DIRECTMSG.SPRESP is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.SPRESP REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword SPRESP.

Table 35. HZSLFMSG output varaibles (continued)

Variable name	Description
	000008A6 **Meaning:** HZSLFMSG_DIRECTMSG.PROBD is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.PROBD REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword PROBD. **000008A7** **Meaning:** HZSLFMSG_DIRECTMSG.SOURCE is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.SOURCE REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword SOURCE. **000008A8** **Meaning:** HZSLFMSG_DIRECTMSG.REFDOC is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.REFDOC REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword REFDOC. **000008A9** **Meaning:** HZSLFMSG_DIRECTMSG.AUTOMATION is not valid. **Action:** Make sure the optional HZSLFMSG_DIRECTMSG.AUTOMATION REXX variable is set to a valid text string which does not exceed the maximum length as documented for the HZSFMSG macro keyword AUTOMATION. **xxxx08xx** **Meaning:** HzsfmsgRc_EnvParm was returned by the HZSFMSG macro. **Action:** Refer to the action under "Return and Reason Codes" on page 335 the HZSFMSG macro. **00000C16:** **Meaning:** The HZSLFMSG function has been invoked by a non-SYSREXX caller. **Action:** Ensure that HZSLFMSG is only called from a REXX health check routine. **xxxx0Cxx** **Meaning:** HzsfmsgRc_EnvError was returned by the HZSFMSG macro. **Action:** Refer to the action under "Return and Reason Codes" on page 335 the HZSFMSG macro. **00001003** **Meaning:** A service used by HZSLFMSG failed. **Action:** Retry the service, if HZSLFMSG continues to fail, obtain the value of the REXX variable HZSLFMSG_SYSTEMDIAG, and contact IBM service.
HZSLFMSG_SYSTEMDIAG	Diagnostic data returned by the failed service.

HZSLFMSG return codes

The return code for the HZSLFMSG function. The possible return codes are as follows:

0 **Meaning:** Service was invoked while the exec was running under System REXX.

Action: RESULT will be set to the return code of the service.

8 **Meaning:** The HZSFMSG request specified incorrect parameters.

Action: Refer to action under the individual reason code returned in HZSLFMSG_RSN.

12 (X'C')
Meaning: HZSLFMSG was not invoked from a System REXX environment.

Action: Make sure the HZSLFMSG service is only called from an exec that has gotten control as a REXX check called by the IBM Health Checker for z/OS.

16 (X'10')
Meaning: HZSLFMSG did not completed because of an component error.

Action: Refer to action under the individual reason code returned in HZSLFMSG_RSN.

20 (X'14')
Meaning: HZSLFMSG encountered problems when storing expected REXX output variables.

Action: Refer to action under the individual reason code returned in HZSLFMSG_RSN.

HZSLSTOP function

Purpose: REXX function indicating that the check has finished running. This is the interface to the assembler HZSCHECK REQUEST=OPCOMPLETE macro.

Invocation: CALL HZSLSTOP

Input variables

The following REXX variable is input to HZSLSTOP:

Table 36. HZSLSTOP input variable

Variable name	Description
HZS_HANDLE	IBM Health Checker for z/OS sets this variable to the correct value when it calls a REXX check. Your REXX check must not modify HZS_HANDLE. The HZS_HANDLE is used to synchronize the check and IBM Health Checker for z/OS, because they do not run in the same address space. If the HZSLSTOP function is used within a REXX procedure, make sure you used for example the EXPOSE procedure option to make HZS_HANDLE accessible to the procedure code.
HZS_PQE_CHKWORK	Current value of the PQE_CHKWORK area. Only a maximum of 2048 characters HZS_PQE_CHKWORK will be saved and restored. HZS_PQE_CHCKWORK is reset before the check is run for the following reasons: • When the check is to run for the first time. • When the check is REFRESHed. • When the check becomes either INACTIVE or DISABLED for any reason besides invalid parameters.

Output variables

The following REXX variable are returned by HZSLSTOP:

Table 37. HZSLSTOP output variables

Variable name	Description
HZSLSTOP_RSN	The reason code explaining an RESULT value of 4 or more. The reason codes are as follows: **00000401** **Meaning:** HZS_PQE_CHKWORK exceeded 2048 bytes. Only the first 2048 bytes of HZS_PQE_CHKWORK will be saved. **Action:** Do not set HZS_PQE_CHKWORK to a character string longer the 2048 characters. **00000858** **Meaning:** HZS_HANDLE was not valid. **Action:** Make sure that the HZSLSTOP serice is only called from a REXX exec called by IBM Health Checker for z/OS. The check must not modify output variable HZS_HANDLE. If this function is used within a procedure, make sure you used the EXPOSE option to make HZS_HANDLE accessible to the procedure code **00000C01** **Meaning:** IBM Health Checker for z/OS is not active. **Action:** Reissue the function when IBM Health Checker for z/OS is active. **00000C03** **Meaning:** The check issued the HZSLSTOP function for a check that was not started. **Action:** Ensure that an HZSLSTART function is issued before a HZSLFMSG or HZSLSTOP function. **00000C05** **Meaning:** The system does not support System REXX checks. **Action:** Run the check on a system at the z/OS R9 level or above. **00000C16** **Meaning:** The HZSLSTOP function has been invoked by a non-SYSREXX caller. **Action:** Ensure that HZSLSTOP is only called from a REXX health check routine. **00001003** **Meaning:** A service used by HZSLSTOP failed. **Action:** Retry the service. If HZSLSTOP continues to fail, obtain the value of the REXX variable HZSLSTOP_SYSTEMDIAG, and contact IBM service.
HZSLSTOP_SYSTEMDIAG	Diagnostic data returned by the failed service that HZSLSTRT uses.

HZSLSTOP return codes

The return code for the HZSLSTOP function. The possible return codes are as follows:

0 **Meaning:** HZSLSTOP was invoked while the exec was running under System REXX.

Action: RESULT will be set to the return code of the service.

4 **Meaning:** HZSLSTOP completed with a warning.

Action: Refer to the action under the individual reason code returned in HZSLSTOP_RSN

8 **Meaning:** The HZSLSTOP function did not complete because of an error.

Action: Refer to action under the individual reason code returned in HZSLSTOP_RSN

12 (X'C')
Meaning: HZSLSTOP was not invoked from a System REXX environment.

Action: Make sure the HZSLSTOP service is only called from an exec that has gotten control as a REXX check called by the IBM Health Checker for z/OS.

16 (X'10')
Meaning: HZSLSTOP did not completed because of an component error.

Action: Refer to action under the individual reason code returned in HZSLSTOP_RSN.

20 (X'14')
Meaning: HZSLSTOP encountered problems when storing expected REXX output variables.

Action: Refer to action under the individual reason code returned in HZSLSTOP_RSN.

Chapter 12. IBM Health Checker for z/OS HZS macros

- Use "HZSADDCK macro — HZS add a check" on page 260 to define a check in a HZSADDCHECK exit routine
- Use "HZSCHECK macro — HZS Check command request" on page 279 to manage a check and in registration routines, to refresh a check
- Use "HZSCPARS macro — HZS Check Parameter Parsing" on page 294 to parse check parameters
- Use "HZSFMSG macro — Issue a formatted check message" on page 307 to issue messages in check routines
- Use "HZSPREAD macro — Read Check Persistent Data" on page 339 to read data that has been preserved in IBM Health Checker for z/OS Persistent data set.
- Use "HZSPWRIT macro — Write Check Persistent Data" on page 349 to write persistent data in the IBM Health Checker for z/OS Persistent data set.
- Use "HZSQUERY macro — HZS Query" on page 357 to obtain information about checks that are currently registered with IBM Health Checker For z/OS.

HZSADDCK macro — HZS add a check

Description

The HZSADDCK macro is used by the HZSADDCHECK dynamic exit routine or by remote, non-REXX check routines to add a check to IBM Health Checker for z/OS. Adding a check includes defining default values, the parameters and routines required to run the check. The exit routine runs in the IBM Health Checker for z/OS address space.

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	Problem state. PSW key 8-15 When problem state and key 8-15 and not APF authorized, or when SECCHECK=ALL is specified, The caller must be authorized for control access to any of the following: • XFACILIT class resource HZS.sysname.checkowner.ADD • XFACILIT class resource HZS.sysname.checkowner.checkname.ADD
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space. Control parameters must be below 2G.

Programming Requirements

- Unless specifying REMOTE=YES and REXX=NO, this macro must be invoked from an exit routine associated with the HZSADDCHECK dynamic exit. Otherwise, it must be invoked from the remote check routine task.
- The check routine and the message table must be in an APF-authorized library.
- The caller should include the HZSZCONS macro to get equate symbols for the return and reason codes.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSADDCK macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSADDCK macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code, when register 15 is not 0.
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as work registers by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

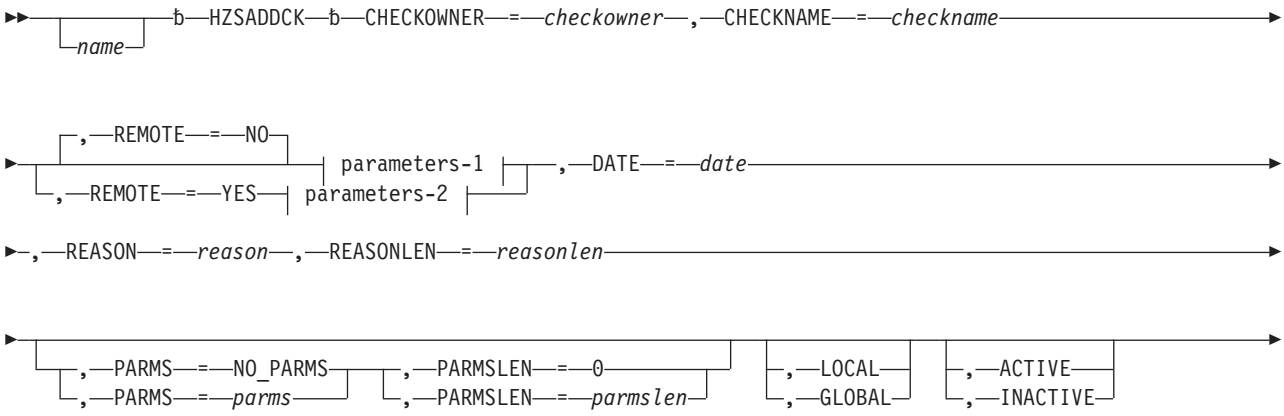
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

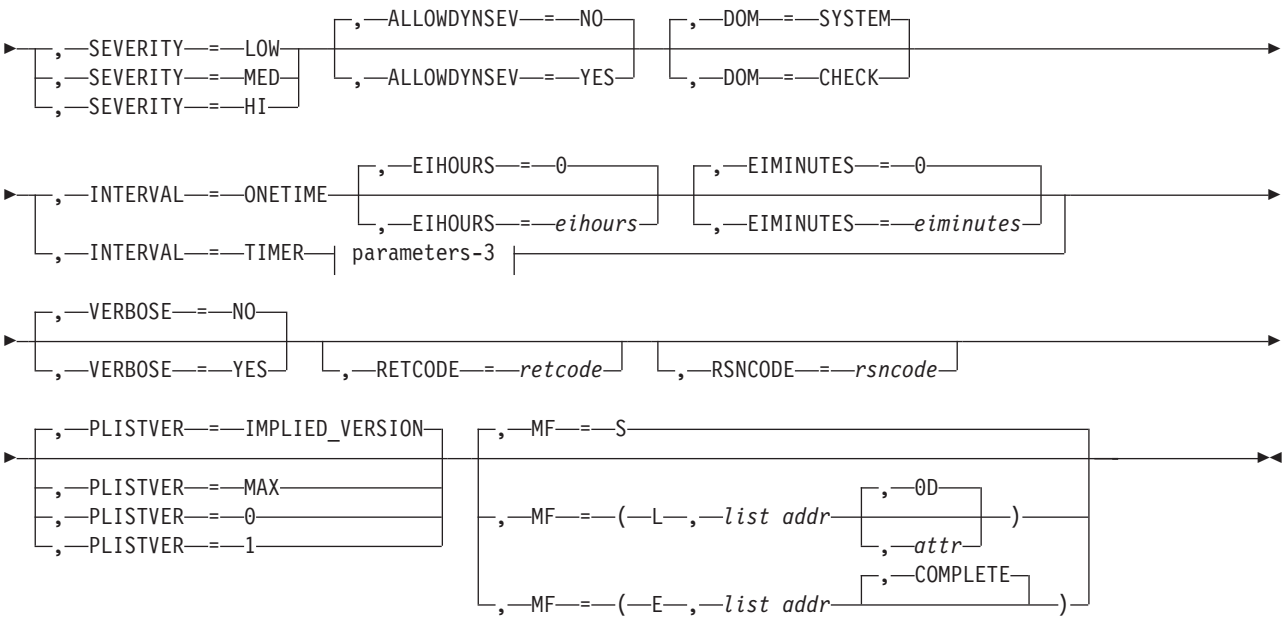
None.

Syntax

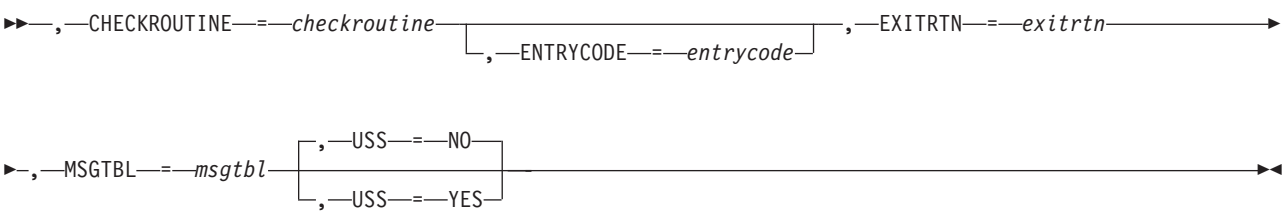
main diagram



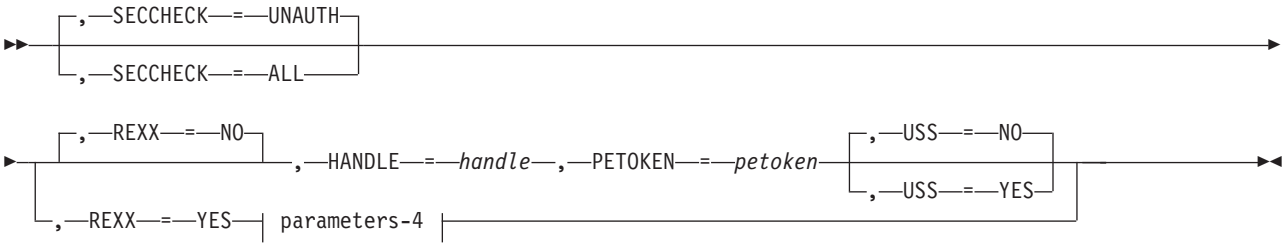
HZSADDCK macro



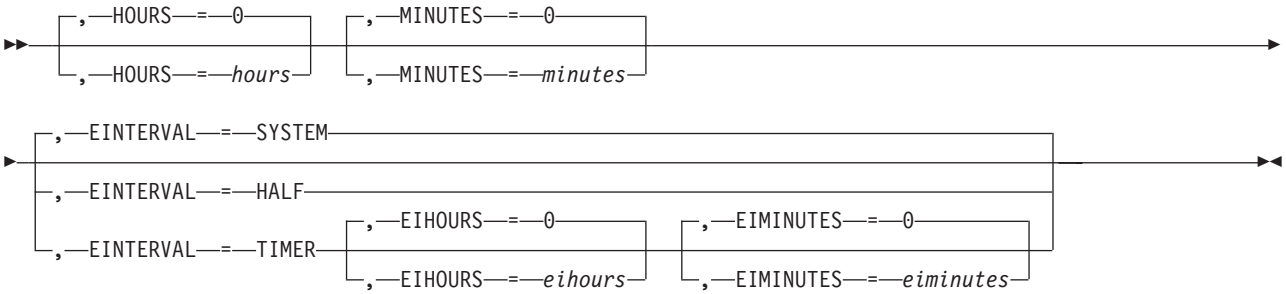
parameters-1



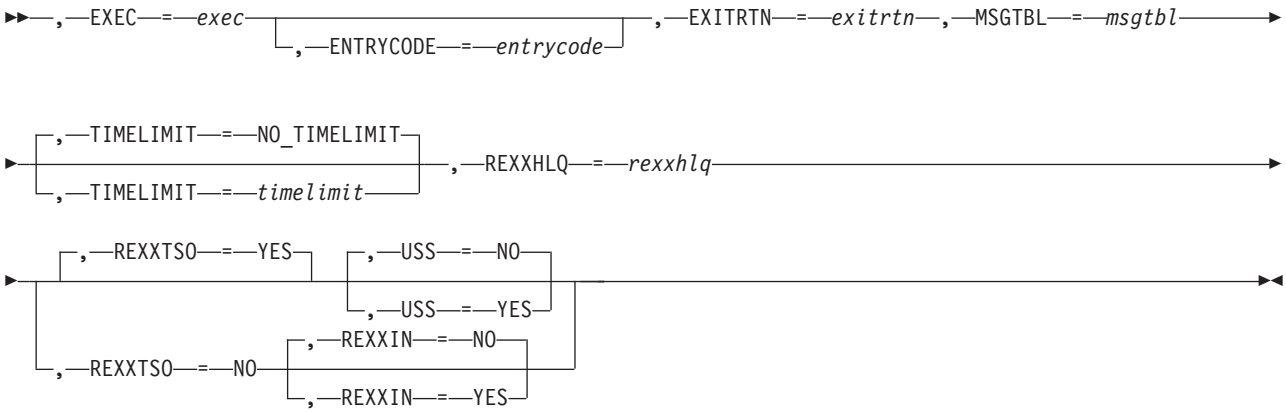
parameters-2



parameters-3



parameters-4



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the HZSADDCK macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ACTIVE
An optional input parameter that indicates the check should run when it is added to the system.

To code: Specify a value.

,ALLOWDYNSEV=NO
,ALLOWDYNSEV=YES
An optional parameter, which indicates if this check is allowed to use keyword SEVERITY on an invocation of service HZSFMSG, to send check messages with a dynamic severity. The default is ALLOWDYNSEV=NO.

,ALLOWDYNSEV=NO
indicates that the check is not allowed to specify a dynamic severity. This is the default.

,ALLOWDYNSEV=YES
indicates that the check is allowed to specify a dynamic severity.

,CHECKNAME=*checkname*
A required input parameter that specifies the name of the check being added. IBM recommends using the naming convention of a short component reference followed by a descriptive title (e.g., GRS_MODE). Upper and lower case alphabetic characters(a-z), numerics (0-9), national characters (@,$,#) and the underscore ('_') are allowed. Lower case alphabetic characters are folded to upper case and are treated as equivalent to their corresponding upper case value.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

CHECKOWNER=*checkowner*
A required input parameter that specifies the owner of the check being added. The check owner and check name identify the check. IBM recommends that you use your company name followed by the short component name (i.e., IBMGRS) as the owner. Upper and lower case alphabetic characters(a-z),

numerics (0-9), national characters (@,$,#) and the underscore ('_') are allowed. Lower case alphabetic characters are folded to upper case and are treated as equivalent to their corresponding upper case value. Do not use as the checkowner any of the following: QUERY, MESSAGES, ACTIVATE, DEACTIVATE, UPDATE, RUN, REFRESH, DELETE, ADDNEW.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CHECKROUTINE=*checkroutine*
When REMOTE=NO is specified, a required input parameter that specifies the module name of the check. The system gives control to the entry point of this module to run the check. The check routine module must be in an APF-authorized library. For a non-remote check, the system must be able to locate the check routine within the joblib or steplib of the IBM Health Checker for z/OS address space, the LPA, or the LNKLST.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,DATE=*date*
A required input parameter, date (its format is YYYYMMDD) that indicates when the default values for the check were defined. When two HZSADDCK requests are received with the same check owner and check name, the request with the latest date will be honored. When the date provided on a matching POLICY UPDATE or POLICY DELETE statement is older than this date, that policy statement is not applied to this check.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,DOM=SYSTEM
,DOM=CHECK
An optional parameter that messages from previous check iterations via Delete Operator Message (DOM) requests. The default is DOM=SYSTEM.

,DOM=SYSTEM
indicates that the system will, just before the start of a new check iteration, execute the DOM requests for all check exception message WTOs from the previous check iteration, if there are any. This is the default.

,DOM=CHECK
indicates that the check will use the HZSFMSG REQUEST=DOM service to issue DOM requests for check exception WTOs from previous check iterations. The system will take care of the DOM only in the following cases:

- when a health check gets deactivated, disabled, deleted, or refreshed
- when Health Checker ends

,EIHOURS=*eihours*
,EIHOURS=0
When INTERVAL=ONETIME is specified, an optional input parameter that specifies the number of hours in the exception interval. It must be in the range 0 through 999. If both EIHours and EIMinutes specify 0, no exception interval is processed. The default is 0.

To code: Specify the RS-type address of a halfword field, or specify a literal decimal value. *eihours* must be in the range 0 through 999.

,EIHOURS=*eihours*

,EIHOURS=0
When EINTERVAL=TIMER and INTERVAL=TIMER are specified, an optional input parameter that specifies the number of hours in the exception interval. It must be in the range 0 through 999. The default is 0.

To code: Specify the RS-type address of a halfword field, or specify a literal decimal value. *eihours* must be in the range 0 through 999.

,EIMINUTES=*eiminutes*
,EIMINUTES=0
When INTERVAL=ONETIME is specified, an optional input parameter that specifies the number of minutes in the exception interval It must be in the range 0 through 59. If both EIHours and EIMinutes specify 0, no exception interval is processed. The default is 0.

To code: Specify the RS-type address of a halfword field, or specify a literal decimal value. *eiminutes* must be in the range 0 through 59.

,EIMINUTES=*eiminutes*
,EIMINUTES=0
When EINTERVAL=TIMER and INTERVAL=TIMER are specified, an optional input parameter that specifies the number of minutes in the exception interval It must be in the range 0 through 59. The default is 0.

To code: Specify the RS-type address of a halfword field, or specify a literal decimal value. *eiminutes* must be in the range 0 through 59.

,EINTERVAL=SYSTEM
,EINTERVAL=HALF
,EINTERVAL=TIMER
When INTERVAL=TIMER is specified, an optional parameter that specifies the time exception interval for the next running of the check. If the previous running of the check resulted in an exception, then this interval is to be used. The default is EINTERVAL=SYSTEM.

,EINTERVAL=SYSTEM
indicates that the check should run according to system rules (namely, according to the interval parameter).

,EINTERVAL=HALF
indicates that the check should run when one half of the interval according to the interval parameter has expired. This value is rounded up to a whole number of minutes.

,EINTERVAL=TIMER
indicates that a timer is used to reschedule the check. The number of hours is combined with the number of minutes to determine how long after the completion of the check routine's running the next running of the check routine should occur. When both the hours and minutes values are zero, the system treats this as if EINTERVAL=SYSTEM had been specified.

,ENTRYCODE=*entrycode*
When REMOTE=NO is specified, an optional input parameter that specifies a unique check entry value when the same check routine will be accessed by multiple checks. This value is passed to the check routine in the field Pqe_Entry_Code.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ENTRYCODE=*entrycode*
When REXX=YES and REMOTE=YES are specified, an optional input

parameter that specifies a unique check entry value when the same check routine will be accessed by multiple checks. This value is passed to the check routine in the field Pqe_EntryCode.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,EXEC=*exec*
When REXX=YES and REMOTE=YES are specified, a required input parameter that is the name of the REXX exec to be invoked.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,EXITRTN=*exitrtn*
When REMOTE=NO is specified, a required input parameter that specifies the name of the exit routine that invoked this HZSADDCK request.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,EXITRTN=*exitrtn*
When REXX=YES and REMOTE=YES are specified, a required input parameter that specifies the name of the exit routine that invoked this HZSADDCK request.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,GLOBAL
An optional input parameter that indicates the check should run on only one system in a sysplex. The system on which the check runs is designated as the global system for that check. Serialization for the global check is accomplished via exclusive ownership of SCOPE=SYSTEMS ENQ with QNAME SYSZHZS and RNAME checkowner.checkname.

To code: Specify a value.

,HANDLE=*handle*
When REXX=NO and REMOTE=YES are specified, a required output parameter that is to hold a handle (token) that identifies the check. This handle is to be used on the HANDLE parameter of the HZSCHECK and HZSFMSG macros.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,HOURS=*hours*
,HOURS=0
When INTERVAL=TIMER is specified, an optional input parameter that specifies the number of hours. It must be in the range 0 through 999. The default is 0.

To code: Specify the RS-type address of a halfword field, or specify a literal decimal value. *hours* must be in the range 0 through 999.

,INACTIVE
An optional input parameter that Indicates the check should not run until the state is changed to active.

To code: Specify a value.

,INTERVAL=ONETIME

,INTERVAL=TIMER
A required parameter that specifies the time interval for the next running of the check.

,INTERVAL=ONETIME
indicates that the check should run once. It will not be rescheduled.

,INTERVAL=TIMER
indicates that a timer is used to reschedule the check. The number of hours is combined with the number of minutes to determine how long after the completion of the check routine's running the next running of the check routine should occur. When both the hours and minutes values are zero, the system treats this as if INTERVAL=ONETIME had been specified.

,LOCAL
An optional input parameter that indicates the check should run on this system.

To code: Specify a value.

,MF=S
,MF=(L,*list addr***)**
,MF=(L,*list addr***,***attr***)**
,MF=(L,*list addr***,0D)**
,MF=(E,*list addr***)**
,MF=(E,*list addr***,COMPLETE)**
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

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

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MINUTES=*minutes*

,MINUTES=0

When INTERVAL=TIMER is specified, an optional input parameter that specifies the number of minutes. It must be in the range 0 through 59. The default is 0.

To code: Specify the RS-type address of a halfword field, or specify a literal decimal value. *minutes* must be in the range 0 through 59.

,MSGTBL=*msgtbl*

When REMOTE=NO is specified, a required input parameter that specifies the module name of the message table that will be used when generating messages for the check.

- The message table must be built using the HZSMSGEN REXX exec.
- The message table module must be in an APF-authorized library.
- The system must be able to locate the message table within the joblib or steplib of the IBM Health Checker for z/OS address space, the LPA, or the LNKLST.

A special text value of '*NONE ' indicates that you are adding a check that has no associated message table and instead the check will be using service HZSFMSG REQUEST=DIRECTMSG or REXX function HZSLFMSG with HZSLFMSG_REQUEST='DIRECTMSG' to issue messages directly from the check routine.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,MSGTBL=*msgtbl*

When REXX=YES and REMOTE=YES are specified, a required input parameter that specifies the module name of the message table that will be used when generating messages for the check.

- The message table must be built using the HZSMSGEN REXX exec.
- The message table module must be in an APF-authorized library.
- The system must be able to locate the message table within the joblib or steplib of the IBM Health Checker for z/OS address space, the LPA, or the LNKLST.

A special text value of '*NONE ' indicates that you are adding a check that has no associated message table and instead the check will be using service HZSFMSG REQUEST=DIRECTMSG or REXX function HZSLFMSG with HZSLFMSG_REQUEST='DIRECTMSG' to issue messages directly from the check routine.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,PARMS=*parms*
,PARMS=NO_PARMS

An optional input parameter that specifies the default parameters for the check. The length of the parameter string is specified by the PARMSLEN parameter. Alphanumeric or national characters separated by commas are the standard form of expressing check parameters. IBM recommends that each parameter be of the form "keyword(value)" and that multiple parameters be separated from each other by a comma. An example of a parameter string following that protocol is "MAXLEN(8),MINLEN(1)". Although the parameters are not checked when the check is added, the check routine itself will likely do so. The default is NO_PARMS.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,PARMSLEN=*parmslen*
,PARMSLEN=0
When PARMS=*parms* is specified, a required input parameter that contains the length of the default parameters for each check. The length must be in the range 1 through 256. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value. *parmslen* must be in the range 0 through 256.

,PETOKEN=*petoken*
When REXX=NO and REMOTE=YES are specified, a required input parameter that is a pause element token obtained by the caller via the IEAVAPE service using an authlvl of IEA_UNAUTHORIZED (even if the caller is authorized). The caller, waiting to be told what to do by IBM Health Checker for z/OS, should pause using that pause element token. IBM Health Checker for z/OS will "release" using that pause element token to wake up the check processing.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, which supports all parameters except those specifically referenced in higher versions.
- **1**, which supports both the following parameters and those from version 0:
  - EXEC
  - HANDLE
  - PETOKEN
  - REXXHLQ
  - TIMELIMIT

To code: Specify one of the following:

- IMPLIED_VERSION

- MAX
- A decimal value of 0, or 1

,REASON=*reason*
A required input parameter that indicates what the check routine validates. The text is limited to 126 characters.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,REASONLEN=*reasonlen*
A required input parameter that contains the length of the Reason text. It must be in the range 1 through 126.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,REMOTE=NO
,REMOTE=YES
An optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,REXX=NO
,REXX=YES
When REMOTE=YES is specified, an optional parameter, which identifies if this is a REXX check The default is REXX=NO.

,REXX=NO
indicates that this is not a REXX check.

,REXX=YES
indicates that this is a REXX check

,REXXHLQ=*rexxhlq*
When REXX=YES and REMOTE=YES are specified, a required input parameter that specifies the high level qualifier for data sets(s) to be made available to the Rexx exec. The output data set (such as the one to which the 'say' function would send its output) is made available when the check is in debug mode and not otherwise. When there is no entry code, or the entry code is 0, the output data set name for a high level qualifier of 'HLQ' will be 'HLQ.execname.REXXOUT'. When there is a non-0 entry code, the output data set name will be 'HLQ.execname.REXXOUT.En' where n is the decimal value of the entry code. If the entry code exceeds 9999999, the value modulo 10000000 will be used. The system will not make any attempt to ensure that the data sets are unique beyond this naming convention. If not already allocated, the data set will be allocated by the system. If the data set is to be created, the IBM

Health Checker for z/OS address space identity must have the authority to create the data set. If the system does attempt to create or use the data set and is not successful, the check routine will not run successfully. The input data set name will be formed using a similar protocol, changing only REXXOUT to REXXIN. The use of the REXXIN data set is controlled by the REXXIN parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,REXXIN=NO
,REXXIN=YES
When REXXTSO=NO, REXX=YES and REMOTE=YES are specified, an optional parameter that indicates if there is a REXX input data set. The default is REXXIN=NO.

,REXXIN=NO
indicates that there is no REXX input data set.

,REXXIN=YES
indicates that the REXX input data set does exist and is to be made available to the exec. Its naming convention is described under the REXXHLQ parameter. If the data set does not exist, the exec will not successfully be given control.

,REXXTSO=YES
,REXXTSO=NO
When REXX=YES and REMOTE=YES are specified, an optional parameter that indicates if this REXX exec needs access to TSO functions. The default is REXXTSO=YES.

,REXXTSO=YES
indicates that the REXX exec needs TSO functions. The exec will execute in a TSO host command environment.

,REXXTSO=NO
indicates that the REXX exec does not need TSO functions. The exec will execute in a MVS host command environment.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SECCHECK=UNAUTH
,SECCHECK=ALL
When REMOTE=YES is specified, an optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases for remote checks. If RACF does not grant authority, the request is rejected.

,SEVERITY=LOW
,SEVERITY=MED

,SEVERITY=HI
A required parameter that indicates the severity assigned to the check.

,SEVERITY=LOW
indicates that this is a low-severity check. When a low-severity check detects an exception, an informational WTO is issued.

,SEVERITY=MED
indicates that this is a medium-severity check. When a medium-severity check detects an exception, an eventual action WTO is issued.

,SEVERITY=HI
indicates that this is a high-severity check. When a high-severity check detects an exception, a critical eventual action WTO is issued.

,TIMELIMIT=*timelimit*
,TIMELIMIT=NO_TIMELIMIT
When REXX=YES and REMOTE=YES are specified, an optional input parameter that is the number of seconds to which the execution of an iteration of the exec is to be limited. A value of 0 is treated the same as "no time limit". TIMELIMIT accepts a value between 0 and 21474536. The default is NO_TIMELIMIT.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,USS=NO
,USS=YES
When REMOTE=NO is specified, an optional parameter that indicates whether the check uses z/OS UNIX System Services. This information is used when z/OS UNIX itself is shut down, at which time IBM Health Checker for z/OS will wait for the completion of the running of any non-remote check that has indicated it uses z/OS UNIX before allowing the z/OS UNIX shutdown to complete. Also, when z/OS UNIX are not available, checks that have indicated they use those services are not run. Thus, indicating "YES" if the check actually does not use z/OS UNIX could delay USS shutdown and would result in the check's not being run when those services are not available. The default is USS=NO.

,USS=NO
indicates the check does not use z/OS UNIX.

,USS=YES
indicates the check does use z/OS UNIX.

,USS=NO
,USS=YES
When REXX=NO and REMOTE=YES are specified, an optional parameter that indicates whether the check uses z/OS UNIX System Services. When z/OS UNIX are not available, checks that have indicated they use those services are not run. Thus, indicating "YES" if the check actually does not use z/OS UNIX would result in the check's not being run when those services are not available. The default is USS=NO.

,USS=NO
indicates the check does not use z/OS UNIX.

,USS=YES
indicates the check does use z/OS UNIX.

,USS=NO

,USS=YES
When REXXTSO=YES, REXX=YES and REMOTE=YES are specified, an optional parameter that indicates whether the check uses z/OS UNIX System Services. When z/OS UNIX are not available, checks that have indicated they use those services are not run. Thus, indicating "YES" if the check actually does not use z/OS UNIX would result in the check's not being run when those services are not available. The default is USS=NO.

,USS=NO
indicates the check does not use z/OS UNIX.

,USS=YES
indicates the check does use z/OS UNIX.

,VERBOSE=NO
,VERBOSE=YES
An optional parameter that identifies the initial verbose mode for the check. The default is VERBOSE=NO.

,VERBOSE=NO
indicates that verbose mode is not to be in effect.

,VERBOSE=YES
indicates that verbose mode is to be in effect.

ABEND Codes

058 The IBM Health Checker for z/OS address space terminated while this call was in process.

290 The HZSADDCK service failed.

The format for reason codes is xxxxyyyy where yyyy is the reason code. The reason codes are in hexadecimal.

Reason Code (Hex)
Explanation

xxxx4007
HZSADDCK could not load the specified check routine.

xxxx4008
HZSADDCK could not load the specified message table.

xxxx4009
HZSADDCK found a message table containing functions that are not supported on this release or the message table was not created by HZSMSGEN.

Return and Reason Codes

When the HZSADDCK macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HZSZCONS provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 38. Return and Reason Codes for the HZSADDCK Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HzsaddckRc_OK **Meaning**: The check was added to IBM Health Checker for z/OS. **Action**: None required
4	—	**Equate Symbol**: HzsaddckRc_Warn **Meaning**: Warning **Action**: Refer to action under the individual reason code.
4	xxxx0401	**Equate Symbol**: HzsaddckRsn_CheckReplaced **Meaning**: The check replaced an active check that had an earlier date. **Action**: None required.
4	xxxx0402	**Equate Symbol**: HzsaddckRsn_CheckInactive **Meaning**: The check was added but will not run until its state is changed to active. **Action**: None required
4	xxxx0414	**Equate Symbol**: HzsaddckRsn_CheckIdentical **Meaning**: Check was not activated because a check with the specified name is already active. **Action**: None required
8	—	**Equate Symbol**: HzsaddckRc_InvParm **Meaning**: HZSADDCK request specifies incorrect parameters. **Action**: Refer to action under the individual reason code.
8	xxxx0801	**Equate Symbol**: HzsaddckRsn_CheckOld **Meaning**: The check was not added because a check with the same name is already being added. That other check has a more recent date than the date provided for this request. **Action**: Avoid adding the same check twice, or make sure that the single version of the check that you want to run has the most current date.
8	xxxx0804	**Equate Symbol**: HzsaddckRsn_BadCheckRoutine **Meaning**: This reason code is not part of the programming interface. **Action**: None.
8	xxxx0805	**Equate Symbol**: HzsaddckRsn_BadMessageTable **Meaning**: This reason code is not part of the programming interface. **Action**: None.

Table 38. Return and Reason Codes for the HZSADDCK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0808	**Equate Symbol**: HzsaddckRsn_BadENV **Meaning**: HZSADDCK for a REMOTE=NO, or a REXX=YES check must be called only from an exit routine associated with the HZSADDCHECK exit. **Action**: Issue HZSADDCK only from a supported environment.
8	xxxx0809	**Equate Symbol**: HzsaddckRsn_BadCheckName **Meaning**: The check name contained invalid characters. **Action**: Specify a valid check name.
8	xxxx080A	**Equate Symbol**: HzsaddckRsn_BadOwnerName **Meaning**: The check owner contained invalid characters. **Action**: Specify a valid check owner.
8	xxxx080B	**Equate Symbol**: HzsaddckRsn_BadDate **Meaning**: The date was not in the format YYYYMMDD or is after today's date. **Action**: Specify a valid date.
8	xxxx080C	**Equate Symbol**: HzsaddckRsn_BadReasonLen **Meaning**: The REASONLEN value is either 0 or exceeds the maximum of 256. **Action**: Specify a valid value for the REASONLEN parameter.
8	xxxx080D	**Equate Symbol**: HzsaddckRsn_BadExitRoutine **Meaning**: The exit routine name was all zeroes or all blanks. **Action**: Specify a valid exit routine.
8	xxxx080E	**Equate Symbol**: HzsaddckRsn_BadTime **Meaning**: The hours value exceeded 999 or the minutes value exceeded 60. **Action**: Specify valid hours and minutes values.
8	xxxx0818	**Equate Symbol**: HzsaddckRsn_BadParmlist **Meaning**: Error accessing parameter list. **Action**: Make sure that the provided parameter list is valid.
8	xxxx0838	**Equate Symbol**: HzsaddckRsn_BadParmListVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Health Checker for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Health Checker for z/OS.
8	xxxx0841	**Equate Symbol**: HzsaddckRsn_BadParmsArea **Meaning**: Error accessing the PARMS area. **Action**: Make sure that the provided PARMS area is valid.

Table 38. Return and Reason Codes for the HZSADDCK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0842	**Equate Symbol**: HzsaddckRsn_BadReasonArea **Meaning**: Error accessing the REASON area. **Action**: Make sure that the provided REASON area is valid.
8	xxxx084F	**Equate Symbol**: HzsaddckRsn_BadParmsLen **Meaning**: The PARMSLEN value is either 0 or exceeds the maximum of 256. **Action**: Specify a valid value for the PARMSLEN parameter.
8	xxxx0859	**Equate Symbol**: HzsaddckRsn_NotAuthorized **Meaning**: Caller is not authorized **Action**: Avoid calling HZSADDCK when not authorized.
8	xxxx0862	**Equate Symbol**: HzsaddckRsn_BadExceptionInterval **Meaning**: The EIHOURS value exceeded 999 or the EIMINUTES value exceeded 60. **Action**: Specify valid hours and minutes values.
8	xxxx0863	**Equate Symbol**: HzsaddckRsn_BadPEToken **Meaning**: The PEToken is not one obtained using authlvl of IEA_UNAUTHORIZED. **Action**: Specify a valid PEToken.
8	xxxx086A	**Equate Symbol**: HzsaddckRsn_BadPETokenHome **Meaning**: The PEToken is not one obtained in the HOME address space. **Action**: Action: Specify a valid PEToken.
8	xxxx086B	**Equate Symbol**: HzsaddckRsn_BadPETokenState **Meaning**: The PEToken is not in a state ready to be used for a PAUSE. **Action**: Specify a valid PEToken.
8	xxxx086C	**Equate Symbol**: HzsaddckRsn_BadPETokenValue **Meaning**: The PEToken appears corrupted. **Action**: Specify a valid PEToken.
0C	—	**Equate Symbol**: HzsaddckRc_EnvError **Meaning**: Environmental Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: HzsaddckRsn_IBMHCNotActive **Meaning**: IBM Health Checker for z/OS is not active **Action**: Re-issue the request when the service is available

Table 38. Return and Reason Codes for the HZSADDCK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
10	—	**Equate Symbol**: HzsaddckRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.
10	xxxx1001	**Equate Symbol**: HzsaddckRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer
16	xxxx106D	**Equate Symbol**: HzsaddckRsn_BadPETokenService **Meaning**: Unexpected error. **Action**: Ensure a valid PEToken has been specified. If this error repeats, contact IBM Support.

Example 1: Add a low severity check that is to run once. The check shares a check routine with other checks, so provides an entry code

The code is as follows.

```
************************************************************
* Add a low severity check that is to run once.            *
************************************************************
         HZSADDCK CHECKOWNER=LOWNER,CHECKNAME=LNAME,         *
               CHECKROUTINE=LCHECKRTN,EXITRTN=LEXITRTN,      *
               MSGTBL=LMSGTBL,DATE=LDATE,                    *
               REASON=LREASON,REASONLEN=LREASONLEN,          *
               ENTRYCODE=LENTRYCODE,                         *
               SEVERITY=LOW,INTERVAL=ONETIME,                *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,ADDCKL)
*
* Place code to check return/reason codes here
*
LOWNER     DC   CL16'MYCOMPANY'
LNAME      DC   CL32'MYCOMPONENT_CHECK_WIDGETS'
LCHECKRTN  DC   CL8'MYMODULE'
LEXITRTN   DC   CL8'MYEXITRT'
LMSGTBL    DC   CL8'MYMSGTBL'
LDATE      DC   CL8'20050601'
LREASON    DC   CL26'Verify widgets are present'
LREASONLEN DC   A(L'LREASON)
LENTRYCODE DC   F'1'
         HZSZCONS              Return code information
DYNAREA    DSECT
LRETCODE   DS   F
LRSNCODE   DS   F
         HZSADDCK MF=(L,ADDCKL),PLISTVER=MAX
```

Example 2: Add a high severity check that is to run once every one hour and fifteen minutes.

The check shares a check routine with other checks, so provides an entry code.

The code is as follows.

```
************************************************************
* Add a high severity check that is to run once             *
************************************************************
HZSADDCK CHECKOWNER=LOWNER,CHECKNAME=LNAME,         * CHECKROUTINE=LC
LENTRYCODE DC   F'2'
LHOURS     DC   H'1'
LMINUTES   DC   H'15'
         HZSZCONS              Return code information
```

```
DYNAREA  DSECT
LRETCODE   DS   F
LRSNCODE   DS   F
         HZSADDCK MF=(L,ADDCKL),PLISTVER=MAX
```

HZSCHECK macro — HZS Check command request

Description

The HZSCHECK macro provides the interface to manage checks that are currently registered with IBM Health Checker For z/OS.

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	Problem state. PSW key 8-15 When problem state and key 8-15 and not APF authorized, or when SECCHECK=ALL is specified, the caller's authorization requirements depend on the input specification. • The caller must be authorized for access to any of the following: – when the check owner has wildcard character(s), XFACILIT class resource HZS.sysname.reqtype – when the check owner has no wildcard characters and the check name has wildcard character(s), XFACILIT class resource HZS.sysname.checkowner.reqtype – when the check owner has no wildcard characters and the check name has no wildcard characters, XFACILIT class resource HZS.sysname.checkowner.checkname.reqtype or XFACILIT class resource HZS.sysname.checkowner.reqtype • The values for reqtype are as follows: – When REQUEST=ACTIVATE is specified, reqtype is ACTIVATE and update authority is needed. – When REQUEST=UPDATE is specified, reqtype is UPDATE and update authority is needed. – When REQUEST=DELETE is specified, reqtype is DELETE and control authority is needed. – When REQUEST=DEACTIVATE is specified, reqtype is DEACTIVATE and update authority is needed. – When REQUEST=REFRESH is specified, reqtype is REFRESH and control authority is needed. – When REQUEST=ADDNEW is specified, reqtype is ADDNEW and update authority is needed. – When REQUEST=RUN is specified, reqtype is RUN and authority for update is needed. – When REQUEST=OPSTART is specified, the authority to have added the check is all that is needed. – When REQUEST=OPCOMPLETE is specified, the authority to have added the check is all that is needed.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.

Requirement	Description
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space. Control parameters must be below 2G.

Programming Requirements

The caller should include the HZSZCONS macro to get equate symbols for the return and reason codes.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSCHECK macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSCHECK macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code, when register 15 is not 0.
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as work registers by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

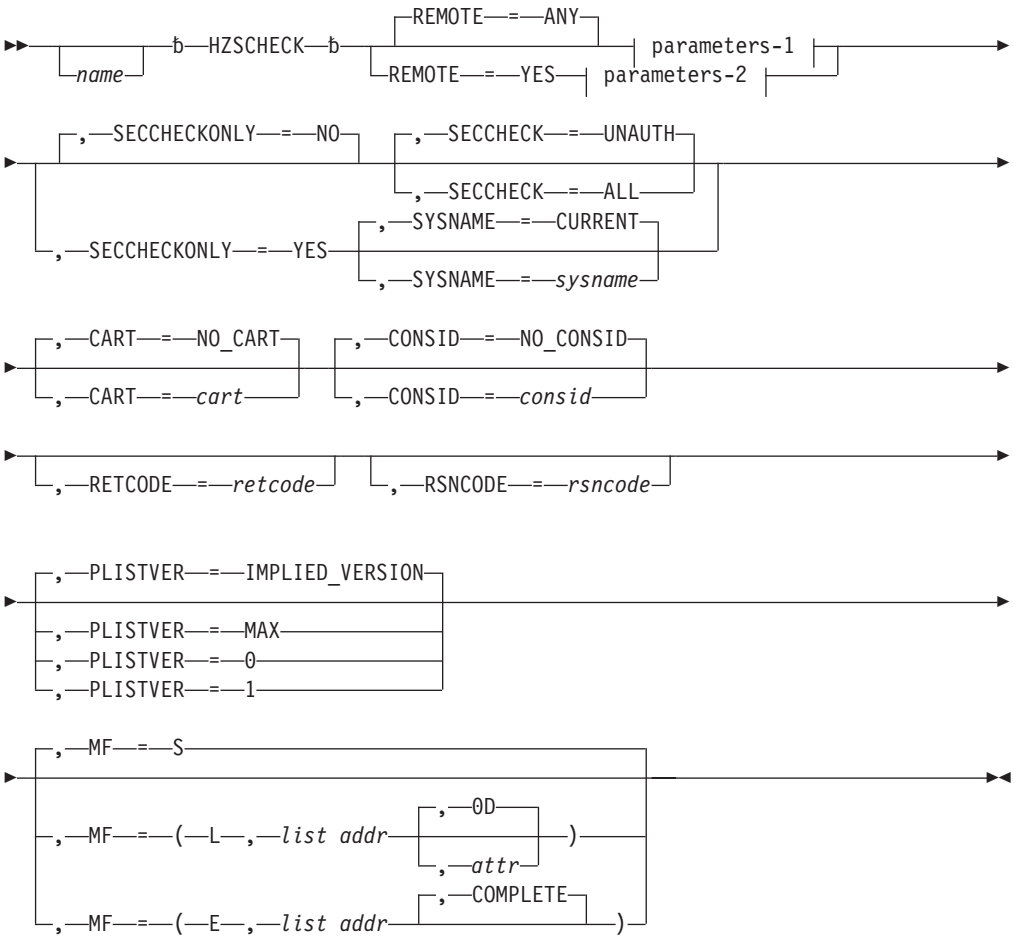
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

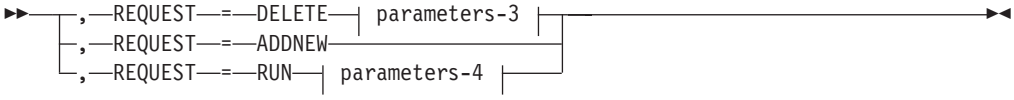
None.

Syntax

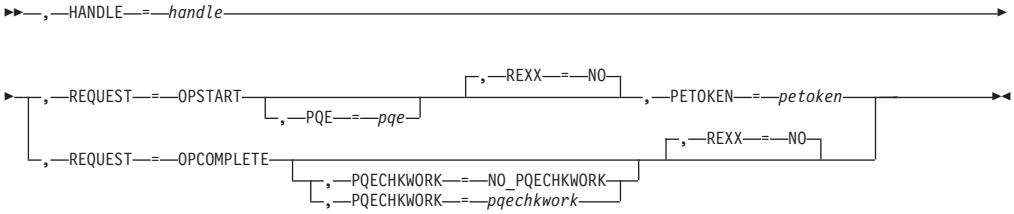
main diagram



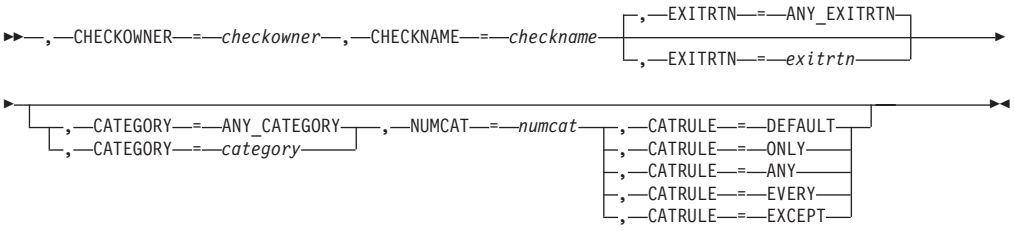
parameters-1



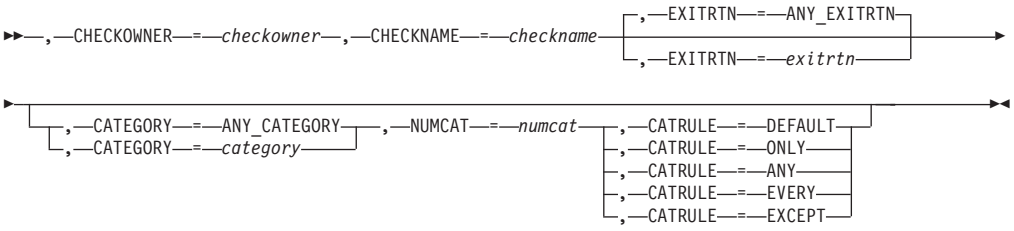
parameters-2



parameters-3



parameters-4



Parameters

The parameters are explained as follows:

name
: An optional symbol, starting in column 1, that is the name on the HZSCHECK macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,CART=*cart*
,CART=NO_CART
: An optional input parameter that specifies the Command And Response Token (CART) to be used if any messages are issued while processing the HZSCHECK request. The default is NO_CART. which indicates that messages issued while processing the HZSCHECK will be issued without a CART.

  To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,CATEGORY=*category*
,CATEGORY=ANY_CATEGORY
: When REQUEST=DELETE and REMOTE=ANY are specified, an optional input parameter that specifies an array of 1 to 16 contiguous 16 character entries each of which contains a category. The categories are used as filters. Each category can include wildcard characters. Checks that belong to categories that match according to the rules of the CATRULE parameter and according to the other filtering parameters (CHECKOWNER, CHECKNAME, and EXITRTN) are processed. The number of categories is specified by the NUMCAT parameter. The default is ANY_CATEGORY.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,CATEGORY=*category*
,CATEGORY=ANY_CATEGORY
: When REQUEST=RUN and REMOTE=ANY are specified, an optional input parameter that specifies an array of 1 to 16 contiguous 16 character entries

each of which contains a category. The categories are used as filters. Each category can include wildcard characters. Checks that belong to categories that match according to the rules of the CATRULE parameter and according to the other filtering parameters (CHECKOWNER, CHECKNAME, and EXITRTN) are processed. The number of categories is specified by the NUMCAT parameter. The default is ANY_CATEGORY.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,CATRULE=DEFAULT
,CATRULE=ONLY
,CATRULE=ANY
,CATRULE=EVERY
,CATRULE=EXCEPT

When CATEGORY=*category*, REQUEST=DELETE and REMOTE=ANY are specified, a required parameter that indicates how to process the category filters.

,CATRULE=DEFAULT
indicates to apply the default (which is CATRULE=ONLY).

,CATRULE=ONLY
indicates to match only if all the categories match the categories to which the target check belongs, and if the target check belongs to exactly the number of categories specified by the NUMCAT parameter.

,CATRULE=ANY
indicates to match if any of the categories provided match any of the categories to which the target check belongs.

,CATRULE=EVERY
indicates to match if every one of the categories provided matches any of the categories to which the target check belongs.

,CATRULE=EXCEPT
indicates to match except when one of the categories provided matches any of the categories to which the target check belongs.

,CATRULE=DEFAULT
,CATRULE=ONLY
,CATRULE=ANY
,CATRULE=EVERY
,CATRULE=EXCEPT

When CATEGORY=*category*, REQUEST=RUN and REMOTE=ANY are specified, a required parameter that indicates how to process the category filters.

,CATRULE=DEFAULT
indicates to apply the default (which is CATRULE=ONLY).

,CATRULE=ONLY
indicates to match only if all the categories match the categories to which the target check belongs, and if the target check belongs to exactly the number of categories specified by the NUMCAT parameter.

,CATRULE=ANY
indicates to match if any of the categories provided match any of the categories to which the target check belongs.

,CATRULE=EVERY
indicates to match if every one of the categories provided matches any of the categories to which the target check belongs.

,CATRULE=EXCEPT
indicates to match except when one of the categories provided matches any of the categories to which the target check belongs.

,CHECKNAME=*checkname*
When REQUEST=DELETE and REMOTE=ANY are specified, a required input parameter that specifies the name of the check to be used as a filter. CHECKNAME can include wildcard characters. All checks with names that match the specified name and that match the other filtering parameters (CHECKOWNER, EXITRTN, CATEGORY) are processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,CHECKNAME=*checkname*
When REQUEST=RUN and REMOTE=ANY are specified, a required input parameter that specifies the name of the check to be used as a filter. CHECKNAME can include wildcard characters. All checks with names that match the specified name and that match the other filtering parameters (CHECKOWNER, EXITRTN, CATEGORY) are processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,CHECKOWNER=*checkowner*
When REQUEST=DELETE and REMOTE=ANY are specified, a required input parameter that specifies the owner of the check to be used as a filter. CHECKOWNER can include wildcard characters. All checks with owners that match the specified owner and that match the other filtering parameters (CHECKNAME, EXITRTN, CATEGORY) are processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CHECKOWNER=*checkowner*
When REQUEST=RUN and REMOTE=ANY are specified, a required input parameter that specifies the owner of the check to be used as a filter. CHECKOWNER can include wildcard characters. All checks with owners that match the specified owner and that match the other filtering parameters (CHECKNAME, EXITRTN, CATEGORY) are processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CONSID=*consid*
,CONSID=NO_CONSID
An optional input parameter that specifies the console ID to be used if any messages are issued while processing the HZSCHECK request. The default is NO_CONSID. If the CONSID parameter is not specified, no messages will be issued while processing the HZSCHECK.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,EXITRTN=*exitrtn*
,EXITRTN=ANY_EXITRTN
When REQUEST=DELETE and REMOTE=ANY are specified, an optional input parameter that specifies the name of the exit routine that was provided via the

EXITRTN parameter on the HZSADDCK macro that added the check. The exit routine is EXITRTN can include wildcard characters. All checks with names that match the specified name and that match the other filtering parameters (CHECKOWNER, CHECKNAME, CATEGORY) are processed. The default is ANY_EXITRTN.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,EXITRTN=*exitrtn*
,EXITRTN=ANY_EXITRTN
When REQUEST=RUN and REMOTE=ANY are specified, an optional input parameter that specifies the name of the exit routine that was provided via the EXITRTN parameter on the HZSADDCK macro that added the check. The exit routine is EXITRTN can include wildcard characters. All checks with names that match the specified name and that match the other filtering parameters (CHECKOWNER, CHECKNAME, CATEGORY) are processed. The default is ANY_EXITRTN.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,HANDLE=*handle*
When REMOTE=YES is specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. The handle is in REXX variable hzs_handle for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,MF=S
,MF=(L,*list addr***)**
,MF=(L,*list addr***,***attr***)**
,MF=(L,*list addr***,0D)**
,MF=(E,*list addr***)**
,MF=(E,*list addr***,COMPLETE)**
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

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

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NUMCAT=*numcat*
When CATEGORY=*category*, REQUEST=DELETE and REMOTE=ANY are specified, a required input parameter that specifies the number of categories contained in the array specified by the CATEGORY parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,NUMCAT=*numcat*
When CATEGORY=*category*, REQUEST=RUN and REMOTE=ANY are specified, a required input parameter that specifies the number of categories contained in the array specified by the CATEGORY parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,PETOKEN=*petoken*
When REXX=NO, REQUEST=OPSTART and REMOTE=YES are specified, a required input parameter that is the updated pause element token returned by the IEAVPSE service (the pause element token was originally obtained via the IEAVAPE service and then was provided as input to the IEAVPSE service which returned an updated token). The caller, waiting to be told what to do by IBM Health Checker for z/OS, should pause using this pause element token. IBM Health Checker for z/OS will "release" using that pause element token to wake up the check processing.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are

assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, which supports all parameters except those specifically referenced in higher versions.
- **1**, which supports both the following parameters and those from version 0:
  - REXXTIMELIM

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, or 1

,PQE=*pqe*

When REQUEST=OPSTART and REMOTE=YES are specified, an optional output parameter that specifies the area into which to place the information mapped by HZSPQE that is associated with this check. This area should begin on a doubleword boundary.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4096-character field.

,PQECHKWORK=*pqechkwork*
,PQECHKWORK=NO_PQECHKWORK

When REQUEST=OPCOMPLETE and REMOTE=YES are specified, an optional input parameter that specifies the PQECHKWORK area which is to be saved and which is to be provided on the next running of the check. This area should begin on a doubleword boundary. The default is NO_PQECHKWORK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2048-character field.

REMOTE=ANY
REMOTE=YES

An optional parameter, which identifies the locale of the check. The default is REMOTE=ANY.

REMOTE=ANY
indicates that the check may either be Remote (runs remotely, in an address space other than that of IBM Health Checker for z/OS) or not Remote (runs locally in the address space of IBM Health Checker for z/OS).

REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,REQUEST=DELETE
,REQUEST=ADDNEW
,REQUEST=RUN

When REMOTE=ANY is specified, a required parameter, which identifies the type of request.

,REQUEST=DELETE
indicates to delete the specified check(s) from IBM Health Checker for z/OS.

,REQUEST=ADDNEW
indicates to call the HZSADDCHECK dynamic exit, which results in running exit routines associated with that exit to add checks that are not

currently added to IBM Health Checker for z/OS. When a check is added, the current policy is processed to obtain any modifications to the new check(s).

The system runs checks when they are added, unless they are inactive.

REQUEST(ADDNEW) is not allowed from within a HZSADDCHECK dynamic exit routine.

,REQUEST=RUN
indicates to run the specified check(s) registered with IBM Health Checker for z/OS. Checks that are inactive will not be run.

,REQUEST=OPSTART
,REQUEST=OPCOMPLETE
When REMOTE=YES is specified, a required parameter, which identifies the type of request.

,REQUEST=OPSTART
indicates that the current operation is starting

,REQUEST=OPCOMPLETE
indicates that the current operation is now complete for the check.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,REXX=NO
When REQUEST=OPSTART and REMOTE=YES are specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is a REXX check.

,REXX=NO
When REQUEST=OPCOMPLETE and REMOTE=YES are specified, an optional parameter, which indicates if the check is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is a REXX check.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SECCHECK=UNAUTH
,SECCHECK=ALL
When SECCHECKONLY=NO is specified, an optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases. If RACF does not grant authority, the request is rejected.

,SECCHECKONLY=NO
,SECCHECKONLY=YES
An optional parameter that indicates whether to do full processing or only security checks. The default is SECCHECKONLY=NO.

,SECCHECKONLY=NO
that indicates to do full processing.

,SECCHECKONLY=YES
that indicates to do only the security check to see if the requesting user has RACF authority to access the data. When this option is specified, the security check is done regardless of the caller's key or state.

,SYSNAME=*sysname*
,SYSNAME=CURRENT
When SECCHECKONLY=YES is specified, an optional input parameter that specifies the system name to be used when doing the security check. Note that this specification is used only when the caller is supervisor state, system key, or APF-authorized. The default is CURRENT. which indicates to use the name of the system on which this request was issued.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

ABEND Codes

058 The IBM Health Checker for z/OS address space terminated while this call was in process.

Return and Reason Codes

When the HZSCHECK macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HZSZCONS provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 39. Return and Reason Codes for the HZSCHECK Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HzscheckRc_OK **Meaning**: SECCHECKONLY=YES was requested and the request passed the security check. **Action**: None required.
4	—	**Equate Symbol**: HzscheckRc_Warn **Meaning**: Warning **Action**: Refer to action under the individual reason code.

Table 39. Return and Reason Codes for the HZSCHECK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
4	xxxx0400	**Equate Symbol**: HzscheckRsn_CommandQueued **Meaning**: The specified HZSCHECK will be completed asynchronously **Action**: None needed
8	—	**Equate Symbol**: HzscheckRc_InvParm **Meaning**: HZSCHECK request specifies incorrect parameters. **Action**: Refer to action under the individual reason code.
8	xxxx0801	**Equate Symbol**: HzscheckRsn_NotAuthorized **Meaning**: Caller is not authorized **Action**: Avoid calling HZSCHECK when not authorized.
8	xxxx0818	**Equate Symbol**: HzscheckRsn_BadParmlist **Meaning**: Error accessing the parameter list **Action**: Make sure that the provided parameter list is valid.
8	xxxx0829	**Equate Symbol**: HzscheckRsn_BadAddRepcatArea **Meaning**: Error while reading the AddCat or RepCat array **Action**: Make sure that the provided area is valid.
8	xxxx082A	**Equate Symbol**: HzscheckRsn_BadRemcatArea **Meaning**: Error while reading the RemCat array **Action**: Make sure that the provided area is valid.
8	xxxx0838	**Equate Symbol**: HzscheckRsn_BadParmListVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Health Checker for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Health Checker for z/OS.
8	xxxx0847	**Equate Symbol**: HzscheckRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	xxxx084B	**Equate Symbol**: HzscheckRsn_BadParmlistValue **Meaning**: A parameter list field contains an unsupported value. **Action**: Check for possible storage overlay
8	xxxx084C	**Equate Symbol**: HzscheckRsn_BadCategoryALET **Meaning**: Bad category ALET. **Action**: Make sure that the ALET associated with the category area is valid. The access register might not have been set up correctly.

Table 39. Return and Reason Codes for the HZSCHECK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx084D	**Equate Symbol**: HzscheckRsn_BadCategoryArea **Meaning**: Error accessing category area. **Action**: Make sure that the provided category area is valid.
8	xxxx0853	**Equate Symbol**: HzscheckRsn_BadAddRepcatALET **Meaning**: Bad ALET for AddCat or RepCat array. **Action**: Make sure that the ALET associated with the AddCat or RepCat array is valid. The access register might not have been set up correctly.
8	xxxx0854	**Equate Symbol**: HzscheckRsn_BadRemcatALET **Meaning**: Bad ALET for RemCat array. **Action**: Make sure that the ALET associated with the RemCat array is valid. The access register might not have been set up correctly.
8	xxxx0855	**Equate Symbol**: HzscheckRsn_BadNumCat **Meaning**: Value provided by NUMCAT exceeds the limit of 16. **Action**: Avoid specifying more than the allowable number of categories.
8	xxxx0856	**Equate Symbol**: HzscheckRsn_BadNumAddRepRemCat **Meaning**: The total value provided by NUMADDCAT, NUMREPCAT, and NUMREMCAT exceeds the limit of 16. **Action**: Avoid specifying more than the allowable number of categories.
8	xxxx0858	**Equate Symbol**: HzscheckRsn_BadHandle **Meaning**: The handle provided with the HANDLE parameter is not valid. **Action**: Specify the handle that was returned by the HZSADDCK macro if this is a REMOTE=YES REXX=NO check, or the handle in REXX variable hzs_handle if this is a REMOTE=YES REXX=YES check.
8	xxxx0863	**Equate Symbol**: HzscheckRsn_BadPEToken **Meaning**: The PEToken is not one obtained using authlvl of IEA_UNAUTHORIZED. **Action**: Specify a valid PEToken.
8	xxxx0864	**Equate Symbol**: HzscheckRsn_BadPqeArea **Meaning**: Error while writing to the PQE area **Action**: Make sure that the provided area is valid.
8	xxxx0865	**Equate Symbol**: HzscheckRsn_BadPqeALET **Meaning**: Bad ALET for the PQE area. **Action**: Make sure that the ALET associated with the PQE area is valid. The access register might not have been set up correctly.
8	xxxx0866	**Equate Symbol**: HzscheckRsn_BadPqeChkWorkArea **Meaning**: Error while reading from the PqeChkWork area **Action**: Make sure that the provided area is valid.

Table 39. Return and Reason Codes for the HZSCHECK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0867	**Equate Symbol**: HzscheckRsn_BadPqeChkWorkALET **Meaning**: Bad ALET for the PqeChkWork area. **Action**: Make sure that the ALET associated with the PqeChkWork area is valid. The access register might not have been set up correctly.
8	xxxx086A	**Equate Symbol**: HzscheckRsn_BadPETokenHome **Meaning**: The PEToken is not one obtained in the HOME address space. **Action**: Specify a valid PEToken.
8	xxxx086B	**Equate Symbol**: HzscheckRsn_BadPETokenState **Meaning**: The PEToken is not in a state ready to be used for a PAUSE. **Action**: Specify a valid PEToken.
8	xxxx086C	**Equate Symbol**: HzscheckRsn_BadPETokenValue **Meaning**: The PEToken appears corrupted. **Action**: Specify a valid PEToken.
0C	—	**Equate Symbol**: HzscheckRc_EnvError **Meaning**: Environmental Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: HzscheckRsn_IBMHCNotActive **Meaning**: IBM Health Checker for z/OS is not active **Action**: For REQUEST=ADDNEW, no action is needed. For any other REQUEST option, re-issue the request when the service is available
0C	xxxx0C02	**Equate Symbol**: HzscheckRsn_BadCommandEnv **Meaning**: The specified command cannot be specified from a HZSADDCHECK dynamic exit **Action**: Do Not issue a ADDNEW or REFRESH command from a HZSADDCHECK dynamic exit routine
0C	xxxx0C03	**Equate Symbol**: HzscheckRsn_BadRemoteEnv **Meaning**: For REQUEST=OPSTART or REQUEST=OPCOMPLETE, the call must be done only once after having been awakened to process a remote function. For that function, the call may be done only once. For REQUEST=OPSTART, the call must be done before the REQUEST=OPCOMPLETE call. **Action**: Avoid using REQUEST=OPSTART or REQUEST=OPCOMPLETE in an incorrect environment.
10	—	**Equate Symbol**: HzscheckRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.

Table 39. Return and Reason Codes for the HZSCHECK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
10	xxxx1001	**Equate Symbol**: HzscheckRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer
16	xxxx106D	**Equate Symbol**: HzscheckRsn_BadPETokenService **Meaning**: Unexpected error. **Action**: Ensure a valid PEToken has been specified. If this error repeats, contact IBM Support.

Examples

None.

HZSCPARS macro — HZS Check Parameter Parsing

Description

The HZSCPARS macro provides functions dealing with parsing the check parameter string. It is intended to help only with parsing the parameter area for the check (as opposed to some general area) and it is of help only for parameter area syntaxes that follow a simple style, namely a combination of "PositionalValue" or "keyword=value" or "keyword=value1,...,valuen" or "keyword(value1,...,valuen)" items separated from each other by one or more blanks or a comma. If "PositionalValue" is found, all items must be of that positional format. If keyword/value format is found, no values of positional format are allowed. You should avoid extra separating commas, as two consecutive commas indicates a null positional value.

A typical sequence would consist of

- REQUEST=PARSE
- For each known parameter
  - REQUEST=CHECKPARM with further checking as needed using one of
    - REQUEST=CHECKDEC
    - REQUEST=CHECKHEX
    - REQUEST=CHECKCHAR
- REQUEST=CHECKNOTPROC
- REQUEST=FREE

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	For local checks, supervisor state and the key of the check routine. For remote checks, problem state and PSW key 8-15.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

The caller must include the HZSZCPAR macro to get mappings for the areas.

All HZSCPARS services called after the parse service must be called in the same PSW key in which you called the parse service.

Restrictions

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSCPARS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSCPARS macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as work registers by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

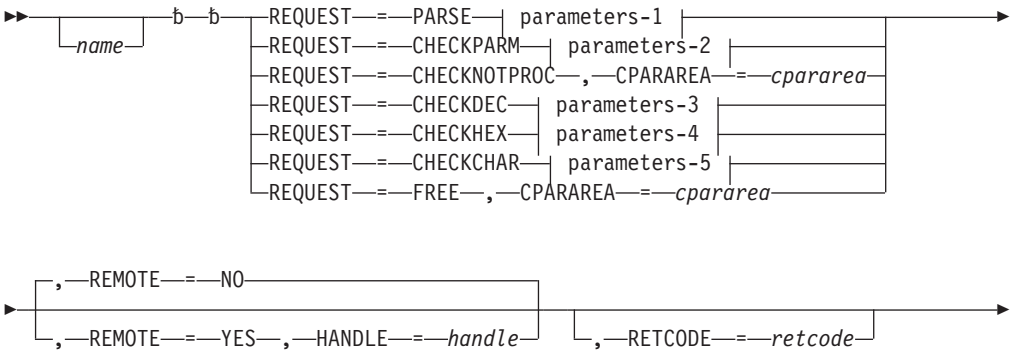
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

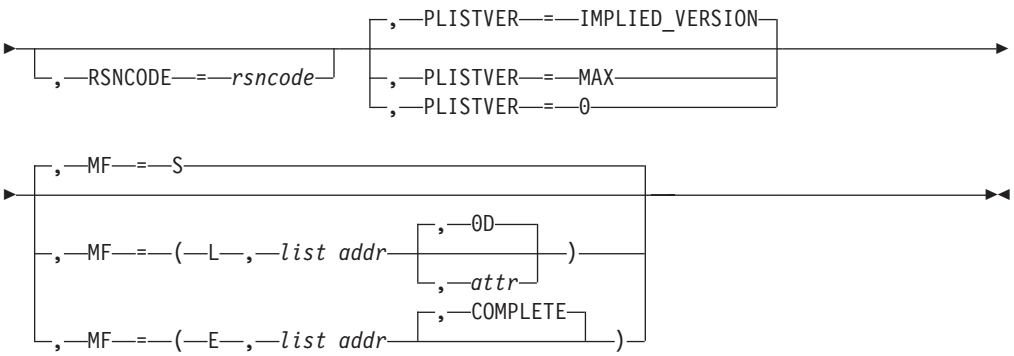
Performance Implications

None.

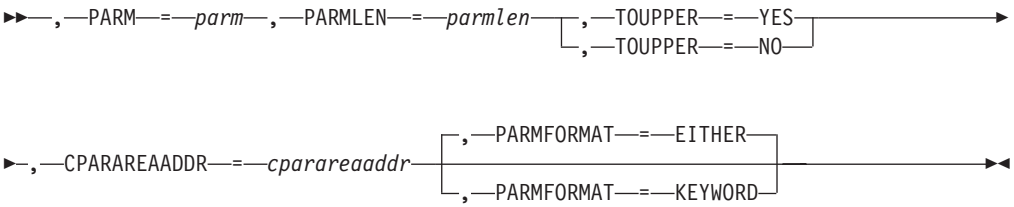
Syntax

main diagram

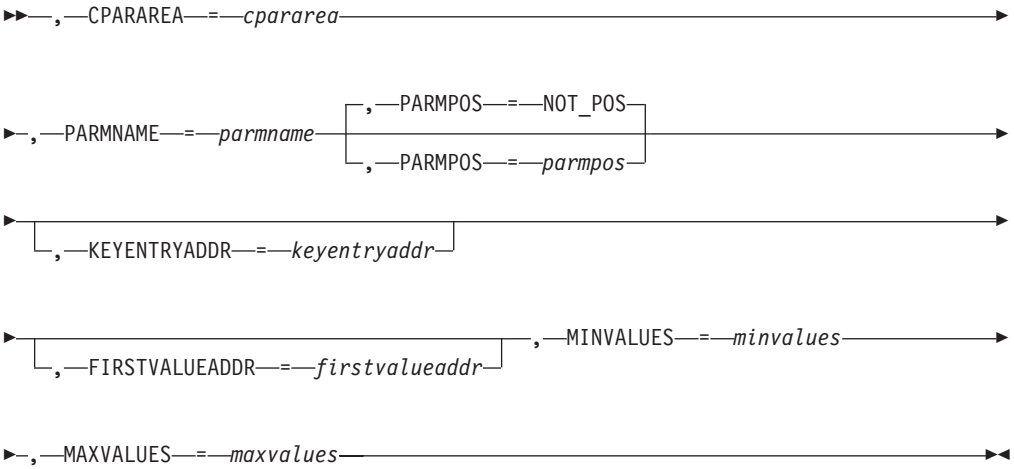




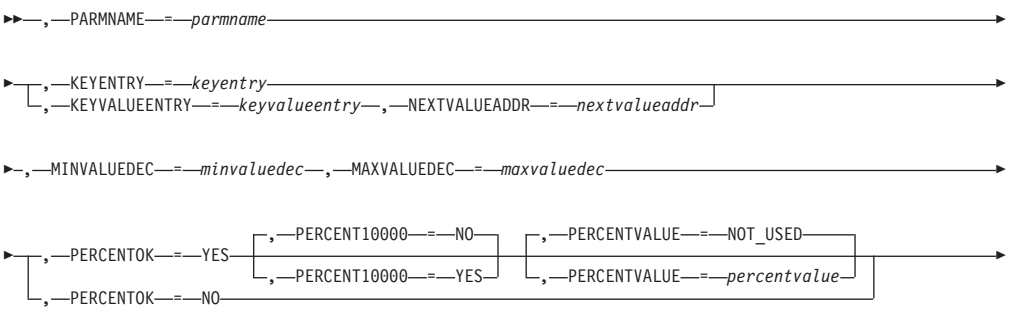
parameters-1



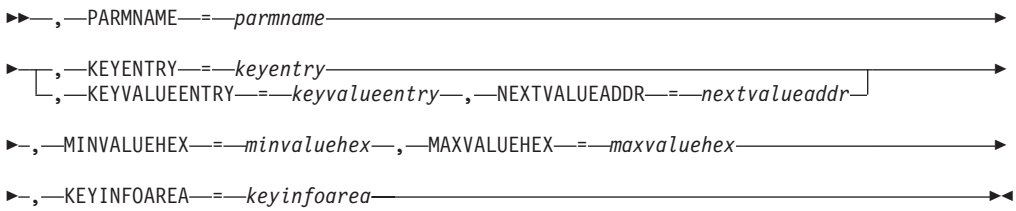
parameters-2



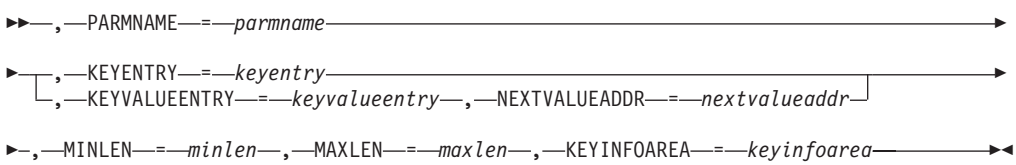
parameters-3



parameters-4



parameters-5



Parameters

The parameters are explained as follows:

name
: An optional symbol, starting in column 1, that is the name on the HZSCPARS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,CPARAREA=*cpararea*
: When REQUEST=CHECKPARM is specified, a required input parameter that is the check parse area (CParArea), the address of which was returned by the PARSE request.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,CPARAREA=*cpararea*
: When REQUEST=CHECKNOTPROC is specified, a required input parameter that is the check parse area (CParArea), the address of which was returned by the PARSE request.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,CPARAREA=*cpararea*
: When REQUEST=FREE is specified, a required input parameter that is the check parse area (CParArea) to be freed.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,CPARAREAADDR=*cpararеaaddr*
: When REQUEST=PARSE is specified, a required output parameter that is to contain the address of the check parse area (CParArea).

  To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,FIRSTVALUEADDR=*firstvalueaddr*
When REQUEST=CHECKPARM is specified, an optional output parameter that is to contain the address of the first CParKeywordValueEntry area of the parameter, or 0 if there are none. A value of 0 is expected when the format is positional (bit CparAreaFormatPositional is on. This should be used except when you are verifying that the number of values that can be specified is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,HANDLE=*handle*
When REMOTE=YES is specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,KEYENTRY=*keyentry*
When REQUEST=CHECKDEC is specified, a required input parameter that is the CParKeywordEntry of the value to be processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYENTRY=*keyentry*
When REQUEST=CHECKHEX is specified, a required input parameter that is the CParKeywordEntry of the value to be processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYENTRY=*keyentry*
When REQUEST=CHECKCHAR is specified, a required input parameter that is the CParKeywordEntry of the value to be processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYENTRYADDR=*keyentryaddr*
When REQUEST=CHECKPARM is specified, an optional output parameter that is to contain the address of the CParKeywordEntry of the parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,KEYINFOAREA=*keyinfoarea*
When REQUEST=CHECKDEC is specified, a required input/output parameter, of the KeywordInfo area that is built. It need not be initialized prior to the call

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYINFOAREA=*keyinfoarea*
When REQUEST=CHECKHEX is specified, a required input/output parameter, of the KeywordInfo area that is built. It need not be initialized prior to the call

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYINFOAREA=*keyinfoarea*
When REQUEST=CHECKCHAR is specified, a required input/output

parameter, of the KeywordInfo area that is built. It need not be initialized prior to the call. At this time, there is no output information within this area that you will need.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYVALUEENTRY=*keyvalueentry*
When REQUEST=CHECKDEC is specified, a required input parameter that is the CParKeywordValueEntry of the value to be processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYVALUEENTRY=*keyvalueentry*
When REQUEST=CHECKHEX is specified, a required input parameter that is the CParKeywordValueEntry of the value to be processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,KEYVALUEENTRY=*keyvalueentry*
When REQUEST=CHECKCHAR is specified, a required input parameter that is the CParKeywordValueEntry of the value to be processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MAXLEN=*maxlen*
When REQUEST=CHECKCHAR is specified, a required input parameter that is the maximum length allowed.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,MAXVALUEDEC=*maxvaluedec*
When REQUEST=CHECKDEC is specified, a required input parameter that is the maximum decimal value allowed. This is a numeric value. When the number has a percent suffix, a value in the range 1-100 is accepted regardless of what is specified with this parameter. The value is treated as an unsigned number, and a value >= 2**63 will be treated as 2**63-1.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,MAXVALUEHEX=*maxvaluehex*
When REQUEST=CHECKHEX is specified, a required input parameter that is the maximum hexadecimal value allowed. This is a numeric value.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,MAXVALUES=*maxvalues*
When REQUEST=CHECKPARM is specified, a required input parameter that indicates the maximum number of values that can be specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,MF=S
,MF=(L,*list addr***)**
,MF=(L,*list addr*,*attr***)**
,MF=(L,*list addr*,**0D)**
,MF=(E,*list addr***)**

,MF=(E,*list addr***,COMPLETE)**
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

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

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MINLEN=*minlen*
When REQUEST=CHECKCHAR is specified, a required input parameter that is the minimum length allowed.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,MINVALUEDEC=*minvaluedec*
When REQUEST=CHECKDEC is specified, a required input parameter that is the minimum decimal value allowed. This is a numeric value. When the number has a percent suffix, a value in the range 1-100 is accepted regardless of what is specified with this parameter. The value is treated as an unsigned number.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,MINVALUEHEX=*minvaluehex*
When REQUEST=CHECKHEX is specified, a required input parameter that is the minimum hexadecimal value allowed. This is a numeric value.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,MINVALUES=*minvalues*
When REQUEST=CHECKPARM is specified, a required input parameter that indicates the minimum number of values that can be specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,NEXTVALUEADDR=*nextvalueaddr*
When KEYVALUEENTRY=*keyvalueentry* and REQUEST=CHECKDEC are specified, a required output parameter that is to contain the address that is to contain the address of the next CParKeywordValueEntry area of the parameter, or 0 if there are none.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,NEXTVALUEADDR=*nextvalueaddr*
When KEYVALUEENTRY=*keyvalueentry* and REQUEST=CHECKHEX are specified, a required output parameter that is to contain the address that is to contain the address of the next CParKeywordValueEntry area of the parameter, or 0 if there are none.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,NEXTVALUEADDR=*nextvalueaddr*
When KEYVALUEENTRY=*keyvalueentry* and REQUEST=CHECKCHAR are specified, a required output parameter that is to contain the address that is to contain the address of the next CParKeywordValueEntry area of the parameter, or 0 if there are none.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,PARM=*parm*
When REQUEST=PARSE is specified, a required input parameter that is the input parameter. The input parameter has a limited acceptable character set:

- Alphanumeric
- Special ('@', '#', '$')
- Additional ('.', '*', '?', '_', '/', '-', '%')
- Delimiters ('=', '(', ')', ',', blank)
- Single quote (within a quoted string, any character is accepted)

A null parameter can be denoted by consecutive commas with no non-blank non-comment text in between (or by a leading comma). For example, the string "A,,B" represents 3 positional parameters, the first being "A", the second being null, and the third being "B".

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,PARMFORMAT=EITHER
,PARMFORMAT=KEYWORD
When REQUEST=PARSE is specified, an optional parameter, which identifies the allowed format of the input. The default is PARMFORMAT=EITHER.

,PARMFORMAT=EITHER
indicates that either positional or keyword format is OK. An example of positional format is ("Val1,...,ValN"). Examples of keyword format are "key1(val1),...,keyN(valN)" and "Key1=val1,...,keyN=valN". Note that within an input string there cannot be a mixture of keyword and positional format.

,PARMFORMAT=KEYWORD
indicates that only keyword format is allowed.

,PARMLEN=*parmlen*
When REQUEST=PARSE is specified, a required input parameter that is the length of the input parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,PARMNAME=*parmname*
When REQUEST=CHECKPARM is specified, a required input parameter that is the name of the parameter to be checked.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PARMNAME=*parmname*
When REQUEST=CHECKDEC is specified, a required input parameter that is the name of the parameter being processed

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PARMNAME=*parmname*
When REQUEST=CHECKHEX is specified, a required input parameter that is the name of the parameter being processed

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PARMNAME=*parmname*
When REQUEST=CHECKCHAR is specified, a required input parameter that is the name of the parameter being processed

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PARMPOS=*parmpos*
,PARMPOS=NOT_POS
When REQUEST=CHECKPARM is specified, an optional input parameter that indicates to return the Nth parameter. The default is NOT_POS. that indicates the parameter is not positional, so use the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,PERCENTOK=YES
,PERCENTOK=NO
When REQUEST=CHECKDEC is specified, a required parameter that indicates if a percent suffix is to be accepted

,PERCENTOK=YES
indicates that a percent suffix is OK.

,PERCENTOK=NO
indicates that a percent suffix is not OK.

,PERCENTVALUE=*percentvalue*
,PERCENTVALUE=NOT_USED
When PERCENTOK=YES and REQUEST=CHECKDEC are specified, an optional input parameter that is the value to which the specified decimal parameter value when it ends with "%" will be applied. This is a numeric value. This value is multipled by the percentage and the result is returned (multiply by N and then divide by 100). The default is NOT_USED.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,PERCENT10000=NO
,PERCENT10000=YES
When PERCENTOK=YES and REQUEST=CHECKDEC are specified, an optional parameter that indicates if the percentage may be up to 10000. The default is PERCENT10000=NO.

,PERCENT10000=NO
indicates that the percentage value has a maximum value of 100.

,PERCENT10000=YES
indicates that the percentage value may be up to 10000.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,REMOTE=NO
,REMOTE=YES
An optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

REQUEST=PARSE
REQUEST=CHECKPARM
REQUEST=CHECKNOTPROC
REQUEST=CHECKDEC

REQUEST=CHECKHEX
REQUEST=CHECKCHAR
REQUEST=FREE
A required parameter, which identifies the type of request.

REQUEST=PARSE
Parse the input string. Note that if the return code from this function is not zero, you should avoid using the other request types, as no valid data will have been returned.

REQUEST=CHECKPARM
Check a particular parameter for number of values

REQUEST=CHECKNOTPROC
Check for parameters that were not processed

REQUEST=CHECKDEC
Check a particular parameter value as a decimal number. The parameter can be a decimal number or a decimal number followed by a suffix of K (multiply by 2**10), M (multiply by 2**20), G (multiply by 2**30), T (multiply by 2**40) P (multiply by 2**50), E (multiply by 2**60). The decimal number is limited to a length of 10 characters and a maximum value of 2147483647. The value that is checked against is the decimal number multiplied by (when a suffix is provided) the value indicated by the suffix.

REQUEST=CHECKHEX
Check a particular parameter value as a hexadecimal number

REQUEST=CHECKCHAR
Check a particular parameter value as character data

REQUEST=FREE
Free the storage area. Note that, for a REMOTE=NO check if you do not use this function, the system will free this area for you upon return from the check routine call in which the area was obtained. Thus for a REMOTE=NO check you must use REQUEST=FREE from the check routine call that issued REQUEST=PARSE only.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,TOUPPER=YES
,TOUPPER=NO
When REQUEST=PARSE is specified, a required parameter, which indicates if the input parameter string is to be translated to upper case before processing.

,TOUPPER=YES
indicates to translate to upper case.

,TOUPPER=NO
indicates not to translate to upper case.

ABEND Codes

290 HZSCPARS service failed a request.

xxx Various abends can occur if an invalid handle or parse area is provided.

An abend 290 will be issued if an error in the request is detected.

In the following HZSCPARS abend reason codes, the bytes designated "xx" are for diagnostic purposes and have no significance to the external interface.

Reason Code (Hex)
Explanation

xxxx002D
A parse area already exists, indicating that parsing has already been done. Use HZSCPARS REQUEST=FREE prior to beginning a new parse.

Return and Reason Codes

When the HZSCPARS macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 40. Return and Reason Codes for the HZSCPARS Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HZSCPARSRc_OK **Meaning**: Requested information returned **Action**: None required
4	—	**Equate Symbol**: HZSCPARSRc_Warn **Meaning**: Warning **Action**: Refer to action under the individual reason code.
4	xxxx0401	**Equate Symbol**: HZSCPARSRsn_NotLocated **Meaning**: For the CHECKPARM request, the parameter was not found. **Action**: None required.
4	xxxx0402	**Equate Symbol**: HZSCPARSRsn_NoParms **Meaning**: For the PARSE request, the input parameter length was 0. **Action**: None required.
8	—	**Equate Symbol**: HZSCPARSRc_InvParm **Meaning**: HZSCPARS request specifies incorrect parameters. **Action**: Refer to action under the individual reason code.

Table 40. Return and Reason Codes for the HZSCPARS Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0801	**Equate Symbol**: HZSCPARSRsn_BadParmLen **Meaning**: The parameter length exceeded the maximum of 4096. **Action**: Specify a valid parameter length.
0C	—	**Equate Symbol**: HZSCPARSRc_EnvError **Meaning**: Environmental Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: HZSCPARSRsn_SyntaxError **Meaning**: A syntax error was detected. A message was issued about the problem. **Action**: Use HZSFMSG REQUEST=STOP,REASON=BADPARM to indicate that the check cannot proceed because of a parameter error.

Examples

None.

HZSFMSG macro — Issue a formatted check message

Description

HZSFMSG is used by a check routine to format and process a check message (using the message table identified by the MSGTBL parameter of the HZSADDCK macro) or to report a functional issue such as a parameter error.

Both check-defined and system-defined messages can be issued.

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	Problem state. PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space. Control parameters must be below 2G.

Programming Requirements

This service is supported only when it is called from a check routine invoked by IBM Health Checker for z/OS.

The check routine must include macro HZSMGB to get a mapping of the MGB which is input to HZSFMSG.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSFMSG macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSFMSG macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0 Reason code, when register 15 is not 0.

0-1 Used as work registers by the system

2-13 Unchanged

14 Used as work registers by the system

15 Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1 Used as work registers by the system

2-13 Unchanged

14-15 Used as work registers by the system

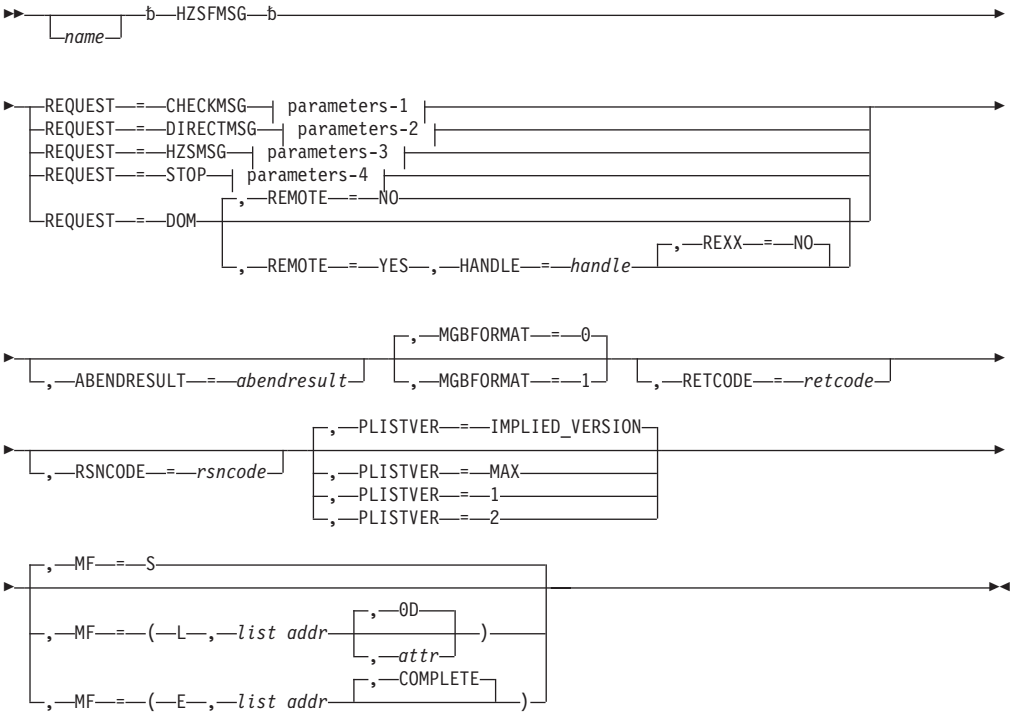
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

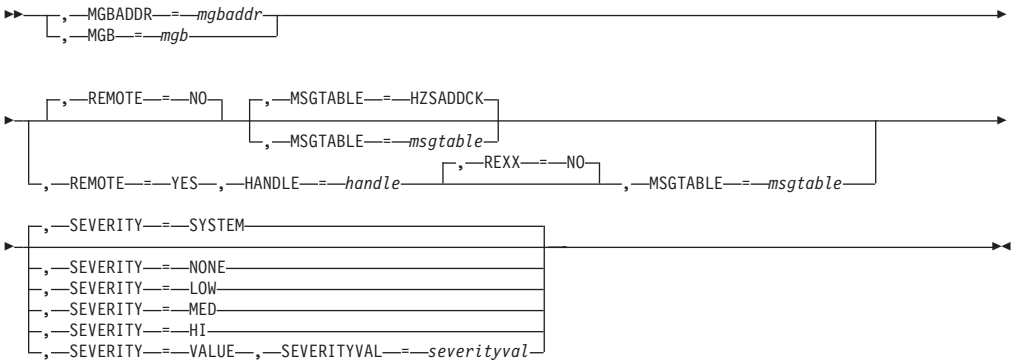
None.

Syntax

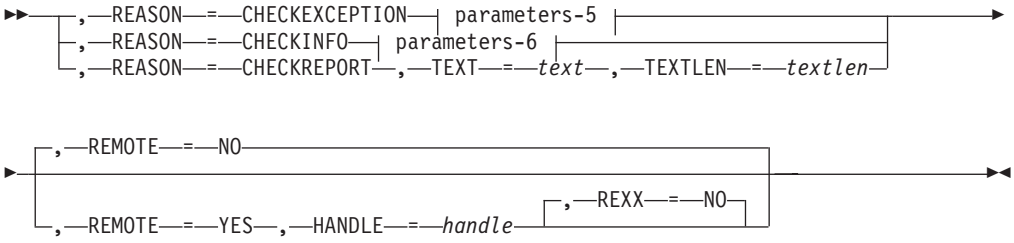
main diagram



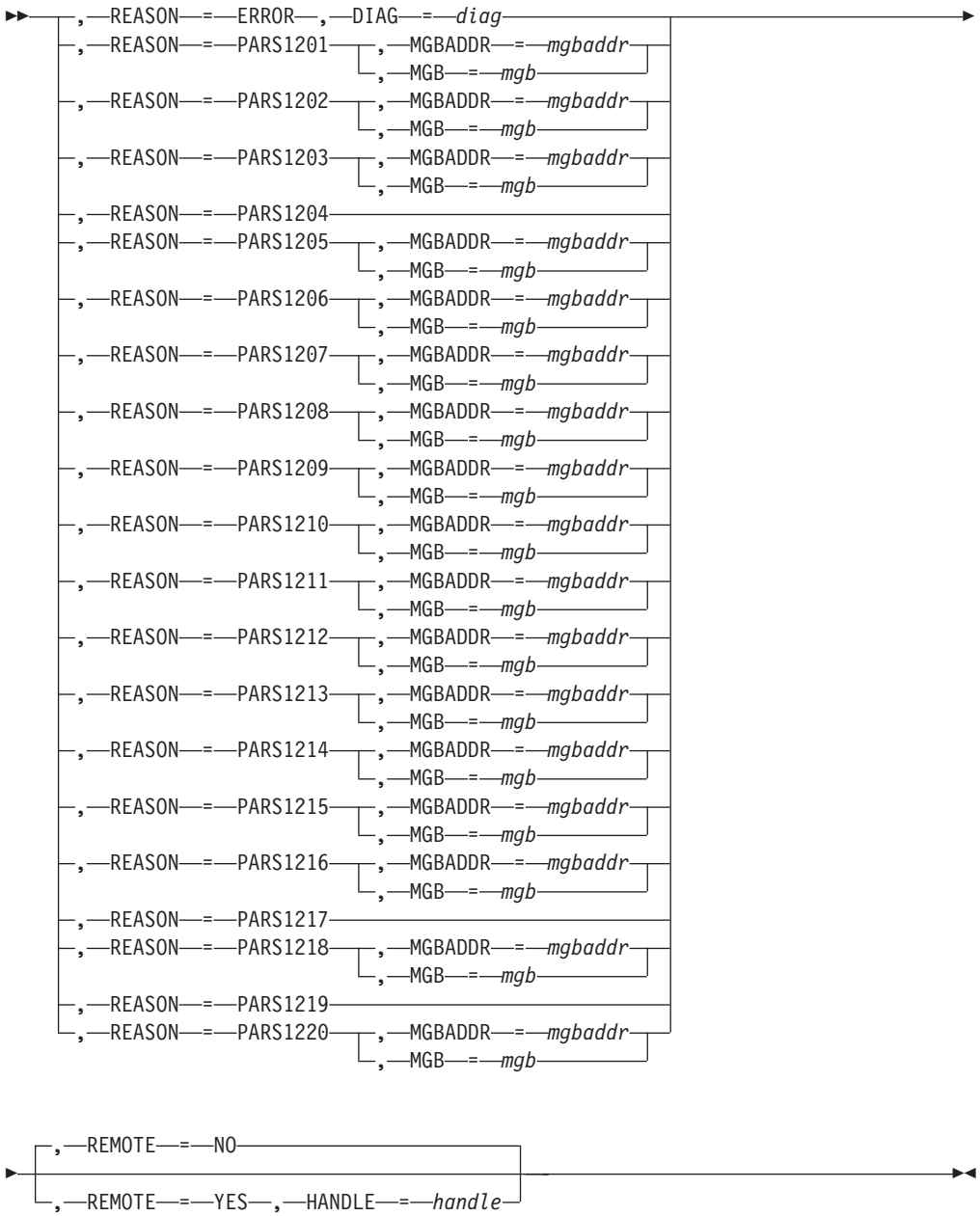
parameters-1



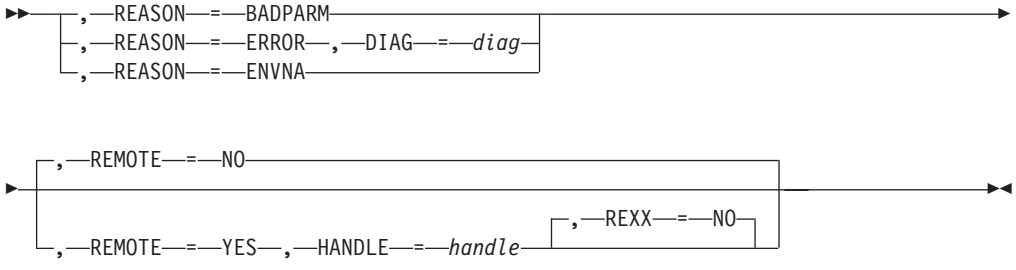
parameters-2



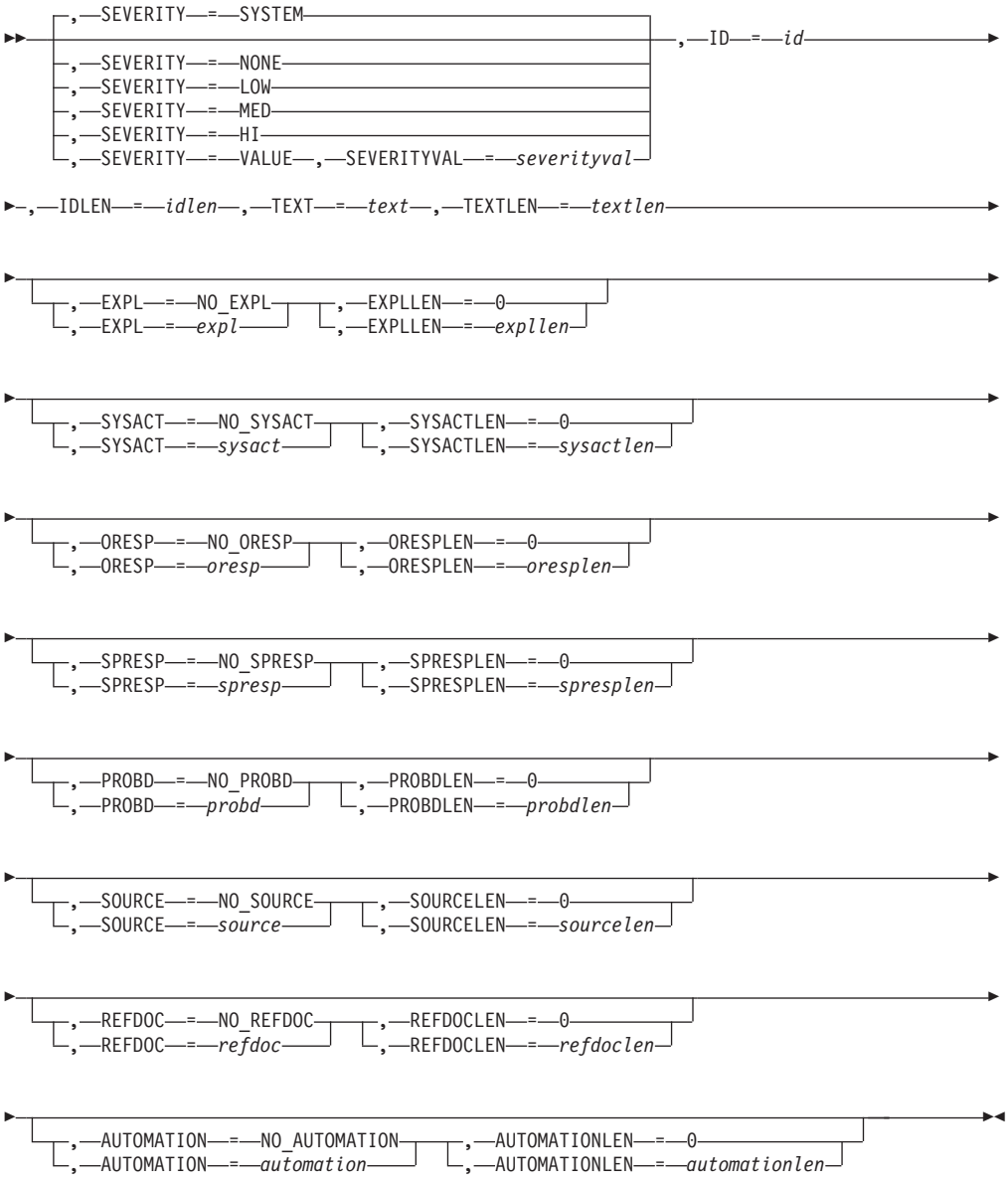
parameters-3



parameters-4



parameters-5



parameters-6

Parameters

The parameters are explained as follows:

name

An optional symbol, starting in column 1, that is the name on the HZSFMSG macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ABENDRESULT=*abendresult*

An optional output parameter, which is to contain diagnostic information about this invocation, when the result is Abend s290. The information is in "readable" EBCDIC.

Macro action

Result value returned

RC=0 OK

RC>0 HZSFMSG RC=rc RSN=rsn

ABEND

HZSFMSG ABEND 290/rsn Message=msgnum Insert=insertnum MsgTbleOffset=offset MsgSegmentData='hex data'X AbendData1=datafield1 .... AbendDatan=datafieldn

To code: Specify the RS-type address, or address in register (2)-(12), of a 256-character field.

,AUTOMATION=*automation*

,AUTOMATION=NO_AUTOMATION

When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the automation information for the message request. The length of the automation information text is specified by the AUTOMATIONLEN parameter. The default is NO_AUTOMATION.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,AUTOMATIONLEN=*automationlen*

,AUTOMATIONLEN=0

When AUTOMATION=*automation*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the automation information text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *automationlen* must be in the range 1 through 65535.

,DIAG=*diag*

When REASON=ERROR and REQUEST=HZSMSG are specified, a required input parameter, which is displayed as hex data in message output to provide diagnostic information for the failure that is being reported. There is no pre-defined format for this data; it may well be internal component diagnostic data.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,DIAG=*diag*

When REASON=ERROR and REQUEST=STOP are specified, a required input parameter, which is displayed as hex data in message output to provide diagnostic information for the failure that is being reported. There is no pre-defined format for this data; it may well be internal component diagnostic data.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,EXPL=*expl*

,EXPL=NO_EXPL
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the explanation text for the message request. The length of the explanation text is specified by the EXPLLEN parameter. The default is NO_EXPL.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,EXPLLEN=*expllen*
,EXPLLEN=0
When EXPL=*expl*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the explanation text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *expllen* must be in the range 1 through 65535.

,HANDLE=*handle*
When REMOTE=YES and REQUEST=CHECKMSG are specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. The handle is in REXX variable hzs_handle for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,HANDLE=*handle*
When REMOTE=YES and REQUEST=DIRECTMSG are specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. The handle is in REXX variable hzs_handle for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,HANDLE=*handle*
When REMOTE=YES and REQUEST=HZSMSG are specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. The handle is in REXX variable hzs_handle for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,HANDLE=*handle*
When REMOTE=YES and REQUEST=STOP are specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. The handle is in REXX variable hzs_handle for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,HANDLE=*handle*
When REMOTE=YES and REQUEST=DOM are specified, a required input parameter that specifies a handle (token) that identifies the check. This handle

was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. The handle is in REXX variable hzs_handle for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,ID=*id*

When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that is an up to 10 character text value that appears as the message identifier. The length of the message identifier is specified by the IDLEN parameter. The recommended message identifier format is ccccHmmmms:

- cccc is the component identifier, such as ISG for global resource serialization.
- the required H represents IBM Health Checker for z/OS
- mmmm is the message number. For example, in message identifier ISGH101E, 101 is the message number
- s is the severity code for the message: E for exception messages and I for information messages

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ID=*id*

When REASON=CHECKINFO and REQUEST=DIRECTMSG are specified, a required input parameter that is an up to 10 character text value that appears as the message identifier. The length of the message identifier is specified by the IDLEN parameter. The recommended message identifier format is ccccHmmmms:

- cccc is the component identifier, such as ISG for global resource serialization.
- the required H represents IBM Health Checker for z/OS
- mmmm is the message number. For example, in message identifier ISGH101E, 101 is the message number
- s is the severity code for the message: E for exception messages and I for information messages

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,IDLEN=*idlen*

When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the message identifier.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *idlen* must be in the range 1 through 10.

,IDLEN=*idlen*

When REASON=CHECKINFO and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the message identifier.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *idlen* must be in the range 1 through 10.

,MF=S

,MF=(L,*list addr***)**
,MF=(L,*list addr***,***attr***)**
,MF=(L,*list addr***,0D)**
,MF=(E,*list addr***)**
,MF=(E,*list addr***,COMPLETE)**

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

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

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MGB=*mgb*
When REQUEST=CHECKMSG is specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1201 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1202 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1203 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1205 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1206 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1207 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1208 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1209 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1210 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1211 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1212 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1213 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1214 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1215 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1216 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGB=*mgb*
When REASON=PARS1218 or REASON=PARS1220 and REQUEST=HZSMSG are specified, a required input parameter that is the MGB control block used to describe the message request. The contents of the MGB are as described under the MGBADDR parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MGBADDR=*mgbaddr*
When REQUEST=CHECKMSG is specified, a required input parameter of the MGB control block used to describe the message request. The MGB identifies which message in the check's message table is requested and describes inserts to be used in that message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1201 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1202 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1203 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1205 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1206 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1207 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1208 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1209 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1210 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1211 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1212 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1213 and REQUEST=HZSMSG are specified, a required

input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1214 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1215 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1216 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBADDR=*mgbaddr*
When REASON=PARS1218 or REASON=PARS1220 and REQUEST=HZSMSG are specified, a required input parameter of the MGB control block used to describe the message request. The MGB describes inserts to be used in the message. The HZSMGB macro maps the MGB (structure name HZSMGB or HZSMGB1, according to the MGBFORMAT parameter of HZSFMSG).

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MGBFORMAT=0
,MGBFORMAT=1
An optional parameter, which indicates the format of the MGB provided by the MGBADDR or MGB parameter. The default is MGBFORMAT=0.

,MGBFORMAT=0
indicates that the format 0 MGB (mapped by dsect HZSMGB in macro HZSMGB is used).

,MGBFORMAT=1
indicates that the format 1 MGB (mapped by dsect HZSMGB1 in macro HZSMGB is used).

,MSGTABLE=*msgtable*

,MSGTABLE=HZSADDCK
When REMOTE=NO and REQUEST=CHECKMSG are specified, an optional input parameter that is the message table for the processing to use. This is intended to be used by "service routines" not owned by the check itself that are called by a check routine and write messages on behalf of the check, and that do not want their messages within the check's msgtable. The default is HZSADDCK. which indicates that the Msgtable provided on the HZSADDCK call is to be used.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MSGTABLE=*msgtable*
When REXX=NO, REMOTE=YES and REQUEST=CHECKMSG are specified, a required input parameter that is the message table for the processing to use.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ORESP=*oresp*
,ORESP=NO_ORESP
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the operator response for the message request. The length of the operator response text is specified by the ORESPLEN parameter. The default is NO_ORESP.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ORESPLEN=*oresplen*
,ORESPLEN=0
When ORESP=*oresp*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the operator response text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *oresplen* must be in the range 1 through 65535.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
,PLISTVER=2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are

assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, which supports all parameters except those specifically referenced in higher versions.
- **2**, which supports both the following parameters and those from version 1:

Parameters for PLISTVER=2

AUTOMATION	ORESPLEN	SPRESP
AUTOMATIONLEN	PROBD	SPRESPLEN
EXPL	PROBDLEN	SYSACT
EXPLLEN	REFDOC	SYSACTLEN
ID	REFDOCLEN	TEXT
IDLEN	SOURCE	TEXTLEN
ORESP	SOURCELEN	

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1, or 2

,PROBD=*probd*
,PROBD=NO_PROBD
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the problem description for the message request. The length of the problem description text is specified by the PROBDPLEN parameter. The default is NO_PROBD.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,PROBDLEN=*probdlen*
,PROBDLEN=0
When PROBD=*probd*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the problem description text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *probdlen* must be in the range 1 through 65535.

,REASON=CHECKEXCEPTION
,REASON=CHECKINFO
,REASON=CHECKREPORT
When REQUEST=DIRECTMSG is specified, a required parameter that indicates the class of the message being issued.

,REASON=CHECKEXCEPTION
indicates that an exception message is being issued.

,REASON=CHECKINFO
indicates that an information message is being issued.

,REASON=CHECKREPORT
indicates that a report message is being issued.

,REASON=ERROR
,REASON=PARS1201
,REASON=PARS1202

,REASON=PARS1203
,REASON=PARS1204
,REASON=PARS1205
,REASON=PARS1206
,REASON=PARS1207
,REASON=PARS1208
,REASON=PARS1209
,REASON=PARS1210
,REASON=PARS1211
,REASON=PARS1212
,REASON=PARS1213
,REASON=PARS1214
,REASON=PARS1215
,REASON=PARS1216
,REASON=PARS1217
,REASON=PARS1218
,REASON=PARS1219
,REASON=PARS1220

When REQUEST=HZSMSG is specified, a required parameter that indicates the type of situation being reported.

,REASON=ERROR
indicates that the message is being issued because of an error. The system is to issue message HZS1002E. This message is also recorded in the check's message buffer.

The state of the check is changed to error. The check remains active.

,REASON=PARS1201
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1201E, parm IS REQUIRED BUT WAS NOT SPECIFIED. The caller must provide exactly one insert, containing the parameter that is required. The insert length is limited to 16.

,REASON=PARS1202
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1202E, parm WAS SPECIFIED BUT IS NOT ALLOWED. The caller must provide exactly one insert, containing the parameter that was specified. The insert length is limited to 16.

,REASON=PARS1203
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1203E, PARAMETER parm VALUE value IS NOT VALID. The caller must provide exactly two inserts, containing the parameter name and the value that was specified, respectively. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1204
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1204E, UNEXPECTED END OF PARAMETER STRING. The caller must provide no inserts.

,REASON=PARS1205
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1205E, A PARAMETER WAS EXPECTED BUT string WAS FOUND INSTEAD. The caller must provide exactly one insert, containing the string that was found. The insert length is limited to 17.

,REASON=PARS1206
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1206E, A DELIMITER WAS EXPECTED BUT string WAS FOUND INSTEAD The caller must provide exactly one insert, containing the string that was found. The insert length is limited to 17.

,REASON=PARS1207
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1207E, PARAMETER parm HAS TOO MANY VALUES, n. The caller must provide exactly two inserts, containing the parameter name and the number of values, respectively. The number of values is to be provided not as a printable field but as a halfword or fullword field containing the information. The length of the first insert is limited to 16. The length of the second insert is limited to 4.

,REASON=PARS1208
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1208E, PARAMETER parm HAS TOO FEW VALUES, n. The caller must provide exactly two inserts, containing the parameter name and the number of values, respectively. The number of values is to be provided not as a printable field but as a halfword or fullword field containing the information. The length of the first insert is limited to 16. The length of the second insert is limited to 4.

,REASON=PARS1209
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1209E, PARAMETER parm IS NOT RECOGNIZED. The caller must provide exactly one insert, containing the parameter that was not recognized. The insert length is limited to 17.

,REASON=PARS1210
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1210E, PARAMETER parm IS MISSING ITS VALUE. The caller must provide exactly one insert, containing the parameter name. The insert length is limited to 16.

,REASON=PARS1211
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1211E, PARAMETER parm VALUE value IS TOO LARGE. The caller must provide exactly two inserts, containing the parameter name and the value, respectively. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1212
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1212E, PARAMETER parm VALUE value IS TOO SMALL. The caller must provide exactly one insert, containing the parameter name. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1213
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1213E, PARAMETER parm VALUE IS TOO LONG. The caller must provide exactly two inserts, containing the parameter name and the value, respectively. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1214
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1214E, PARAMETER parm VALUE value IS TOO

SHORT. The caller must provide exactly two inserts, containing the parameter name and the value, respectively. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1215
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1215E, PARAMETER parm VALUE value IS NOT DECIMAL. The caller must provide exactly two inserts, containing the parameter name and the value, respectively. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1216
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1216E, PARAMETER parm VALUE value IS NOT HEXADECIMAL. The caller must provide exactly two inserts, containing the parameter name and the value, respectively. The length of the first insert is limited to 16. The length of the second insert is limited to 17.

,REASON=PARS1217
indicates that the message is being issued because parameters were provided but none were expected, HZS1217E, PARAMETERS WERE SPECIFIED BUT ARE NOT ALLOWED. The caller must provide no inserts.

,REASON=PARS1218
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1218E, PARAMETER NUMBER n WAS NOT PROCESSED. The caller must provide exactly one insert, containing the parameter that was not recognized. The number of values is to be provided not as a printable field but as a halfword or fullword field containing the information. The length of the insert is limited to 4.

,REASON=PARS1219
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1219E, MIXING POSITIONAL AND KEYWORD FORMATS IS NOT ALLOWED. The caller must provide no inserts.

,REASON=PARS1220
indicates that the message is being issued due to a parameter parsing error, issuing message HZS1220E, *parm1* IS NOT ALLOWED WITH *parm2*. The caller must provide two inserts, containing *parm1* and *parm2*. The length of each insert is limited to 24.

,REASON=BADPARM
,REASON=ERROR
,REASON=ENVNA
When REQUEST=STOP is specified, a required parameter that indicates the type of situation being reported

,REASON=BADPARM
indicates that the parameters are not valid. The system is to issue message HZS1001E. This message is also recorded in the check's message buffer.

The state of the check is changed to parameter error. The check remains disabled until the PARMS are changed, presumably to address the error.

,REASON=ERROR
indicates that the message is being issued because of an error. The system is to issue message HZS1002E.

The state of the check is changed to error. The check is disabled. If a request is made to run the check, the check routine receives control for check initialization.

,REASON=ENVNA
indicates that the check is not applicable in the current system environment. Message HZS1003E is written as hardcopy-only and is also written to the check's message buffer.

The state of the check is changed to not applicable. The check is disabled. The check will not be called again until the reason for the condition is resolved and the check is refreshed (or its parameter is changed).

,REFDOC=*refdoc*
,REFDOC=NO_REFDOC
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the reference documentation for the message request. The length of the reference documentation text is specified by the REFDOCLEN parameter. The default is NO_REFDOC.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,REFDOCLEN=*refdoclen*
,REFDOCLEN=0
When REFDOC=*refdoc*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the reference documentation text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *refdoclen* must be in the range 1 through 65535.

,REMOTE=NO
,REMOTE=YES
When REQUEST=CHECKMSG is specified, an optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,REMOTE=NO
,REMOTE=YES
When REQUEST=DIRECTMSG is specified, an optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,REMOTE=NO

,REMOTE=YES
When REQUEST=HZSMSG is specified, an optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,REMOTE=NO
,REMOTE=YES
When REQUEST=STOP is specified, an optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,REMOTE=NO
,REMOTE=YES
When REQUEST=DOM is specified, an optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

REQUEST=CHECKMSG
REQUEST=DIRECTMSG
REQUEST=HZSMSG
REQUEST=STOP
REQUEST=DOM
A required parameter that identifies the source of the message text.

REQUEST=CHECKMSG
indicates that the message text is provided in the message table identified by the MSGTBL parameter of the HZSADDCK macro when the check was added.

REQUEST=DIRECTMSG
indicates that the message content is not provided by any message table, but is provided directly via additional parameters to HZSFMSG.

REQUEST=HZSMSG
indicates that the message text is provided by IBM Health Checker for z/OS.

REQUEST=STOP
indicates that the system is to stop calling this check. The message text is provided by IBM Health Checker for z/OS.

REQUEST=DOM
indicates that the system is to issue DOM requests at this time for all outstanding WTOs for this check from all previous iterations. REQUEST=DOM is:

- only allowed before any check exception messages have been sent in the current check iteration.
- only allowed for checks which have been added with parameter DOM(CHECK), see macro HZSADDCK for details.

If a check with attribute DOM(CHECK) does not trigger a DOM request for previous WTOs this way, check exception messages of the current check iteration will only be stored in the message buffer and no WTOs will be sent for them.

A check can use this to avoid repeated WTOs for identical exceptions in consecutive check iterations. The remaining code flow stays exactly the same, in particular HZSFMSG is still called, the (current and possibly refreshed) message text gets copied into the message buffer etc. The system will just not automatically issue a DOM request for previous WTOs and it will not issue a new WTO for those HZSFMSG calls where the check still has WTOs from a previous iteration for which DOM has not been issued yet.

A check iteration without any check exceptions will not trigger automatic DOM requests for outstanding WTOs for a check with attribute DOM(CHECK). For such iterations the check is required to issue HZSFMSG REQUEST=DOM at the latest at the very end of the iteration, otherwise the system will disable the check.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,REXX=NO
When REMOTE=YES and REQUEST=CHECKMSG are specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is not a REXX check.

,REXX=NO
When REMOTE=YES and REQUEST=DIRECTMSG are specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is not a REXX check.

,REXX=NO
When REMOTE=YES and REQUEST=STOP are specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is not a REXX check.

,REXX=NO
When REMOTE=YES and REQUEST=DOM are specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is not a REXX check.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SEVERITY=SYSTEM
,SEVERITY=NONE
,SEVERITY=LOW
,SEVERITY=MED
,SEVERITY=HI
,SEVERITY=VALUE
When REQUEST=CHECKMSG is specified, an optional parameter that is the check exception message severity to use. The SEVERITY keyword is:

- only allowed for check exceptions (not for info or report messages)
- only allowed for checks which have been added with parameter AllowDynSev(YES), see macro HZSADDCK.

The default is SEVERITY=SYSTEM.

,SEVERITY=SYSTEM
indicates that the system will determine the severity from the SEVERITY specified at ADD CHECK time, or on any subsequent UPDATE CHECK statements.

,SEVERITY=NONE
indicates that you're assigning no severity to this exception message

,SEVERITY=LOW
indicates that this is a low-severity check exception.

,SEVERITY=MED
indicates that this is a medium-severity check exception.

,SEVERITY=HI
indicates that this is a high-severity check exception.

,SEVERITY=VALUE
indicates that the value specified by SEVERITYVAL is to be used.

,SEVERITY=SYSTEM
,SEVERITY=NONE
,SEVERITY=LOW
,SEVERITY=MED
,SEVERITY=HI
,SEVERITY=VALUE
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional parameter that is the check exception message severity to use. The SEVERITY keyword is only allowed for checks which have been added with parameter AllowDynSev(YES), see macro HZSADDCK. The default is SEVERITY=SYSTEM.

,SEVERITY=SYSTEM
indicates that the system will determine the severity from the SEVERITY specified at ADD CHECK time, or on any subsequent UPDATE CHECK statements.

,SEVERITY=NONE
indicates that you're assigning no severity to this exception message

,SEVERITY=LOW
indicates that this is a low-severity check exception.

,SEVERITY=MED
indicates that this is a medium-severity check exception.

,SEVERITY=HI
indicates that this is a high-severity check exception.

,SEVERITY=VALUE
indicates that the value specified by SEVERITYVAL is to be used.

,SEVERITYVAL=*severityval*
When SEVERITY=VALUE and REQUEST=CHECKMSG are specified, a required input parameter, that indicates the severity to be used. It must be one of the values defined by the equates generated by the list form of the HZSFMSG macro. For example HZSFMSG MF=(L,xxx) generates xxx_XSEVERITY_yyy, where yyy is one of SYSTEM, NONE, LOW, MED, or HI.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,SEVERITYVAL=*severityval*
When SEVERITY=VALUE, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter, that indicates the severity to be used. It must be one of the values defined by the equates generated by the list form of the HZSFMSG macro. For example HZSFMSG MF=(L,xxx) generates xxx_XSEVERITY_yyy, where yyy is one of SYSTEM, NONE, LOW, MED, or HI.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,SOURCE=*source*
,SOURCE=NO_SOURCE
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the message source for the message request. The length of the message source text is specified by the SOURCELEN parameter. The default is NO_SOURCE.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,SOURCELEN=*sourcelen*
,SOURCELEN=0
When SOURCE=*source*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the message source text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *sourcelen* must be in the range 1 through 65535.

,SPRESP=*spresp*
,SPRESP=NO_SPRESP
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the system programmer response

for the message request. The length of the system programmer response text is specified by the SPRESPLEN parameter. The default is NO_SPRESP.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,SPRESPLEN=*spresplen*
,SPRESPLEN=0
When SPRESP=*spresp*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the system programmer response text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *spresplen* must be in the range 1 through 65535.

,SYSACT=*sysact*
,SYSACT=NO_SYSACT
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, an optional input parameter that is the system action for the message request. The length of the system action text is specified by the SYSACTLEN parameter. The default is NO_SYSACT.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,SYSACTLEN=*sysactlen*
,SYSACTLEN=0
When SYSACT=*sysact*, REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the system action text with a maximum of 65535. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *sysactlen* must be in the range 1 through 65535.

,TEXT=*text*
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that is the message text for the message request. The length of the message text is specified by the TEXTLEN parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,TEXT=*text*
When REASON=CHECKINFO and REQUEST=DIRECTMSG are specified, a required input parameter that is the message text for the message request. The length of the message text is specified by the TEXTLEN parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,TEXT=*text*
When REASON=CHECKREPORT and REQUEST=DIRECTMSG are specified, a required input parameter that is the message text for the message request. The length of the message text is specified by the TEXTLEN parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,TEXTLEN=*textlen*
When REASON=CHECKEXCEPTION and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the message text with a maximum of 65535.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *textlen* must be in the range 1 through 65535.

,TEXTLEN=*textlen*
When REASON=CHECKINFO and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the message text with a maximum of 65535.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *textlen* must be in the range 1 through 65535.

,TEXTLEN=*textlen*
When REASON=CHECKREPORT and REQUEST=DIRECTMSG are specified, a required input parameter that contains the length of the message text with a maximum of 65535.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *textlen* must be in the range 1 through 65535.

ABEND Codes

058 The IBM Health Checker for z/OS address space terminated while this call was in process.

290 HZSFMSG service failed a request. There may be additional diagnostic data in the registers at time of the abend.

Register	Diagnostic data when REQUEST=CHECKMSG fails
2	The message id passed in MGB_MessageNumber
3	The number of variable inserts passed in MGB_INSERT_CNT
4	The PQE address passed to the Check routine
5	The number of variables that have not been processed
6	Addition information for internal diagnosis by IBM
7	Address of data in the message table that was being processed
8	Data pointed to by R7, or the address of the check routine

An abend 290 will be issued if an error in the request is detected. Addition detail is recorded in LOGREC for this error.

In the following HZSFMSG abend reason codes, the bytes designated "xx" are for diagnostic purposes and have no significance to the external interface.

User errors are indicated by an abend reason code of the form xxxx4xxx.

Component errors are indicated by an abend reason code of the form xxxx1xxx.

Reason Code (Hex)	Explanation

xxxx4106
The HZSMGB was not avaliable in storage in the caller's key.

xxxx4107
A variable insert in the HZSMGB had a bad address or length.

xxxx4108
The message number could not be found in the message table.

xxxx4109
The MGB_INSERT_CNT contain a value that was higher than the maximum number of insert allowed.

xxxx410A
The message table is in error. A message insert was requested in the incorrect sequence.

xxxx410B
A message insert is required to complete the message, but it was not provided.

xxxx410C
A message insert was provided, but it was not needed to complete the message.

xxxx410D
A message insert address was zero.

xxxx410E
A message insert length was not valid for the requested variable. A text insert must be from 0-256, a hex insert must be from 1-100, and the rest must be 8 charaters or less.

xxxx410F
The parameter list was not available in storage in the caller's key.

xxxx4110
The address of the HZSMGB area is zero when the request required a completed HZSMGB.

xxxx4111
The parameter version is not supported.

xxxx4112
The calling routine is not a check routine, or the environment is not valid. For example due to:

- missing MESSAGETABLE
- missing REMOTE(YES)
- caller not in TASK mode
- invalid REXX home

xxxx4113
The calling routine did not provide a valid handle.

xxxx4114
The calling remote routine is not a check routine.

xxxx4115
ABENDRESULT was specified, but could not be set because it is not in storage in the caller's key.

xxxx4016
The variable defined in the HZSMGB area has a length greater than the value defined by Maxlen in the message table.

xxxx4116
The variable defined in the HZSMGB area has a length greater than the value defined by Fieldsize in the message table.

xxxx4117
The message table supplied by a remote check is not valid. Make sure that the message table was built via the HZSMSGEN exec and has not been overlaid.

xxxx4118
A remote check issued HZSFMSG other than from the INITRUN or RUN function

xxxx4119
HZSFMSG has been called with an unrecognized REQUEST type

xxxx4120
HZSFMSG has been called with an unrecognized REASON for this particular REQUEST type

xxxx4122
IDLEN parameter is out of range

xxxx4123
TEXTLEN parameter is out of range

xxxx4124
ID parameter is null

xxxx4125
TEXT parameter is null

xxxx412B
EXPL parameter is null

xxxx412C
EXPLLEN parameter is out of range

xxxx412E
SYSACT parameter is null

xxxx412F
SYSACTLEN parameter is out of range

xxxx4130
ORESP parameter is null

xxxx4131
ORESPLEN parameter is out of range

xxxx4132
SPRESP parameter is null

xxxx4133
SPRESPLEN parameter is out of range

xxxx4134
PROBD parameter is null

xxxx4135
PROBDLEN parameter is out of range

xxxx4136
SOURCE parameter is null

xxxx4137
SOURCELEN parameter is out of range

xxxx4138
REFDOC parameter is null

xxxx4139
REFDOCLEN parameter is out of range

xxxx413A
AUTOMATION parameter is null

xxxx413B
AUTOMATIONLEN parameter is out of range

xxxx4140
REQUEST=DOM is not allowed for DOM(SYSTEM)

xxxx4141
REQUEST=DOM is not allowed after first check exception in a check iteration

xxxx4150
Non-SYSTEM SEVERITY or SEVERITYVAL not allowed for an AllowDynSev(NO) check

xxxx4151
Bad 'sev' in SEVERITY(sev)

xxxx4152
Bad 'val' in SEVERITYVAL(val)

xxxx1001
An unexpected internal error occurred.

xxxx1013
The message table contains data that cannot be processed.

xxxx1014
The Pqe control block was not found.

xxxx1015
A message variable description was bad.

xxxx1017
The message table contains data that incorrectly defines a Maxlen value. The table is corrupted.

xxxx1018
The message table contains data that allows a WTO line to exceed 71 characters. The table is corrupted.

xxxx1019
The Hcklog control block contains errors.

Return and Reason Codes

When the HZSFMSG macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HZSZCONS provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 41. Return and Reason Codes for the HZSFMSG Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HzsfmsgRc_OK **Meaning**: The request completed successfully. **Action**: None required
8	—	**Equate Symbol**: HzsfmsgRc_InvParm **Meaning**: HZSFMSG request specifies incorrect parameters. **Action**: Refer to action under the individual reason code.
8	xxxx0837	**Equate Symbol**: HzsfmsgRsn_ErrorLimitExceeded **Meaning**: The check routine has abended too many times, messages will not be processed. **Action**: Fix the check routine.
0C	—	**Equate Symbol**: HzsfmsgRc_EnvError **Meaning**: Environmental Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: HzsfmsgRsn_IBMHCNotActive **Meaning**: IBM Health Checker for z/OS is not active **Action**: Re-issue the request when the service is available
10	—	**Equate Symbol**: HzsfmsgRc_CompError **Meaning**: Component Error. An associated dump and logrec entry has been created using abend 290 and the reason code. **Action**: Refer to action under the individual reason code.
10	xxxx1001	**Equate Symbol**: HzsfmsgRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer
10	xxxx1013	**Equate Symbol**: HzsfmsgRsn_MsgTblError **Meaning**: The message table could not be processed. **Action**: Report the problem to the system programmer
10	xxxx1014	**Equate Symbol**: HzsfmsgRsn_Pqe_NotValid **Meaning**: The Pqe control block could not be found. **Action**: Report the problem to the system programmer

Table 41. Return and Reason Codes for the HZSFMSG Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
10	xxxx1015	**Equate Symbol**: HzsfmsgRsn_BadMsgTblSegment **Meaning**: A message variable is incorrectly defined in the message table. **Action**: Report the problem to the system programmer
10	xxxx1017	**Equate Symbol**: HfmsgAbend_BadMsgTblOutLen **Meaning**: The message table contains data that incorrectly defines a Maxlen value. The table is corrupted. **Action**: Report the problem to the system programmer
10	xxxx1018	**Equate Symbol**: HzsfmsgAbend_MsgTblMissingNewLine **Meaning**: The message table contains data that allows a WTO line to exceed 71 characters. The table is corrupted **Action**: Report the problem to the system programmer
10	xxxx1019	**Equate Symbol**: HzsfmsgRsn_HckLog_NotValid **Meaning**: The Hcklog control block contains errors. **Action**: Report the problem to the system programmer

Examples

Example 1:

Operation:

- Issue a message with two inserts where the first insert is known at assembly time and the second is an 8-character name not known at assembly time.

The code is as follows.

```
************************************************************
* Issue a message with two inserts                         *
************************************************************
SYSSTATE ARCHLVL=1 * save regs, get dynamic storage, chain saveareas,
         MVC   MGB_MsgIVal(L'Insert2Val),MyMod Insert value
         DROP  3
         ST    3,MGB_Inserts+4       Save insert address
         HZSFMSG  REQUEST=CHECKMSG,MGBADDR=TheMGBAddr,       *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,FMSGL)
         DROP  2
*
* Place code to check return/reason codes here
*
* free dynamic storage, restore regs
         BR   14
MyMod    DC   CL8'MYMODULE'
* Area for first insert
Insert1Area DS 0H
Insert1Len  DC AL2(L'Insert1Val)
Insert1Val  DC C'This is the first insert'
         LTORG ,
         HZSZCONS ,            Return code information
         HZSMGB   ,            Insert mapping
DYNAREA  DSECT
LRETCODE DS   F
LRSNCODE DS   F
* Area for 2 inserts (HZSMGB_LEN accounts for one, so
* we add one more "length of MGB_Inserts")
TheMGBAddr DS A
TheMGBArea DS CL(HZSMGB_LEN+1*L'MGB_Inserts)
```

```
* Area for second insert
Insert2Area DS 0H
Insert2Len  DS CL(L'MGB_MsgInsertD_Header)
Insert2Val  DS CL(L'MyMod)
         HZSFMSG MF=(L,FMSGL),PLISTVER=MAX
DYNAREA_LEN EQU *-DYNAREA
```

Example 2:

Operation:

- Same as example 1, but using MGBFORMAT=1 and the MGB parameter rather than the MGBADDR parameter.

The code is as follows.

```
*************************************************************
* Issue a message with two inserts, MGBFORMAT=1.            *
* For MGBFORMAT=0, it was necessary to move the insert      *
* data if it was not in the form of a halfword length       *
* followed by the data.                                     *
* For MGBFORMAT=1, that is not necessary since the length   *
* is specified in the MGB_MsgInsertDesc DSECT.              *
*************************************************************
         SYSSTATE ARCHLVL=1
* save regs, get dynamic storage, chain saveareas, set usings
* somewhere there needs to be code to set Insert2Val
         LA    2,TheMGB1Area
         USING HZSMGB1,2
         MVC   MGB1_MessageNumber,=F'1' Message 1
         MVC   MGB1_insert_cnt,=F'2' Two inserts
* address first insert description
         LA    3,MGB1_insert_structure_Entries
         USING MGB1_MsgInsertDesc,3
* fill in first insert description
         LA    4,L'Insert1Val
         ST    4,MGB1_MsgInsertDesc_Length
         LA    4,Insert1Val
         ST    4,MGB1_MsgInsertDesc_Addr
* move on to next insert description
         LA    3,MGB1_insert_structure_Entries_Len(,3)
* fill in second insert description
         LA    4,L'Insert2Val
         ST    4,MGB1_MsgInsertDesc_Length
         MVC   Insert2Val(L'Insert2Val),MyMod Insert value
         LA    4,Insert2Val
         ST    4,MGB1_MsgInsertDesc_Addr
         DROP  3
         HZSFMSG  REQUEST=CHECKMSG,MGB=TheMGB1Area,          *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MGBFORMAT=1,MF=(E,FMSGL)
         DROP  2
*
* Place code to check return/reason codes here
*
* free dynamic storage, restore regs
         BR   14
MyMod    DC   CL8'MYMODULE'
* Area for first insert
Insert1Val  DC C'This is a static insert'
         LTORG ,
         HZSZCONS ,            Return code information
         HZSMGB   ,            Insert mapping
DYNAREA  DSECT
LRETCODE DS   F
LRSNCODE DS   F
* Area for 2 inserts (HZSMGB1_LEN accounts for none,
* we add two "length of MGB1_MsgInsertDesc"
           DS 0F
TheMGB1Area DS CL(HZSMGB1_LEN+1*L'MGB1_Inserts)
* Area for second insert
Insert2Val  DS CL(L'MyMod)
         HZSFMSG MF=(L,FMSGL),PLISTVER=MAX
DYNAREA_LEN EQU *-DYNAREA
```

HZSPREAD macro — Read Check Persistent Data

Description

The HZSPREAD macro is an interface used by check routines to read data that has been preserved in the IBM Health Checker for z/OS Persistent Data data set, which is allocated using the HZSPDATA ddname in the startup proc. Two groups of data are preserved for an IPL, the first and the most recent.

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	Problem state. PSW key 8-15 When problem state and key 8-15 and not APF authorized, or when SECCHECK=ALL is specified, the caller must be authorized for read access to either of the following: • XFACILIT class resource HZS.sysname.checkowner.PDATA • XFACILIT class resource HZS.sysname.checkowner.checkname.PDATA
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters must be below 2G. The user-provided buffer (via BUFFER) has the same requirements and restrictions as the control parameters.

Programming Requirements

- This service is supported only when it is called from a check routine invoked by IBM Health Checker for z/OS.
- The storage used by this service should be in the same key as the caller.

Restrictions

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSPREAD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSPREAD macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code, when register 15 is not 0. Otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

main diagram

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the HZSPREAD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,BUFFER=*buffer*
A required input parameter that is the buffer in which to return the data. The buffer should be in the same key as the caller.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,BYTESAVAIL=*bytesavail*
A required output parameter that indicates the total number of bytes that were available to be returned. If that number is less than or equal to the sum of the values provided by the DataLen and StartByte parameters, then all requested bytes were returned. If that number is greater than the sum of the values provided by the DataLen and StartByte parameters, then the number of bytes returned matches the value provided by the DataLen parameter, and subsequent calls should be made to retrieve the additional data, adding the value in the StartByte parameter to the value in the DataLen parameter to form the value for the StartByte parameter in the next call.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,CHECKNAME=*checkname*
A required input parameter that specifies the name of the check that has saved persistent data.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

CHECKOWNER=*checkowner*
A required input parameter that specifies the owner of the check that has saved persistent data.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,DATALEN=*datalen*
A required input parameter that contains the number of bytes of data to return.

When requesting data for a check other than your own, a startbyte of 0 will be used regardless of what you specify, so you should use a datalen that will accomplish returning all the data in a single request.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,HANDLE=*handle*
When REMOTE=YES is specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the

HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. It is provided in REXX variable HZS_HANDLE for a REMOTE=YES REXX=YES check.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,INSTANCE=FIRST
,INSTANCE=MOSTRECENT
A required parameter that indicates which instance of the data is to be returned.

,INSTANCE=FIRST
indicates that the first instance of this check's data for the selected IPL should be returned.

,INSTANCE=MOSTRECENT
indicates that the most recent instance of this check's data for the selected IPL should be returned.

,IPL=CURRENT
,IPL=PRIOR
A required parameter that indicates which IPL's data is to be returned.

,IPL=CURRENT
indicates that data from this IPL is to be returned.

,IPL=PRIOR
indicates that data from the prior IPL is to be returned.

,MF=S
,MF=(L,*list addr***)**
,MF=(L,*list addr***,***attr***)**
,MF=(L,*list addr***,0D)**
,MF=(E,*list addr***)**
,MF=(E,*list addr***,COMPLETE)**
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,*attr*
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter

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

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,REMOTE=NO
,REMOTE=YES
An optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RETIPLTOD=*retipltod*
An optional output parameter that is to contain the IPL TOD of the persistent data. It is in STCK format.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,RETPTIME=*retptime*
An optional output parameter that specifies the time the persistent data record was written. It is in STCK format.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,REXX=NO
,REXX=YES
When REMOTE=YES is specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is not a REXX check.

,REXX=YES
indicates that the check is a REXX check.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SECCHECK=UNAUTH
,SECCHECK=ALL
An optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases. If RACF does not grant authority, the request is rejected.

,STARTBYTE=*startbyte*
,STARTBYTE=FIRST_BYTE
An optional input parameter that indicates which byte to begin with. For the first call, the most likely value would be 0 to indicate the "first byte". Subsequent calls would most likely use the previous StartByte value plus the value provided by the DataLen parameter, when the value returned in the BytesAvail parameter of the previous call exceeded the value provided by the DataLen parameter.

Note that if reading from a check other than your own a StartByte of 0 is used, regardless of what you specify. The default is FIRST_BYTE.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

ABEND Codes

None.

Return and Reason Codes

When the HZSPREAD macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HZSZCONS provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 42. Return and Reason Codes for the HZSPREAD Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HzspreadRc_OK **Meaning**: The request was successfully processed. Note that, if your buffer was not large enough, you might not have retrieved all possible data. Refer to the BytesAvail parameter description for more information. **Action**: None required
8	—	**Equate Symbol**: HzspreadRc_InvParm **Meaning**: HZSPREAD request specifies incorrect parameters. **Action**: Refer to action under the individual reason code.
8	xxxx0801	**Equate Symbol**: HzspreadRsn_NotAuthorized **Meaning**: Caller is not authorized to access persistent data for this check **Action**: Avoid calling HZSPREAD to access data for a check when not authorized.
8	xxxx0808	**Equate Symbol**: HzspreadRsn_BadENV **Meaning**: HZSPREAD is supported only when called within the HZS address space. **Action**: Invoke HZSPREAD only within the HZS address space.
8	xxxx0818	**Equate Symbol**: HzspreadRsn_BadParmlist **Meaning**: Error accessing parameter list. **Action**: Make sure that the provided parameter list is valid.
8	xxxx082D	**Equate Symbol**: HzspreadRsn_NoMatch **Meaning**: No persistent data records exist for this check. **Action**: Make sure that you requested the proper information.
8	xxxx0830	**Equate Symbol**: HzspreadRsn_DataDoesNotExist **Meaning**: The startbyte requested for the specified instance is not available. **Action**: Make sure that you requested the proper information.

Table 42. Return and Reason Codes for the HZSPREAD Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0843	**Equate Symbol**: HzspreadRsn_SrbMode **Meaning**: SRB mode. **Action**: Avoid issuing HZSPREAD in SRB mode.
8	xxxx0844	**Equate Symbol**: HzspreadRsn_NotEnabled **Meaning**: Not Enabled. **Action**: Avoid using HZSPREAD when not enabled.
8	xxxx0845	**Equate Symbol**: HzspreadRsn_Locked **Meaning**: Locked **Action**: Avoid using HZSPREAD when a lock is held.
8	xxxx0846	**Equate Symbol**: HzspreadRsn_FRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using HZSPREAD when an EUT FRR is established.
8	xxxx0847	**Equate Symbol**: HzspreadRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	xxxx0848	**Equate Symbol**: HzspreadRsn_BadBufferALET **Meaning**: Bad answer area ALET. **Action**: Make sure that the ALET associated with the buffer is valid. The access register might not have been set up correctly.
8	xxxx0849	**Equate Symbol**: HzspreadRsn_BadBuffer **Meaning**: Error accessing buffer **Action**: Make sure that the provided buffer is valid.
8	xxxx0858	**Equate Symbol**: HzspreadRsn_BadHandle **Meaning**: The handle provided with the HANDLE parameter is not valid. **Action**: Specify the handle that was returned by the HZSADDCK macro if this is a REMOTE=YES REXX=NO check.
8	xxxx085A	**Equate Symbol**: HzspreadRsn_WrongRemoteFunction **Meaning**: The check routine is not currently processing either the INITRUN or the RUN remote function. **Action**: Avoid invoking HZSPREAD for a remote check when not within the INITRUN or RUN function.

Table 42. Return and Reason Codes for the HZSPREAD Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx085B	**Equate Symbol**: HzspreadRsn_BadRemoteEnvironment **Meaning**: HZSPREAD was invoked from a task other than the one that issued HZSCHECK REQUEST=OPSTART. **Action**: Avoid invoking HZSPREAD from an incorrect task.
8	xxxx0861	**Equate Symbol**: HzspreadRsn_WrongFunction **Meaning**: The check routine is not currently processing either the INIT, CHECK, or CLEANUP function. **Action**: Avoid invoking HZSPREAD for a local check when not within the INIT or CHECK function.
10	—	**Equate Symbol**: HzspreadRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.
10	xxxx1001	**Equate Symbol**: HzspreadRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer

Examples

Example 1:: Operation:

- Retrieve the peristent data stored from the previous IPL.

The code is as follows.

```
************************************************************
* Retrieve previous IPL persistent data                    *
************************************************************
         SYSSTATE ARCHLVL=2
         LHI   2,L'thebuff     Length to read
         ST    2,thelen
         SLR   2,2             Value of 0
         ST    2,startbyte     Start at byte 0
         HZSPREAD CHECKOWNER=cowner,CHECKNAME=cname,        *
               IPL=PRIOR,INSTANCE=FIRST,                    *
               STARTBYTE=startbyte,                         *
               BUFFER=thebuff,DATALEN=thelen,               *
               BYTESAVAIL=avail,                            *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,PREADL)
*
* Place code to check return/reason codes here
*
*
* Place code to process the output buffer here
*
cowner   DC   CL16'MY_CHECK_OWNER'
cname    DC   CL32'MY_CHECK_NAME'
         LTORG ,
         HZSZCONS ,            Return code information
DYNAREA  DSECT
thelen   DS   F                Length of output buffer
avail    DS   F                Available bytes
startbyte DS  F                Starting byte position
```

```
LRETCODE DS   F
LRSNCODE DS   F
         HZSPREAD MF=(L,PREADL),PLISTVER=MAX
         DS   0D
thebuff  DS   16384X           Output buffer
```

Example 2:: Operation:

- If additional data exists, re-issue HZSPREAD to read the next data

The code is as follows.

```
*
* Place code to check available bytes here, in case
* more bytes were available than could be returned in
* the output buffer
*
************************************************************
* Retrieve next group of prior IPL persistent data         *
* The example presumes that a call similar to that         *
* in the first example had been made                       *
************************************************************
         SYSSTATE ARCHLVL=2
         CLC   avail,thelen
         JNH   done_pread
         L     2,startbyte     Previous starting byte
         AL    2,thelen        Increment by length read
         ST    2,startbyte     Next starting byte
         LHI   2,L'thebuff     Length to read this time
         ST    2,thelen
         HZSPREAD CHECKOWNER=cowner,CHECKNAME=cname,       *
               IPL=PRIOR,INSTANCE=FIRST,                   *
               STARTBYTE=startbyte,                        *
               BUFFER=thebuff,DATALEN=thelen,              *
               BYTESAVAIL=avail,                           *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,          *
               MF=(E,PREADL)
*
* Place code to check return/reason codes here
*
*
* Place code to process the output buffer here
*
done_pread DS 0H
         ....
cowner   DC   CL16'MY_CHECK_OWNER'
cname    DC   CL32'MY_CHECK_NAME'
         LTORG ,
         HZSZCONS ,            Return code information
DYNAREA  DSECT
thelen   DS   F                Length of output buffer
avail    DS   F                Available bytes
startbyte DS  F                Starting byte position
LRETCODE DS   F
LRSNCODE DS   F
         HZSPREAD MF=(L,PREADL),PLISTVER=MAX
         DS   0D
```

HZSPWRIT macro — Write Check Persistent Data

Description

The HZSPWRIT macro is an interface used by check routines to write persistent data into the IBM Health Checker for z/OS Persistent Data data set, which is allocated using the HZSPDATA ddname in the startup proc. Use HZSPWRIT only within the Init, Check, or Cleanup function for a local check, or within the InitRun or Run function for a remote check.

The data is associated with the writing check, and can be retrieved by the HZSPREAD macro, specifying the check owner and check name. The data remains even if the writing check is deleted.

If the check iteration completes with an abend (or a remote check iteration is designated unsuccessful) or an invocation of HZSPWRIT is not successful, then the persistent data for that iteration is not retained. Note that "unsuccessful" has no correlation with whether or not the check detected exception(s).

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	Problem state. PSW key 8-15 When problem state and key 8-15 and not APF authorized, or when SECCHECK=ALL is specified, the caller must be authorized for update access to either of the following: • XFACILIT class resource HZS.sysname.checkowner.PDATA • XFACILIT class resource HZS.sysname.checkowner.checkname.PDATA
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters must be below 2G. The user-provided buffer (via BUFFER) has the same requirements and restrictions as the control parameters.

Programming Requirements

- This service is supported only when it is called from a check routine invoked by IBM Health Checker for z/OS.

Restrictions

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSPWRIT macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSPWRIT macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code, when register 15 is not 0. Otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

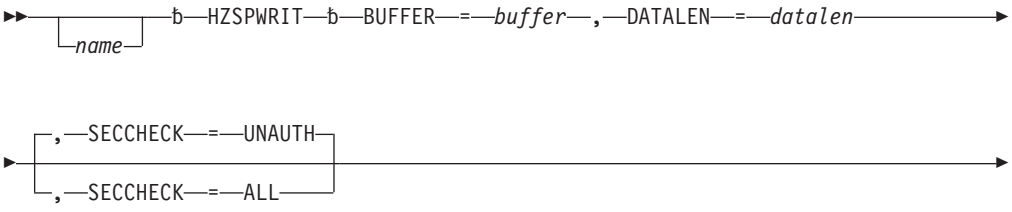
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

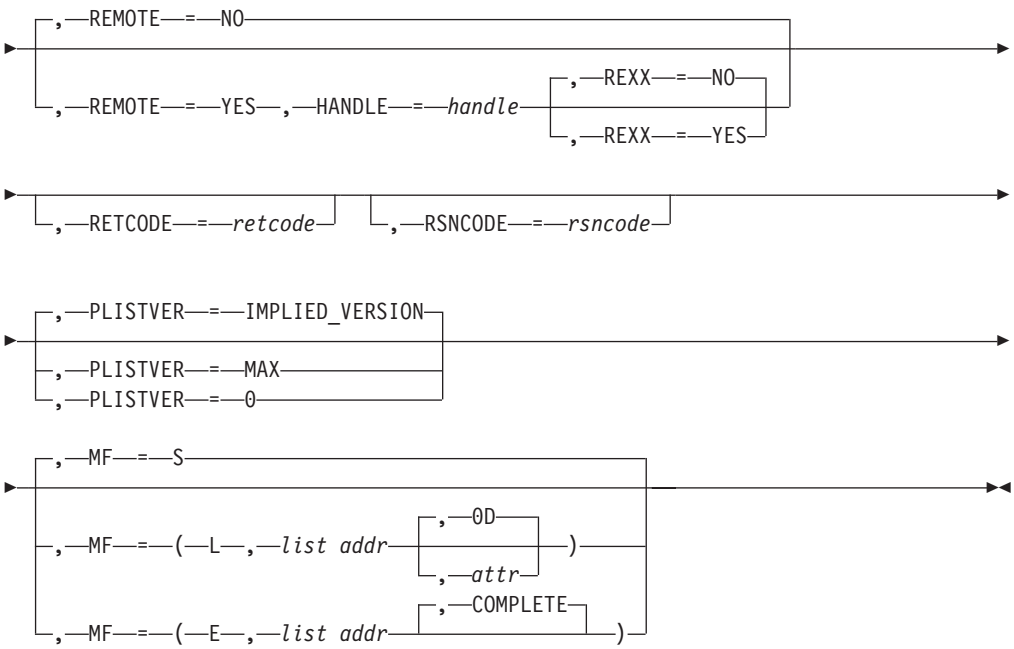
Performance Implications

None.

Syntax

main diagram





Parameters

The parameters are explained as follows:

name
: An optional symbol, starting in column 1, that is the name on the HZSPWRIT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

BUFFER=*buffer*
: A required input parameter, area that contains the data to be written. The buffer should be in the same key as the caller.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,DATALEN=*datalen*
: A required input parameter that contains the number of bytes of data from the buffer to be written.

  To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,HANDLE=*handle*
: When REMOTE=YES is specified, a required input parameter that specifies a handle (token) that identifies the check. This handle was returned via the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES REXX=NO check. It is provided in REXX variable HZS_HANDLE for a REMOTE=YES REXX=YES check.

  To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,MF=S
,MF=(L,*list addr***)**
,MF=(L,*list addr*,*attr***)**
,MF=(L,*list addr*,**0D)**
,MF=(E,*list addr***)**

,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,*attr*

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

,COMPLETE

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

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

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX

- A decimal value of 0

,REMOTE=NO
,REMOTE=YES
An optional parameter, which identifies the locale of the check. The default is REMOTE=NO.

,REMOTE=NO
indicates that the check runs locally in the address space of IBM Health Checker for z/OS.

,REMOTE=YES
indicates that the check runs remotely, in an address space other than that of IBM Health Checker for z/OS.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,REXX=NO
,REXX=YES
When REMOTE=YES is specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=NO.

,REXX=NO
indicates that the check is not a REXX check.

,REXX=YES
indicates that the check is a REXX check.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SECCHECK=UNAUTH
,SECCHECK=ALL
An optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases. If RACF does not grant authority, the request is rejected.

ABEND Codes
None.

Return and Reason Codes
When the HZSPWRIT macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.

- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HZSZCONS provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 43. Return and Reason Codes for the HZSPWRIT Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HzspwritRc_OK **Meaning**: The request was successfully processed. **Action**: None required
8	—	**Equate Symbol**: HzspwritRc_InvParm **Meaning**: HZSPWRIT request specified incorrect parameters. **Action**: Refer to action under the individual reason code.
8	xxxx0801	**Equate Symbol**: HzspwritRsn_NotAuthorized **Meaning**: Caller is not authorized to write persistent data for this check **Action**: Avoid calling HZSPWRIT to write data when not authorized.
8	xxxx0808	**Equate Symbol**: HzspwritRsn_BadENV **Meaning**: HZSPWRIT is supported only when called within the HZS address space. **Action**: Invoke HZSPWRIT only within the HZS address space.
8	xxxx0818	**Equate Symbol**: HzspwritRsn_BadParmlist **Meaning**: Error accessing parameter list. **Action**: Make sure that the provided parameter list is valid.
8	xxxx0843	**Equate Symbol**: HzspwritRsn_SrbMode **Meaning**: SRB mode. **Action**: Avoid issuing HZSPWRIT in SRB mode.
8	xxxx0844	**Equate Symbol**: HzspwritRsn_NotEnabled **Meaning**: Not Enabled. **Action**: Avoid using HZSPWRIT when not enabled.
8	xxxx0845	**Equate Symbol**: HzspwritRsn_Locked **Meaning**: Locked **Action**: Avoid using HZSPWRIT when a lock is held.

Table 43. Return and Reason Codes for the HZSPWRIT Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0846	**Equate Symbol**: HzspwritRsn_FRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using HZSPWRIT when an EUT FRR is established.
8	xxxx0847	**Equate Symbol**: HzspwritRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	xxxx0848	**Equate Symbol**: HzspwritRsn_BadBufferALET **Meaning**: Bad buffer ALET. **Action**: Make sure that the ALET associated with the buffer is valid. The access register might not have been set up correctly.
8	xxxx0849	**Equate Symbol**: HzspwritRsn_BadBuffer **Meaning**: Error accessing buffer **Action**: Make sure that the provided buffer is valid.
8	xxxx0858	**Equate Symbol**: HzspwritRsn_BadHandle **Meaning**: The handle provided with the HANDLE parameter is not valid. **Action**: Specify the handle that was returned by the HZSADDCK macro if this is a REMOTE=YES REXX=NO check.
8	xxxx085A	**Equate Symbol**: HzspwritRsn_WrongRemoteFunction **Meaning**: The check routine is not currently processing either the INITRUN or the RUN remote function. **Action**: Avoid invoking HZSPWRIT for a remote check when not within the INITRUN or RUN function.
8	xxxx085B	**Equate Symbol**: HzspwritRsn_BadRemoteEnvironment **Meaning**: HZSPWRIT was invoked from a task other than the one that issued HZSCHECK REQUEST=OPSTART. **Action**: Avoid invoking HZSPWRIT from an incorrect task.
8	xxxx0861	**Equate Symbol**: HzspwritRsn_WrongFunction **Meaning**: The check routine is not currently processing either the INIT, CHECK, or CLEANUP function. **Action**: Avoid invoking HZSPWRIT for a local check when not within the INIT or CHECK function.

Table 43. Return and Reason Codes for the HZSPWRIT Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
0C	—	**Equate Symbol**: HzspwritRc_EnvError **Meaning**: Environmental Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C15	**Equate Symbol**: HzspwritRsn_DataCorrupted **Meaning**: The persistent data being managed by the system for this check has been overlaid. It will not be written to the HZSPDATA data set. **Action**: Report the problem to the system programmer
10	—	**Equate Symbol**: HzspwritRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.
10	xxxx1001	**Equate Symbol**: HzspwritRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer

Example

- Write persistent data

The code is as follows.

```
****************************************
* Write data that is to persist        *
****************************************
SYSSTATE ARCHLVL=2 *
* Previous code would have saved data into area the buff *
* and saved the number of bytes to write in thelen       *
* HZSPWRIT BUFFER=thebuff,DATALEN=thelen,                *
*   RETCODE=LRETCODE,RSNCODE=LRSNCODE,                   *
*   MF=(E,PWRITL)                                        *
*
* Place code to check return/reason codes here            *
*
*          LTORG ,
*          HZSZCONS ,        Return Code information
* DYNAREA  DSECT
* thelen   DS   F            Length of output buffer
* LRETCODE DS   F
* LRSNCODE DS   F
*          HZSPWRIT MF=(L,PWRITL),PLISTVER=MAX
*          DS   0D
*          DS   8192X        Output buffer
```

HZSQUERY macro — HZS Query

Description

The HZSQUERY macro provides the interface to obtain information about checks that are currently registered with IBM Health Checker for z/OS.

Environment

The requirements for the caller are:

Requirement	Description
Minimum authorization:	Problem state. PSW key 8-15 When problem state and key 8-15 and not APF authorized, or when SECCHECK=ALL is specified, the caller's authorization requirements depend on the input specification. • The caller must be authorized for read access to any of the following: – when the check owner has wildcard character(s), XFACILIT class resource HZS.sysname.reqtype – when the check owner has no wildcard characters and the check name has wildcard character(s), XFACILIT class resource HZS.sysname.checkowner.reqtype – when the check owner has no wildcard characters and the check name has no wildcard characters, XFACILIT class resource HZS.sysname.checkowner.checkname.reqtype or XFACILIT class resource HZS.sysname.checkowner.reqtype • The values for reqtype are as follows – When REQUEST=MSGBUFF is specified, reqtype is MESSAGES. – When REQUEST=CHKINFO is specified, reqtype is QUERY. – When REQUEST=GENINFO is specified, reqtype is QUERY.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.

Requirement	Description
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters must be below 2G. The user-provided answer area via the ANSAREA parameter has the same requirements and restrictions as the control parameters. The user-provided answer area via the ANSAREA64 parameter has the same requirements and restrictions as the control parameters but can be above 2G for an AMODE 64 caller. The user-provided area via the QUAAC1HDR parameter has the same requirements and restrictions as the control parameters but can be above 2G for an AMODE 64 caller.

Programming Requirements

The caller must include the HZSQUAA macro to get a mapping for the answer area.

The caller should include the HZSZCONS macro to get equate symbols for the return and reason codes.

The caller must include the HZSPQE macro to get a mapping for some of the subfields within the answer area.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.

The caller may not have an FRR established.

Input Register Information

Before issuing the HZSQUERY macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HZSQUERY macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code, when register 15 is not 0.
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as work registers by the system
15	Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1 Used as work registers by the system

2-13 Unchanged

14-15 Used as work registers by the system

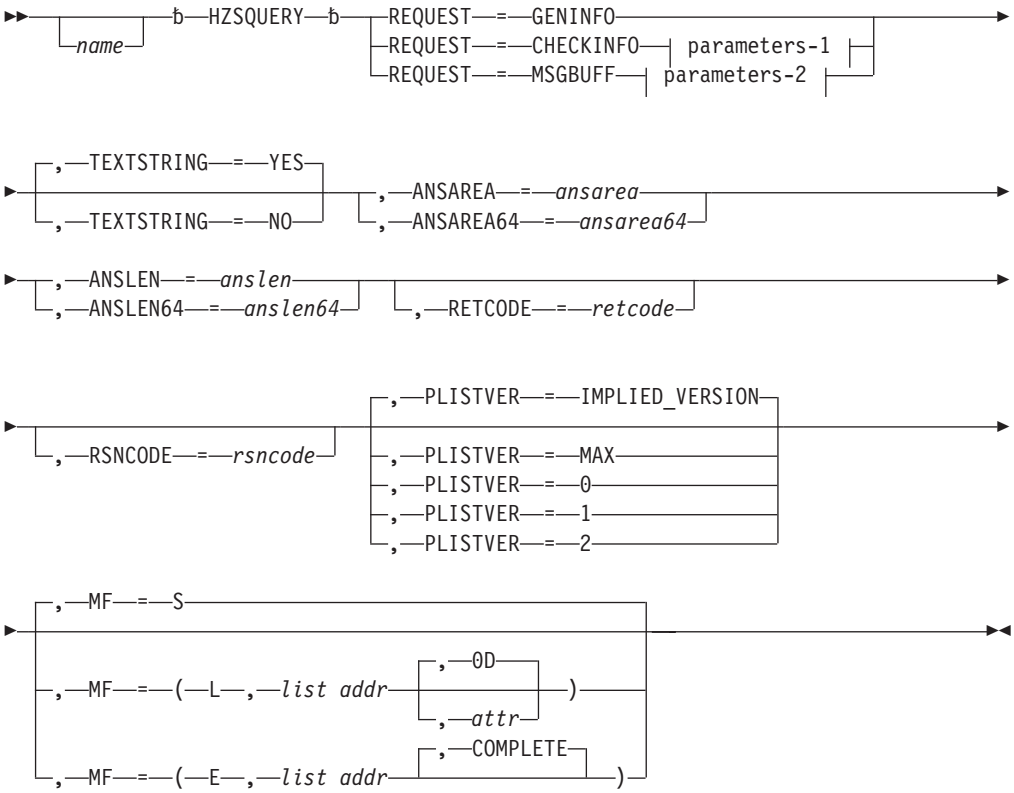
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

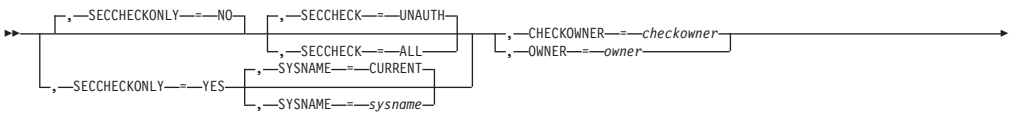
None.

Syntax

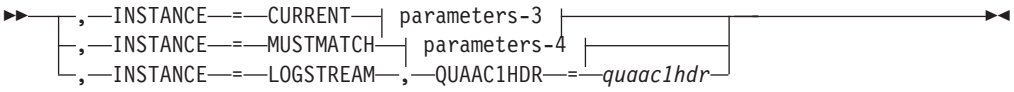
main diagram



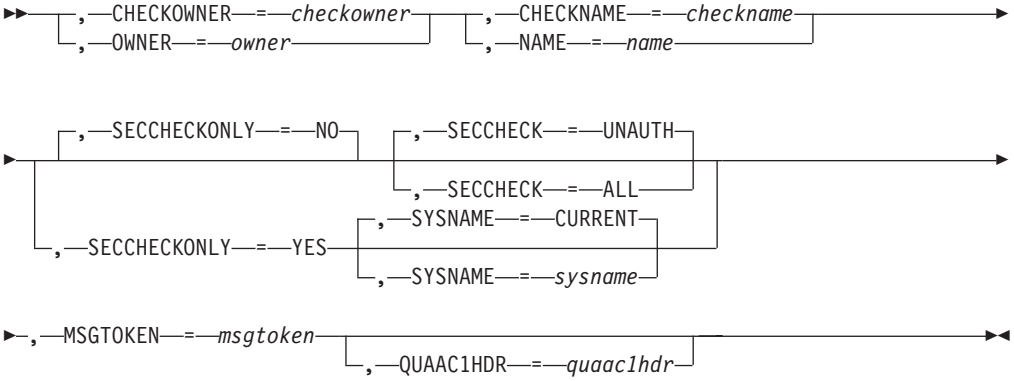
parameters-1



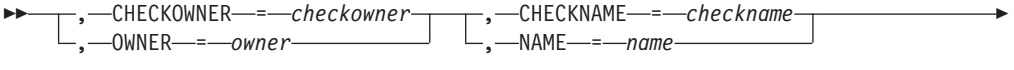
parameters-2

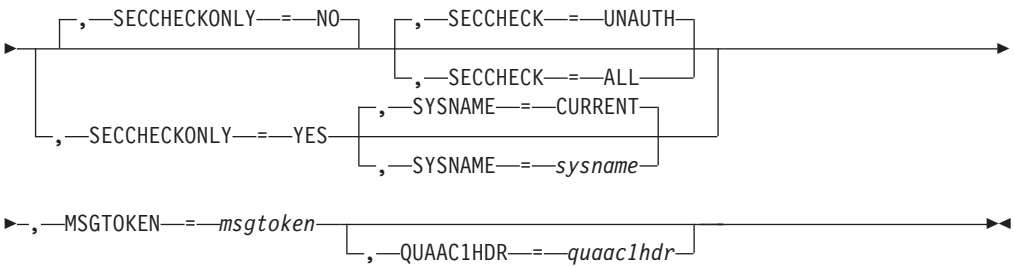


parameters-3



parameters-4





Parameters

The parameters are explained as follows:

name
: An optional symbol, starting in column 1, that is the name on the HZSQUERY macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ANSAREA=*ansarea*
: A required output parameter which is to contain the returned information The area is mapped by macro HZSQUAA. The header area is mapped by DSECT HZSQUAAHDR.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSAREA64=*ansarea64*
: A required output parameter which is to contain the returned information The area is mapped by macro HZSQUAA. This area can be above 2G. The header area is mapped by DSECT HZSQUAAHDR64.

  To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSLEN=*anslen*
: A required input parameter which contains the length of the provided answer area. The length must be at least the value specified by symbol HZSQUERY_MIN_ANSLEN in macro HZSQUAA.

  To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ANSLEN64=*anslen64*
: A required input parameter which contains the length of the provided answer area. The length must be at least the value specified by symbol HZSQUERY_MIN_ANSLEN64 in macro HZSQUAA. It can exceed 2G.

  To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,CATEGORY=*category*
,CATEGORY=ANY_CATEGORY
: When REQUEST=CHECKINFO is specified, an optional input parameter that specifies an array of 1 to 16 contiguous 16 character entries each of which contains a category to be associated with the check. The categories are used as filters. Each category can include wildcard characters. Checks that belong to categories that match according to the rules of the CATRULE parameter and according to the other filtering parameters (OWNER, NAME, and EXITRTN) are processed. The number of categories is specified by the NUMCAT parameter. The default is ANY_CATEGORY.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,CATRULE=DEFAULT
,CATRULE=ONLY
,CATRULE=ANY
,CATRULE=EVERY
,CATRULE=EXCEPT
,CATRULE=VALUE
When CATEGORY=*category* and REQUEST=CHECKINFO are specified, a required parameter that indicates how to process the category filters.

,CATRULE=DEFAULT
indicates to apply the default (which is CATRULE=ONLY).

,CATRULE=ONLY
indicates to match only if all the categories match the categories to which the target check belongs, and if the target check belongs to exactly the number of categories specified by the NUMCAT parameter.

,CATRULE=ANY
indicates to match if any of the categories provided match any of the categories to which the target check belongs.

,CATRULE=EVERY
indicates to match if every one of the categories provided matches any of the categories to which the target check belongs.

,CATRULE=EXCEPT
indicates to match except when one of the categories provided matches any of the categories to which the target check belongs.

,CATRULE=VALUE
Indicates that the value specified by CATRULEVAL is to be used.

,CATRULEVAL=*catruleval*
When CATRULE=VALUE, CATEGORY=*category* and REQUEST=CHECKINFO are specified, a required input parameter that indicates the category rule to be applied. It must be one of the values defined by the xxx_CATRULE_yyy equates generated by HZSQUERY MF=(L,xxx).

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,CHECKNAME=*checkname*
When REQUEST=CHECKINFO is specified, a required input parameter field that identifies the name of the check. If the first character is x'00', or the value is all blanks, information about all checks is returned. Wildcard processing is performed on the name, using the standard wildcard symbols of "*" and "?". The check pattern is delineated by the last non-blank found within the input. Example: A check pattern of "*" indicates to match all checks. Example: A check pattern of "A*" indicates to match all checks with names beginning with "A".

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,CHECKNAME=*checkname*
When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the name of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,CHECKNAME=*checkname*
When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the name of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,CHECKOWNER=*checkowner*
When REQUEST=CHECKINFO is specified, a required input parameter field that identifies the owner of the check. If the first character is x'00', or the value is all blanks, information about checks with all owners is returned. Wildcard processing is performed on the name, using the standard wildcard symbols of "*" and "?". The owner pattern is delineated by the last non-blank found within the input. Example: an owner pattern of "*" indicates to match all owners. Example: an owner pattern of "A*" indicates to match all owners with names beginning with "A".

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CHECKOWNER=*checkowner*
When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the owner of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CHECKOWNER=*checkowner*
When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the owner of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CHECKTYPE=ALL
,CHECKTYPE=NOTDELETED
,CHECKTYPE=DELETED
When REQUEST=CHECKINFO is specified, an optional parameter, of the checks for which information is to be returned. The default is CHECKTYPE=ALL.

,CHECKTYPE=ALL
that indicates that no restrictions are to be made. Return information about checks of any type. The type of the returned check is defined by field Hzsquaac_Type / Hzsquaac1_Type in macro HZSQUAA.

,CHECKTYPE=NOTDELETED
that indicates to return information only about checks that are not deleted and are not delete-pending

,CHECKTYPE=DELETED
that indicates to return information only about checks that are deleted or are delete-pending.

,EXCEPTION=NOTAPPLICABLE
,EXCEPTION=BYDATE

,EXCEPTION=BYSYNCVAL
,EXCEPTION=ALL
When CHECKTYPE=ALL and REQUEST=CHECKINFO are specified, an optional parameter that indicates what policy exception processing to do The default is EXCEPTION=NOTAPPLICABLE.

,EXCEPTION=NOTAPPLICABLE
that indicates that policy exception processing is not applicable to this request

,EXCEPTION=BYDATE
that indicates to find only checks that have policy DATE exceptions, i.e. a policy statement that matches this check was not applied because its DATE was older than the check's DATE.

,EXCEPTION=BYSYNCVAL
that indicates to find only checks that have policy SYNCVAL exceptions, i.e. a policy statement that matches this check was not applied because its SYNCVAL or (E)INTERVAL settings conflicted with the check's values.

,EXCEPTION=ALL
that indicates to find only checks that had any kind of policy exception.

,EXCEPTION=NOTAPPLICABLE
,EXCEPTION=BYDATE
,EXCEPTION=BYSYNCVAL
,EXCEPTION=ALL
When CHECKTYPE=NOTDELETED and REQUEST=CHECKINFO are specified, an optional parameter that indicates what policy exception processing to do The default is EXCEPTION=NOTAPPLICABLE.

,EXCEPTION=NOTAPPLICABLE
that indicates that policy exception processing is not applicable to this request

,EXCEPTION=BYDATE
that indicates to find only checks that have policy DATE exceptions, i.e. a policy statement that matches this check was not applied because its DATE was older than the check's DATE.

,EXCEPTION=BYSYNCVAL
that indicates to find only checks that have policy SYNCVAL exceptions, i.e. a policy statement that matches this check was not applied because its SYNCVAL or (E)INTERVAL settings conflicted with the check's values.

,EXCEPTION=ALL
that indicates to find only checks that had any kind of policy exception.

,EXITRTN=*exitrtn*
,EXITRTN=ANY_EXITRTN
When REQUEST=CHECKINFO is specified, an optional input parameter that identifies the name of the exit routine associated with the check, to be used as a filter. EXITRTN can include wildcard characters. All checks with names that match the specified name and that match the other filtering parameters (OWNER, NAME, CATEGORY) are processed. The default is ANY_EXITRTN.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,GLOBALCHECK=DONOTFIND
,GLOBALCHECK=FINDSYSTEM
When REQUEST=CHECKINFO is specified, an optional parameter that

indicates what to process for global checks. It is relevant only to calls that specify something other than OUTPUTSTYLE=SHORT. The default is GLOBALCHECK=DONOTFIND.

,GLOBALCHECK=DONOTFIND
that indicates not to find the system on which the global check is being run.

,GLOBALCHECK=FINDSYSTEM
that indicates to find the system on which the global check is to be run. Field PQE_GlobalCheck_SYSNAME contains the name of that system, or zeroes if no system is currently tagged to run that check.

,INSTANCE=CURRENT
,INSTANCE=MUSTMATCH
,INSTANCE=LOGSTREAM
When REQUEST=MSGBUFF is specified, a required parameter that indicates how to compare the instance of the check designated by the MSGTOKEN parameter to the instance of the check.

,INSTANCE=CURRENT
indicates to return the message buffer(s) for the current instance of the check, and set bit HzsquaaH_MsgBuffWrongInstance / HzsquaaH64_MsgBuffWrongInstance when the instance of the check designated by the MSGTOKEN parameter is not the current instance.

,INSTANCE=MUSTMATCH
indicates to return data only if the message buffer(s) for the instance of the check designated by the MSGTOKEN parameter are available. They might not be available if the instance is not the current instance.

,INSTANCE=LOGSTREAM
Indicates that the message data to be returned is to be found within the logstream. You would use this when you want the information from a previous iteration of a check. Note that if you are basing this request on the "current" information returned from a previous call that specified INSTANCE=CURRENT or INSTANCE=MUSTMATCH, it is possible that the logstream data that you receive back will be the same information that was returned for that previous call. This could happen if the message buffer has already been written to the logstream and no new iteration of the check has yet begun. Compare the Hcklog_CheckHasRunCount fields of the two calls and if they are identical, this has happened (and you would just call "again" using the data returned on this call).

Also of note is that if the first Hcklog area does not have a value of 1 in field HckLog_BufNum, it means that this query was done while the check's data was being written to the logstream. The data retrieved is a subset of the data that INSTANCE=CURRENT or INSTANCE=MUSTMATCH would have returned. You can use the information returned within the QUAAC1HDR parameter to reference the check instance prior to this one, on a subsequent HZSQUERY call. Bit HZSQUAAH_MsgbuffIncomplete / HZSQUAAH64_MsgbuffIncomplete will als be on if this has occurred.

The system used the IXGCONN REQUEST=CONNECT service to access the logstream. The caller of HZSQUERY must satisfy the System Authorization Facility (SAF) checks that are performed by that service. Refer to the documentation of IXGCONN for additional information.

,LOCALE=ANY

,LOCALE=HZSPROC
,LOCALE=REMOTE
When REQUEST=CHECKINFO is specified, an optional parameter, which identifies the locale of the check. The default is LOCALE=ANY.

,LOCALE=ANY
indicates that the check can be of any locale (hzsproc, remote or REXX)

,LOCALE=HZSPROC
indicates that the check must be of locale HZSPROC (i.e., runs in the IBM Health Checker for z/OS address space start by hzsproc).

,LOCALE=REMOTE
indicates that the check is remote (i.e., does not run in the IBM Health Checker for z/OS address space start).

,MF=S
,MF=(L,*list addr***)**
,MF=(L,*list addr***,***attr***)**
,MF=(L,*list addr***,0D)**
,MF=(E,*list addr***)**
,MF=(E,*list addr***,COMPLETE)**
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,*list addr*
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

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

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MSGTOKEN=*msgtoken*
When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, a required input parameter, field that is the message token returned by a previous HZSQUERY request.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,MSGTOKEN=*msgtoken*
When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, a required input parameter, field that is the message token returned by a previous HZSQUERY request.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,NAME=*name*
When REQUEST=CHECKINFO is specified, a required input parameter field that identifies the name of the check. If the first character is x'00', or the value is all blanks, information about all checks is returned. Wildcard processing is performed on the name, using the standard wildcard symbols of "*" and "?". The check pattern is delineated by the last non-blank found within the input. Example: A check pattern of "*" indicates to match all checks. Example: A check pattern of "A*" indicates to match all checks with names beginning with "A".

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,NAME=*name*
When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the name of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,NAME=*name*
When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the name of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,NUMCAT=*numcat*
When CATEGORY=*category* and REQUEST=CHECKINFO are specified, a required input parameter that indicates how many categories are contained in the array specified by the CATEGORY parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,OUTPUTSTYLE=FULL
,OUTPUTSTYLE=NO_CHKWORK
,OUTPUTSTYLE=SHORT
,OUTPUTSTYLE=VALUE
When REQUEST=CHECKINFO is specified, an optional parameter that indicates the style of output The default is OUTPUTSTYLE=FULL.

,OUTPUTSTYLE=FULL
that indicates to return the full amount of data, each HzsquaaCData / HzsquaaC1Data entry mapped by macro HZSPQE or HZSDPQE.

,OUTPUTSTYLE=NO_CHKWORK
that indicates to return the full amount of data except for the x'800' bytes of the PQE_CHKWORK area. Each HzsquaaCData / HzsquaaC1Data entry mapped by macro HZSPQE (up to the PQE_CHKWORK field) or HZSDPQE.

,OUTPUTSTYLE=SHORT
that indicates to return the "short form", each HzsquaaCData /HzsquaaC1Data entry mapped by DSECT HZSQUAACS in macro HZSQUAA.

,OUTPUTSTYLE=VALUE
indicates that the value specified by OUTPUTSTYLEV is to be used.

,OUTPUTSTYLEV=*outputstylev*
When OUTPUTSTYLE=VALUE and REQUEST=CHECKINFO are specified, a required input parameter that indicates the output style to be used. It must be one of the values defined by the xxx_OUTPUTSTYLE_yyy equates generated by HZSQUERY MF=(L,xxx).

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,OWNER=*owner*
When REQUEST=CHECKINFO is specified, a required input parameter field that identifies the owner of the check. If the first character is x'00', or the value is all blanks, information about checks with all owners is returned. Wildcard processing is performed on the name, using the standard wildcard symbols of "*" and "?". The owner pattern is delineated by the last non-blank found within the input. Example: an owner pattern of "*" indicates to match all owners. Example: an owner pattern of "A*" indicates to match all owners with names beginning with "A".

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNER=*owner*
When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the owner of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNER=*owner*
When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, a required input parameter field that identifies the owner of the check. No Wildcard processing is performed on the name.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
,PLISTVER=2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, which supports all parameters except those specifically referenced in higher versions.
- **1**, which supports both the following parameters and those from version 0:
  - POLICYNAME
- **2**, which supports both the following parameters and those from version 0 and 1:
  - ANSAREA64
  - ANSLEN64
  - QUAAC1HDR

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, or 2

,QUAACVER=0
,QUAACVER=1

When REQUEST=CHECKINFO is specified, an optional parameter that indicates the format of information to be returned for REQUEST=CHECKINFO, as mapped by DSECTs within HZSQUAA. Quaacver=1 returns more information than Quaacver=0. The default is QUAACVER=0.

,QUAACVER=0

The header information for each CHECKINFO entry is mapped by DSECT HZSQUAAC.

,QUAACVER=1

The header information for each CHECKINFO entry is mapped by DSECT HZSQUAAC1. Note that when you specify, or conditionally plan to specify on a subsequent invocation, INSTANCE=LOGSTREAM, you must specify QUAACVER=1 in order to get back the proper data for the INSTANCE=LOGSTREAM specification.

,QUAAC1HDR=*quaac1hdr*

When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, an optional output parameter, area mapped by HZSQUAAC1 field HZSQUAAC1HDR that is to returned. This area identifies the next older check iteration MSGBUFF should the caller wish to retrieve that information using HZSQUERY with INSTANCE=LOGSTREAM.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,QUAAC1HDR=*quaac1hdr*

When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, an optional output parameter, area mapped by HZSQUAAC1 field HZSQUAAC1HDR that is to returned. This area identifies the next older check

iteration MSGBUFF should the caller wish to retrieve that information using HZSQUERY with INSTANCE=LOGSTREAM.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,QUAAC1HDR=*quaac1hdr*
When INSTANCE=LOGSTREAM and REQUEST=MSGBUFF are specified, a required input/output parameter, area mapped by HZSQUAAC1 field HZSQUAAC1HDR that was returned in the answer area of a previous HZSQUERY request that specified either REQUEST=CHECKINFO with QUAACVER=1 or REQUEST=MSGBUFF with the QUAAC1HDR parameter. If that previous HZSQUERY was REQUEST=CHECKINFO, the HZSQUAAC1HDR area is part of the HZSQUAAC1 area of the answer area. If that previous HZSQUERY was REQUEST=MSGBUFF, the HZSQUAAC1HDR area was returned via the QUAAC1HDR parameter. On input, this area identifies the instance of the MSGBUFF for which data is to be returned. On output, this area identifies the next older check iteration MSGBUFF should the caller wish to retrieve that information.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

REQUEST=GENINFO
REQUEST=CHECKINFO
REQUEST=MSGBUFF
A required parameter, which identifies the type of request.

REQUEST=GENINFO
Get general information about IBM Health Checker for z/OS. This includes the procedure used to start it, and the started task identifier assigned to it. The returned information consists of

- a header area (mapped by DSECT HZSQUAAHDR or DSECT HZSQUAAHDR64 in macro HZSQUAA, depending on whether you use the ANSLEN or ANSLEN64 parameter), which contains the procedure used to start IBM Health Checker for z/OS, the started task identifier, and the logstream name, as well as a value of one in field HzsquaahNumQuaaG / Hzsquaah64NumQuaaG indicating that there is one entry provided, with the address of that entry being in field HzsquaahQuaaGAddr / Hzsquaah64QuaaGAddr.
- the entry (mapped by DSECT HZSQUAAG in macro HZSQUAA)

REQUEST=CHECKINFO
Get information about the specified check. The information consists of

- a header area (mapped by DSECT HZSQUAAHDR or DSECT HZSQUAAHDR64 in macro HZSQUAA, depending on whether you use the ANSLEN or ANSLEN64 parameter), which contains the number of entries that follows (HzsquaahNumQuaaC / Hzsquaah64NumQuaaC) and the address of the first entry (HzsquaahQuaaCAddr / HzsquaahQuaaCAddr).
- entries (mapped by DSECT HZSQUAAC / HZSQUAAC1 in macro HZSQUAA) each of which has a field that indicates the length of that entry (HzsquaaCLen / HzsquaaC1Len). The length field should be added to the address of an entry to get the address of the next entry.

REQUEST=MSGBUFF
Get information about the message buffer(s) specified by the input

MSGTOKEN. That MSGTOKEN would have been returned on a previous HZSQUERY request in field HzsquaaCMsgToken. The information consists of

- a header area (mapped by DSECT HZSQUAAHDR or DSECT HZSQUAAHDR64 in macro HZSQUAA, depending on whether you use the ANSLEN or ANSLEN64 parameter), which contains the number of entries that follows (HzsquaahNumHCKL / Hzsquaah64NumHCKL) and the address of the first entry (HzsquaahHcklAddr / Hzsquah64HcklAddr).
- entries (mapped by DSECT HZSLOG in macro HZSZHCKL) each of which has a field that indicates the length of that entry (Hcklog_BufLen). The length field should be added to the address of an entry to get the address of the next entry.

,RESULT=ANY
,RESULT=EXCEPTIONS
When CHECKTYPE=ALL and REQUEST=CHECKINFO are specified, an optional parameter that indicates what result processing to do The default is RESULT=ANY.

,RESULT=ANY
that indicates that any check result is applicable to this request

,RESULT=EXCEPTIONS
that indicates to find only checks that detected exception(s). Note that DELETED checks are not considered to have detected exception(s).

,RESULT=ANY
,RESULT=EXCEPTIONS
When CHECKTYPE=NOTDELETED and REQUEST=CHECKINFO are specified, an optional parameter that indicates what result processing to do The default is RESULT=ANY.

,RESULT=ANY
that indicates that any check result is applicable to this request

,RESULT=EXCEPTIONS
that indicates to find only checks that detected exception(s).

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,REXX=ANY
,REXX=NO
,REXX=YES
When LOCALE=REMOTE and REQUEST=CHECKINFO are specified, an optional parameter, which indicates if this is a REXX check. The default is REXX=ANY.

,REXX=ANY
indicates that the check can either be a REXX check or not.

,REXX=NO
indicates that the check is not a REXX check.

,REXX=YES
indicates that the check is a REXX check.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SECCHECK=UNAUTH
,SECCHECK=ALL
When SECCHECKONLY=NO and REQUEST=CHECKINFO are specified, an optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases. If RACF does not grant authority, the request is rejected.

,SECCHECK=UNAUTH
,SECCHECK=ALL
When SECCHECKONLY=NO, INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, an optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases. If RACF does not grant authority, the request is rejected.

,SECCHECK=UNAUTH
,SECCHECK=ALL
When SECCHECKONLY=NO, INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, an optional parameter that indicates whether to do RACF security checks. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases. If RACF does not grant authority, the request is rejected.

,SECCHECKONLY=NO
,SECCHECKONLY=YES
When REQUEST=CHECKINFO is specified, an optional parameter that indicates whether to do full processing or only security checks The default is SECCHECKONLY=NO.

,SECCHECKONLY=NO
that indicates to do full processing.

,SECCHECKONLY=YES
that indicates to do only the security check to see if the requesting user has RACF authority to access the data. When this option is specified, the security check is done regardless of the caller's key or state.

,SECCHECKONLY=NO
,SECCHECKONLY=YES
When INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, an optional parameter that indicates whether to do full processing or only security checks The default is SECCHECKONLY=NO.

,SECCHECKONLY=NO
that indicates to do full processing.

,SECCHECKONLY=YES
that indicates to do only the security check to see if the requesting user has RACF authority to access the data. When this option is specified, the security check is done regardless of the caller's key or state.

,SECCHECKONLY=NO
,SECCHECKONLY=YES
When INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, an optional parameter that indicates whether to do full processing or only security checks The default is SECCHECKONLY=NO.

,SECCHECKONLY=NO
that indicates to do full processing.

,SECCHECKONLY=YES
that indicates to do only the security check to see if the requesting user has RACF authority to access the data. When this option is specified, the security check is done regardless of the caller's key or state.

,SYSNAME=*sysname*
,SYSNAME=CURRENT
When SECCHECKONLY=YES and REQUEST=CHECKINFO are specified, an optional input parameter that specifies the system name to be used when doing the security check. Note that this specification is used only when the caller is supervisor state, system key, or APF-authorized. The default is CURRENT. which indicates to use the name of the system on which this request was issued.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,SYSNAME=*sysname*
,SYSNAME=CURRENT
When SECCHECKONLY=YES, INSTANCE=CURRENT and REQUEST=MSGBUFF are specified, an optional input parameter that specifies the system name to be used when doing the security check. Note that this specification is used only when the caller is supervisor state, system key, or APF-authorized. The default is CURRENT. which indicates to use the name of the system on which this request was issued.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,SYSNAME=*sysname*
,SYSNAME=CURRENT
When SECCHECKONLY=YES, INSTANCE=MUSTMATCH and REQUEST=MSGBUFF are specified, an optional input parameter that specifies

the system name to be used when doing the security check. Note that this specification is used only when the caller is supervisor state, system key, or APF-authorized. The default is CURRENT. which indicates to use the name of the system on which this request was issued.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,TEXTSTRING=YES
,TEXTSTRING=NO
An optional parameter that indicates whether to return the "text strings" mapped in HZSPQE for options that return data within the HZSPQE. It is relevant only to REQUEST=CHKINFO calls that specify something other than OUTPUTSTYLE=SHORT. The default is TEXTSTRING=YES.

,TEXTSTRING=YES
that indicates to return the HZSPQE "text strings".

,TEXTSTRING=NO
that indicates not to return the "text strings". If not using the HZSPQE output for displaying, specifying "NO" avoids setting some fields that you might not need.

ABEND Codes

058 The IBM Health Checker for z/OS address space terminated while this call was in process.

Return and Reason Codes

When the HZSQUERY macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HZSZCONS provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 44. Return and Reason Codes for the HZSQUERY Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: HzsqueryRc_OK **Meaning**: Requested information returned **Action**: None required
4	—	**Equate Symbol**: HzsqueryRc_Warn **Meaning**: Warning **Action**: Refer to action under the individual reason code.
4	xxxx0401	**Equate Symbol**: HzsqueryRsn_NotAllDataReturned **Meaning**: Not all data was returned because the answer area is not big enough. Answer area field HZSQUAAHTLEN /HZSQUAAH64TLEN indicates how much space is currently required. **Action**: Allocate a larger area and request the function again.

Table 44. Return and Reason Codes for the HZSQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	—	**Equate Symbol**: HzsqueryRc_InvParm **Meaning**: HZSQUERY request specifies incorrect parameters. **Action**: Refer to action under the individual reason code.
8	xxxx0801	**Equate Symbol**: HzsqueryRsn_NotAuthorized **Meaning**: Caller is not authorized. For INSTANCE=LOGSTREAM, the first eight bytes of the DIAG area in the header (HZSQUAAHDIAG or HZSQUAAH64DIAG) contain the four-byte return code and four-byte reason code from the IXGCONN service. **Action**: Avoid calling HZSQUERY when not authorized
8	xxxx0818	**Equate Symbol**: HzsqueryRsn_BadParmlist **Meaning**: Error accessing parameter list. **Action**: Make sure that the provided parameter list is valid.
8	xxxx0838	**Equate Symbol**: HzsqueryRsn_BadParmListVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Health Checker for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Health Checker for z/OS.
8	xxxx0843	**Equate Symbol**: HzsqueryRsn_SrbMode **Meaning**: SRB mode. **Action**: Avoid issuing HZSQUERY in SRB mode.
8	xxxx0844	**Equate Symbol**: HzsqueryRsn_NotEnabled **Meaning**: Not Enabled. **Action**: Avoid using HZSQUERY when not enabled.
8	xxxx0845	**Equate Symbol**: HzsqueryRsn_Locked **Meaning**: Locked **Action**: Avoid using HZSQUERY when a lock is held.
8	xxxx0846	**Equate Symbol**: HzsqueryRsn_FRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using HZSQUERY when an EUT FRR is established.
8	xxxx0847	**Equate Symbol**: HzsqueryRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. The access register might not have been set up correctly.

Table 44. Return and Reason Codes for the HZSQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0848	**Equate Symbol**: HzsqueryRsn_BadAnsAreaALET **Meaning**: Bad answer area ALET. **Action**: Make sure that the ALET associated with the answer area is valid. The access register might not have been set up correctly.
8	xxxx0849	**Equate Symbol**: HzsqueryRsn_BadAnsArea **Meaning**: Error accessing answer area. **Action**: Make sure that the provided answer area is valid.
8	xxxx084A	**Equate Symbol**: HzsqueryRsn_BadAnsLen **Meaning**: AnsLen is less than size of the header area. **Action**: Provide a larger answer area (as indicated by the ANSLEN keyword).
8	xxxx084B	**Equate Symbol**: HzsqueryRsn_BadParmlistValue **Meaning**: A parameter list field contains an unsupported value. **Action**: Check for possible storage overlay
8	xxxx084C	**Equate Symbol**: HzsqueryRsn_BadCategoryALET **Meaning**: Bad category ALET. **Action**: Make sure that the ALET associated with the category area is valid. The access register might not have been set up correctly.
8	xxxx084D	**Equate Symbol**: HzsqueryRsn_BadCategory **Meaning**: Error accessing category area. **Action**: Make sure that the provided category area is valid.
8	xxxx084E	**Equate Symbol**: HzsqueryRsn_MsgTokenNotValid **Meaning**: MSGTOKEN is not valid. **Action**: Make sure that the MSGTOKEN specifies a value returned by HZSQUERY. As that might represent a check that no longer exists, it might be necessary to re-issue HZSQUERY to get a new MSGTOKEN.
8	xxxx085C	**Equate Symbol**: HzsqueryRsn_XM **Meaning**: For INSTANCE=LOGSTREAM, a cross-memory environment exists. **Action**: Avoid using HZSQUERY INSTANCE=LOGSTREAM when the primary address space does not match the home address space.
8	xxxx085D	**Equate Symbol**: HzsqueryRsn_BadQUAAC1HDRALET **Meaning**: Bad QUAAC1HDR ALET. **Action**: Make sure that the ALET associated with the QUAAC1HDR area is valid. The access register might not have been set up correctly.
8	xxxx085E	**Equate Symbol**: HzsqueryRsn_BadQUAAC1HDR **Meaning**: Error accessing QUAAC1HDR area. **Action**: Make sure that the provided QUAAC1HDR area is valid.

Table 44. Return and Reason Codes for the HZSQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
0C	—	**Equate Symbol**: HzsqueryRc_EnvError **Meaning**: Environmental Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: HzsqueryRsn_IBMHCNotActive **Meaning**: IBM Health Checker for z/OS is not active **Action**: Re-issue the request when the service is available
0C	xxxx0C21	**Equate Symbol**: HzsqueryRsn_LogstreamRecordNotFound **Meaning**: The requested record within the logstream specified within the QUAAC1HDR area could not be found. The requested data could not be retrieved. The first eight bytes of the DIAG area in the header (HZSQUAAHDIAG or HZSQUAAH64DIAG) contain the four-byte return code and four-byte reason code from the IXGBRWSE service. **Action**: Avoid calling HZSQUERY when the BlockID returned within the QUAAC1HDR area is 0. If the BlockID was not 0, notify the system programmer.
0C	xxxx0C22	**Equate Symbol**: HzsqueryRsn_LogstreamGap **Meaning**: A gap was detected in the logstream specified within the QUAAC1HDR area. The requested data could not be retrieved. The first eight bytes of the DIAG area in the header (HZSQUAAHDIAG or HZSQUAAH64DIAG) contain the four-byte return code and four-byte reason code from the IXGBRWSE service. **Action**: Notify the system programmer.
0C	xxxx0C23	**Equate Symbol**: HzsqueryRsn_LogstreamLossOfData **Meaning**: A loss of data was detected in the logstream specified within the QUAAC1HDR area. The system received reason code IxgRsnCodeWarningLossOfData when attempting to browse the logstream. The requested data could not be retrieved. The first eight bytes of the DIAG area in the header (HZSQUAAHDIAG or HZSQUAAH64DIAG) contain the four-byte return code and four-byte reason code from the IXGBRWSE service. **Action**: Notify the system programmer.
0C	xxxx0C24	**Equate Symbol**: HzsqueryRsn_LogstreamError **Meaning**: The system received an unexpected return / reason code from a system logger function. The requested data could not be retrieved. The first eight bytes of the DIAG area in the header (HZSQUAAHDIAG or HZSQUAAH64DIAG) contain the four-byte return code and four-byte reason code from the IXGBRWSE service. **Action**: Notify the system programmer.

Table 44. Return and Reason Codes for the HZSQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
0C	xxxx0C25	**Equate Symbol**: HzsqueryRsn_LogstreamBadData **Meaning**: The data retrieved from the logstream specified within the QUAAC1HDR area was not valid. **Action**: Notify the system programmer.
0C	xxxx0C26	**Equate Symbol**: HzsqueryRsn_StorageNotAvailable **Meaning**: The system could not obtain working storage needed to process the request. **Action**: Try re-running the job with a larger region size.
0C	xxxx0C27	**Equate Symbol**: HzsqueryRsn_BadLogstream **Meaning**: The system could not connect to the logstream specified within the QUAAC1HDR area. The first eight bytes of the DIAG area in the header (HZSQUAAHDIAG or HZSQUAAH64DIAG) contain the four-byte return code and four-byte reason code from the IXGCONN service. **Action**: Make sure that the area has been properly initialized and that the logstream data set is accessible. Make sure that the system logger is active.
10	—	**Equate Symbol**: HzsqueryRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.
10	xxxx1001	**Equate Symbol**: HzsqueryRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer

Examples

None.

Part 3. Check Descriptions

Chapter 13. IBM Health Checker for z/OS checks

This chapter describes the checks supplied with IBM Health Checker for z/OS.

We'll be adding more checks to IBM Health Checker for z/OS periodically, both as APARs and integrated into z/OS. For the most up-to-date information on checks available, see the following Web site:

`http://www.ibm.com/servers/eserver/zseries/zos/hchecker/check_table.html`

For check output messages, see the component message books or use message explanations directly from the **LookAt** Web site at http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/. Because checks along with their output messages might be added by PTF between releases of component message books, LookAt will contain the most up to date message information. For more information about using LookAt to find check messages, see "Finding check message documentation with LookAt" on page 41.

All checks are **local** checks (run in the IBM Health Checker for z/OS address space) unless otherwise noted.

Where are the migration checks?

We have listed all the migration checks with their individual components. You can identify migration checks by the phrase ZOSMIG (or simply MIG for ICSF checks) in the check name. You can find the migration checks in one of the following ways:

- Use the list of checks shown in the partial table of contents in Part 3, "Check Descriptions," on page 379.
- If you are using a PDF version of this book, do a search on the phrase ZOSMIG to find all but the ICSF migration checks, or search on MIG (case sensitive) to find all the migration checks.

Allocation checks (IBMALLOC)

ALLOC_ALLC_OFFLN_POLICY

Description:
Checks the value of the ALLC_OFFLN POLICY in the current Allocation settings.

Reason for check:
This check ensure that the best ALLC_OFFLN POLICY allocation setting for the particular environment is set. Certain ALLC_OFFLN POLICY values can result in a deadlock.

z/OS releases the check applies to:
z/OS V1R13 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMALLOC,ALLOC_ALLC_OFFLN_POLICY)
 SEVERITY(LOW)
 INTERVAL(24:00)
 DATE('date_of_the_change')
 PARM('POLICY(WTOR)')
 REASON('The reason for the change.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes. the following parameters are accepted indicating the value of ALLC_OFFLN POLICY which the user wants as the recommended setting:

- PARM('POLICY(WTOR)')
- PARM('POLICY(CANCEL)')
- PARM('POLICY(WAITHOLD)')
- PARM('POLICY(WAITNOH)')

The default is PARM('POLICY(WTOR,CANCEL,WAITNOH)').

You can also specify more than one acceptable value by separating them via commas as such: PARM('POLICY(*val1*,*val2*...*valn*)'), where the values correspond to the different values of the ALLC_OFFLN POLICY.If any of these values match the current setting, then the check will be successful. If the checks finds that current ALLC_OFFLN POLICY setting does not match the parameter, the check issues an exception.

Reference:
For additional information, see the ALLOCxx chapter*z/OS MVS Initialization and Tuning Guide*e .

Messages:
This check issues the following messages:

- IEFAH001I (successful)
- IEFAH002E (exception)

See IEFA messages in *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:
When successful, the report that the check produces is shown below:

```
CHECK(IBMALLOC,ALLOC_ALLC_OFFLN_POLICY)
START TIME: 09/24/2010 16:11:40.728961
CHECK DATE: 20100830  CHECK SEVERITY: LOW
CHECK PARM: POLICY(WTOR,CANCEL,WAITNOH)


IEFAH001I Option value WTOR matches the owner
recommendation of WTOR,CANCEL,WAITNOH

END TIME: 09/24/2010 16:11:40.729241  STATUS: SUCCESSFUL
```

For an exception, the report that the check produces is shown below:

```
CHECK(IBMALLOC,ALLOC_ALLC_OFFLN_POLICY)
START TIME: 09/24/2010 16:11:40.728961
CHECK DATE: 20100830  CHECK SEVERITY: LOW
CHECK PARM: POLICY(WTOR,CANCEL,WAITNOH)


* Low Severity Exception *

IEFAH002E Option value WAITHOLD does not match the owner
recommendation of WTOR,CANCEL,WAITNOH

END TIME: 09/24/2010 16:11:40.729241  STATUS: EXCEPTION-LOW
```

ALLOC_SPEC_WAIT_POLICY

Description:
Checks the value of the SPEC_WAIT POLICY in the current Allocation settings.

Reason for check:
This check ensure that the best SPEC_WAIT POLICY allocation setting for the particular environment is set. Certain SPEC_WAIT POLICY values can result in a deadlock.

z/OS releases the check applies to:
z/OS V1R13 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMALLOC,ALLOC_SPEC_WAIT_POLICY )
 ACTIVE
 SEVERITY(LOW)
 INTERVAL(24:00)
 DATE('date_of_the_change')
 PARM('POLICY(WTOR)')
 REASON('The reason for the change.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes. the following parameters are accepted to indicate the value of SPEC_WAIT POLICY the user wants as the recommended setting:

- PARM('POLICY(WTOR)')
- PARM('POLICY(CANCEL)')
- PARM('POLICY(WAITHOLD)')
- PARM('POLICY(WAITNOH)')

The default is PARM('POLICY(WTOR,CANCEL,WAITNOH)').

You can also specify more than one acceptable value by separating them via commas as such: PARM('POLICY(*val1,val2...valn*)'), where the values correspond to the different values of the SPEC_WAIT POLICY.If any of these values match the current setting, then the check will be successful. If the checks finds that current SPEC_WAIT POLICYYsetting does not match the parameter, the check issues an exception.

Reference:
For additional information, see the ALLOCxx chapterz/*OS MVS Initialization and Tuning Guide* .

Messages:
This check issues the following messages:

- IEFAH001I (successful)
- IEFAH002E (exception)

See IEFA messages in *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:
When successful, the report that the check produces is shown below:

```
CHECK(IBMALLOC,ALLOC_SPEC_WAIT_POLICY)
START TIME: 10/25/2010 11:17:41.822822
CHECK DATE: 20100831  CHECK SEVERITY: LOW
CHECK PARM: POLICY(WTOR,CANCEL,WAITNOH)


IEFAH001I Option value WTOR matches the owner
recommendation of WTOR,CANCEL,WAITNOH

END TIME: 10/25/2010 11:17:41.825402  STATUS: SUCCESSFUL
```

For an exception, the report that the check produces is shown below:

```
CHECK(IBMALLOC,ALLOC_SPEC_OFFLN_POLICY)
START TIME: 10/25/2010 12:15:43.753021
CHECK DATE: 20100831  CHECK SEVERITY: LOW
CHECK PARM: POLICY(WTOR,CANCEL,WAITNOH)


* Low Severity Exception *

IEFAH002E Option value WAITHOLD does not match the owner
recommendation of WTOR,CANCEL,WAITNOH

END TIME: 09/24/2010 16:11:40.729241  STATUS: EXCEPTION-LOW
```

ALLOC_TIOT_SIZE

Description:
Checks the value of the TIOT SIZE in the current Allocation settings.

Reason for check:
This check ensure that the best TIOT SIZE value for the particular environment is set. Since the size of the TIOT is directly related to how many DD statements can be specified per job step, jobs may fail if there are more DDs than space for them in the TIOT.

z/OS releases the check applies to:
z/OS V1R13 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMALLOC,ALLOC_TIOT_SIZE )
 ACTIVE
 SEVERITY(LOW)
 INTERVAL(24:00)
 DATE('date_of_the_change')
 PARM('SIZE(24,32)')
 REASON('The reason for the change.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes. The following parameter lets you specify a customized range for the TIOT SIZE values that is optimal for your installation:

PARM('SIZE(min,max)')
These range values must be integers between 16 and 64. To specify only one acceptable value, specify identical min and max values.

The default is PARM('SIZE(24,32)')

The default is PARM('SIZE(32,64)').

Reference:
For additional information, see the ALLOCxx chapter*z/OS MVS Initialization and Tuning Guide* .

Messages:
This check issues the following messages:

- IEFAH001I (successful)
- IEFAH002E (exception)

See IEFA messages in *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:
When successful, the report that ALLOC_TIOT_SIZE produces is shown below:

```
CHECK(IBMALLOC,ALLOC_TIOT_SIZE)
START TIME: 09/24/2010 16:11:40.728961
CHECK DATE: 20100830  CHECK SEVERITY: LOW
CHECK PARM: SIZE(32,64)


IEFAH001I Option value 32 matches the owner
recommendation of 32-64

END TIME: 09/24/2010 16:11:40.729241  STATUS: SUCCESSFUL
```

For an exception, the report that ALLOC_TIOT_SIZE produces is shown below:

```
CHECK(IBMALLOC,ALLOC_TIOT_SIZE)
START TIME: 09/24/2010 16:11:40.728961
CHECK DATE: 20100830  CHECK SEVERITY: LOW
CHECK PARM: SIZE(32,64)


* Low Severity Exception *
```

```
IEFAH002E Option value 24 does not match the owner
recommendation of 32-64

END TIME: 09/24/2010 16:11:40.729241  STATUS: EXCEPTION-LOW
```

ASM checks (IBMASM)

ASM_NUMBER_LOCAL_DATASETS

Description:
Checks on the number of usable local page data sets. The check generates an exception if the number is below the recommended value of 3. This is a one-time check that is also run whenever a page data set is dynamically added or deleted.

Reason for check:
Running with a sufficient number of usable paging data sets ensures that paging I/O is distributed over multiple devices which enhances paging throughput.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMASM,ASM_NUMBER_LOCAL_DATASETS),
      INTERVAL(ONETIME),
      SEVERITY(LOW),
      PARM('MINLOCALS(3)'),
      DATE('date_of_the_change')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, in keyword MINLOCALS, which is an integer, 1-255, indicating the recommended minimum number of usable local page data sets. The default is 3.

Reference:
For information on auxiliary storage management, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
This check issues the following exception messages:

- ILRH0101E

See the ILRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ASM_PAGE_ADD

Description:
Checks on the ability to dynamically add additional paging data sets via the PAGEADD command. The check generates an exception if the number of paging data sets that can be added is at or below the warning value of 2. This is a one-time check that is also run whenever a page data set is dynamically added or deleted.

Reason for check:
Specifying an appropriate PAGTOTL value (IEASYSxx) allows for the paging data sets that are defined at IPL time, and allows room for expansion if additional data sets are subsequently needed.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMASM,ASM_PAGE_ADD),
      INTERVAL(ONETIME),
      SEVERITY(MED),
      PARM('MINADDS(2)'),
      DATE('date_of_the_change')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, in keyword MINADDS, which is an integer, 1-255, indicating the the recommended minimum number of paging data sets that can be dynamically added. The default is MINADDS(2).

Reference:

- For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .
- For information on the PAGTOTL parameter, see *z/OS MVS Initialization and Tuning Reference*.
- For information on the PAGEADD command, see *z/OS MVS System Commands*.

Messages:
This check issues the following exception messages:

- ILRH0103E

See the ILRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ASM_PLPA_COMMON_SIZE

Description:
Checks on the combined size of the PLPA and Common page data sets in relation to the size of CSA/ECSA and PLPA/EPLPA . The check generates an exception if the PLPA and Common page data sets size can not accommodate the percentage (identified by the warning threshold) of the slots required for all CSA/ECSA and PLPA/EPLPA.

Reason for check:
You should define the PLPA and Common page data sets based on the size of CSA/ECSA and PLPA/EPLPA, if known.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMASM,ASM_PLPA_COMMON_SIZE),
      INTERVAL(ONETIME),
      PARM('THRESHOLD(100%)'),
      DATE('date_of_the_change')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, as follows:

- When not using dynamic severity: THRESHOLD(*value*),
- When using dynamic severity: THRESHOLD_HIGH(*value*), THRESHOLD_MED(*value*), THRESHOLD_LOW(*value*), THRESHOLD_NONE(*value*)

  Note that the thresholds specified should decrease in value as the severity increases, since a lower percentage indicates a more severe condition (the PLPA and COMMON paging data sets are able to accommodate a smaller amount of pages), as shown in the following example:

  ```
  PARM('THRESHOLD_HIGH(50%),THRESHOLD_MED(80%), THRESHOLD_LOW(100%)')
  ```

The variable for the parameters is as follows:

value
An integer, 0-100 indicating the warning threshold percent. It may optionally be followed by %.

Default: THRESHOLD(100%)

You can use synonyms for these parameters, as follows:

Parameter	Synonyms
THRESHOLD_HIGH	• THRESHOLD_HI • THRESHOLD_H

Parameter	Synonyms
THRESHOLD_MED	• THRESHOLD_M
THRESHOLD_LOW	• THRESHOLD_L
THRESHOLD_NONE	• THRESHOLD_NO • THRESHOLD_N

Note that your number for the HIGH cases is smaller than the MED cases, as shown in the following example:

```
PARM('THRESHOLD_HIGH(70%),THRESHOLD_MED(85%),THRESHOLD_LOW(100%)')
```

Reference:
For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
This check issues the following exception messages:

- ILRH0105E

See the ILRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM).*

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ASM_PLPA_COMMON_USAGE

Description:
Looks at the slot usage of the PLPA and Common page data sets. The check generates an exception if the combined usage of both data sets meets or exceeds the warning threshold.

Reason for check:
You should prevent full conditions on the PLPA and Common page data sets.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMASM,ASM_PLPA_COMMON_USAGE),
      INTERVAL(00:30),
      PARM('THRESHOLD(80%)'),
      DATE('date_of_the_change')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, as follows:

- When not using dynamic severity: THRESHOLD(value),

- When using dynamic severity: THRESHOLD_HIGH(value), THRESHOLD_MED(value), THRESHOLD_LOW(value), THRESHOLD_NONE(value)

  Note that the thresholds specified should increase in value as the severity increases, since a higher value indicates a more severe condition (higher usage of the PLPA and COMMON paging data sets), as shown in the following example:

  ```
  PARM('THRESHOLD_HIGH(90%),THRESHOLD_MED(80%), THRESHOLD_LOW(70%)')
  ```

The variable for the parameters is as follows:

value
: An integer, 0-100 indicating the warning threshold percent. It may optionally be followed by %.

Default: THRESHOLD(80%)

You can use synonyms for these parameters, as follows:

Parameter	Synonyms
THRESHOLD_HIGH	• THRESHOLD_HI • THRESHOLD_H
THRESHOLD_MED	• THRESHOLD_M
THRESHOLD_LOW	• THRESHOLD_L
THRESHOLD_NONE	• THRESHOLD_NO • THRESHOLD_N

```
PARM('THRESHOLD_HIGH(90%),THRESHOLD_MED(80%),THRESHOLD_LOW(70%)')
```

Reference:
For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
This check issues the following exception messages:

- ILRH0111E

See the ILRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ASM_LOCAL_SLOT_USAGE

Description:
Looks at the slot usage of each local page data set. The check generates an exception if the usage on any data set meets or exceeds the warning threshold.

Reason for check:
To maximize the efficiency of ASM slot management, you should keep the slot usage on all local page data sets below 30% .

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMASM,ASM_LOCAL_SLOT_USAGE),
      INTERVAL(00:30),
      SEVERITY(MED),
      PARM('THRESHOLD(30%)'),
      DATE('date_of_the_change')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, as follows:

- When not using dynamic severity: THRESHOLD(*value*),
- When using dynamic severity: THRESHOLD_HIGH(*value*), THRESHOLD_MED(*value*), THRESHOLD_LOW(*value*), THRESHOLD_NONE(*value*)

  Note that the thresholds specified should increase in value as the severity increases, since a higher value indicates a more severe condition (higher usage of the paging data set), as shown in the following example:

  ```
  PARM('THRESHOLD_HIGH(80%),THRESHOLD_MED(60%), THRESHOLD_LOW(30%)')
  ```

The variable for the parameters is as follows:

value
An integer, 0-100 indicating the warning threshold percent. It may optionally be followed by %.

Default: THRESHOLD(30%)

You can use synonyms for these parameters, as follows:

Parameter	Synonyms
THRESHOLD_HIGH	• THRESHOLD_HI • THRESHOLD_H
THRESHOLD_MED	• THRESHOLD_M
THRESHOLD_LOW	• THRESHOLD_L
THRESHOLD_NONE	• THRESHOLD_NO • THRESHOLD_N

```
PARM('THRESHOLD_HIGH(60%),THRESHOLD_MED(25%)')
```

Reference:
For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
This check issues the following exception messages:

- ILRH0107E

See the ILRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Catalog checks (IBMCATALOG)

CATALOG_IMBED_REPLICATE

Description:
Check collects all catalog names in the users environment that are defined with obsolete attributes IMBED and/or REPLICATE. You can then redefine the catalogs without these attributes.

In order to run this check, the user must have RACF read access to **all** the catalogs on the system. If you do not have the correct RACF access, you will receive 100-8 catalog errors.

Once you have identified all the catalogs you need to redefine without IMBED and REPLICATE attributes, IBM suggests that you turn the check off. This is recommended because:

- Users can no longer define catalogs with the obsolete IMBED or REPLICATE attributes. That means that once you have identified any existing catalogs that were defined with IMBED and REPLICATE and redefined them, it is no longer useful to run this check.
- Leaving the check on after you have identified and/or redefined any catalogs defined with IMBED or REPLICATE attributes, the check can cause a performance issue. the check does a sequential search of the active Master Catalog, which means it reads all the records in the master catalog. This I/O to the master catalog to read all the records can impact performance.

You can turn the check off temporarily using F HZSPROC command or permanently using the HZSPRMxx parmlib member.

Reason for check:
No supported releases of z/OS honor either the IMBED or REPLICATE attributes for new catalogs, they are obsoleted by newer, cached DASD devices. Using the IMBED and REPLICATE attributes can waste DASD space and degrade performance, in some cases causing unplanned outages. In addition, servicing catalogs with these attributes is very difficult.

If the check finds instances of IMBED or REPLICATE attributes, the system issues exception message IGGHC104E and generates a report in message IGGHC106I in the message buffer to describe the check's findings. IBM suggests that you use the EXPORT/IMPORT command to remove the attributes:

- Use the EXPORT command to create a back up and later to recover.
- Use the IMPORT command for the exported copies.

Ideally, you should do this during system down time, when the catalogs cannot be accessed by any users.

z/OS releases the check applies to:
z/OS V1R11 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCATALOG,CATALOG_IMBED_REPLICATE)
ACTIVE
SEVERITY(LOW)
INTERVAL(TIMER) HOURS(24)
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
To use the EXPORT and IMPORT commands, see *z/OS DFSMS Access Method Services Commands* for information on the EXPORT and IMPORT commands.

Messages:
This check issues the following exception messages:

- IGGHC104E

See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

CATALOG_RNLS

Description:
Check verifies that your data sets reside on different volumes as other shared catalogs.

CATALOG MANAGEMENT uses the SYSIGGV2 resource while serializing access to catalogs. The SYSIGGV2 resource is used to serialize the entire catalog BCS component across all I/O as well as to serialize access to specific catalog entries. The SYSZVVDS resource is used to serialize access to associated VVDS records. The SYSZVVDS resource along with the SYSIGGV2 resource provide an essential mechanism to facilitate cross system sharing of catalogs. When customer's data sets reside on the same volume as other shared catalogs, deadlocks can occur. This GRS RNL Health Check will help preventing lockouts from shared volumes. The check will perform the system check and indicate when GRS RNLs do not match IBM recommendations for SYSIGGV2, SYSZVVDS, and SYSVTOC.

Reason for check:
All Catalog RESERVES should be converted to SYSTEMS ENQUEUEs unless catalogs are shared outside the sysplex.

z/OS releases the check applies to:
z/OS V2R1 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCATALOG,CATALOG_RNLS)
ACTIVE
```

```
SEVERITY(LOW)
INTERVAL(TIMER) HOURS(336)
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
No

Messages:
This check issues the following exception messages:

- IGGHC110I
- IGGHC111E

See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

Communications Server checks (IBMCS)

CSRES_AUTOQ_GLOBALTCPIPDATA

Description:
Checks to see if:

- The AUTOQUIESCE operand has been specified on the UNRESPONSIVETHRESHOLD resolver setup statement and
- The GLOBALTCPIPDATA resolver setup statement has not been specified in the resolver setup file

Reason for check:
The AUTOQUIESCE operand is ignored if the AUTOQUIESCE operand is specified on the UNRESPONSIVETHRESHOLD resolver setup statement but no GLOBALTCPIPDATA resolver setup statement is specified.

z/OS releases the check applies to:
z/OS V1R13 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSRES_AUTOQ_GLOBALTCPIPDATA)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on AUTOQUIESCE and GLOBALTCPIPDATA resolver setup statement, see the AUTOQUIESCE and GLOBALTCPIPDATA sections in *z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- EZBH015E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSRES_AUTOQ_RESOLVEVIA

Description:
Checks if the RESOLVEVIA statement has been specified with the value TCP in the global TCPIP.DATA file when the autonomic quiescing of unresponsive name servers function is active.

Reason for check:
The default value for the RESOLVEVIA statement is UDP. If the autonomic quiescing of unresponsive name server function is active, the resolver polls unresponsive name servers using UDP, not TCP. If your installation uses TCP, the results of the resolver's polling attempts will not accurately reflect the responsiveness of the name server in your installation.

z/OS releases the check applies to:
z/OS V1R13 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSRES_AUTOQ_RESOLVEVIA)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
EVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on AUTOQUIESCE resolver setup statement and RESOLVEVIA on the TCPIP.DATA file, see the AUTOQUIESCE and RESOLVEVIA sections in *z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- EZBH021E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSRES_AUTOQ_TIMEOUT

Description:
Checks if the configured resolver timeout value in the global TCPIP.DATA file exceeds the optimal setting when the autonomic quiescing of unresponsive name servers function is active.

Reason for check:
The default value for the RESOLVERTIMEOUT statement is 5 seconds. If the autonomic quiescing of unresponsive name server function is active, the resolver polls unresponsive name servers every six seconds. The resolver uses the value you specify or default for the RESOLVERTIMEOUT statement in the global TCPIP.DATA file to determine how long to wait for a response to the poll. If you specify a value for the RESOLVERTIMEOUT statement that is greater than 5 seconds, the resolver will be less efficient when polling unresponsive name servers.

z/OS releases the check applies to:
z/OS V1R13 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSRES_AUTOQ_TIMEOUT)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
PARM('TIMEOUT(5)')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on AUTOQUIESCE resolver setup statement and RESOLVERTIMEOUT on the TCPIP.DATA file, see the AUTOQUIESCE and RESOLVERTIMEOUT sections in*z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- EZBH018E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_CINET_PORTRNG_RSV_*tcpipstackname*

Description:
Checks whether the port range specified by INADDRANYPORT and INADDRANYCOUNT in the BPXPRMxx parmlib member is reserved for OMVS on this stack, when operating in a CINET environment. A port range is reserved on a TCP/IP stack using the PORTRANGE TCP/IP profile statement. The *tcpipstackname* suffix is the job name of the TCP/IP stack to which this check applies. Use CSTCP_CINET_PORTRNG_RSV_* to reference this check for all stacks.

Reason for check:
When operating in a CINET environment, the range of ports defined as available for CINET use by the INADDRANYPORT and INADDRANYCOUNT parameters in the BPXPRMxx parmlib member should be reserved (using the PORTRANGE TCP/IP profile statement) on every TCP/IP stack. This prevents a TCP/IP stack from allocating a port that CINET might subsequently attempt to use, which could result in an ABEND and the following message:

```
BPXF219I A SOCKETS PORT ASSIGNMENT CONFLICT EXISTS BETWEEN UNIX  SYSTEM SERVICES AND TCPIPstackname
```

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSTCP_CINET_PORTRNG_RSV_tcpipstackname)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(MEDIUM)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on using multiple instances of TCP/IP in a CINET environment, see Considerations for multiple instances of TCP/IP in *z/OS Communications Server: IP Configuration Guide*.

Messages:
This check issues the following exception messages:

- EZBH008E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_IPMAXRT4_*tcpipstackname*

Description:
Checks to see whether the total number of IPv4 indirect static and dynamic routes in the TCP/IP stack's routing table has exceeded the maximum threshold (default 2000). The *tcpipstackname* suffix is the job name of the TCP/IP stack to which this check applies. Use CSTCP_IPMAXRT4_* to reference this check for all stacks.

Reason for check:
A high number of routes added by OMPROUTE and the TCP/IP stack can potentially result in high CPU consumption from routing changes. A large routing table is considered to be inefficient in network design and operation. By default, this check is performed once at 30 minutes after stack initialization, at once whenever the total number of routes exceeds the threshold, and then will be repeated once at the specified intervals (default 168 hours for weekly).

z/OS releases the check applies to:
z/OS V1R12 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
ADDREPLACE POLICY STMT(IPMAXRT4)
UPDATE
CHECK(IBMCS,CSTCP_IPMAXRT4_tcpipstackname)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
PARM('IPMAXRT4(2000)')
ACTIVE
SEVERITY(LOW)
INTERVAL(168:00)
```

The following examples show how you can make runtime updates of CSTCP_IPMAXRT4_*tcpipstackname* default values:

```
F hzsproc,update,check=(ibmcs,cstcp_ipmaxrt4_*),parm='ipmaxrt4(1000)'
F hzsproc,update,check=(ibmcs,cstcp_ipmaxrt4_*),interval=02:00
F hzsproc,update,check=(ibmcs,cstcp_ipmaxrt4_*),interval=01:00,parm='ipmaxrt4(1500)'
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes - IPMAXRT4 is an integer value indicating the maximum threshold value for the number of IPv4 indirect static and dynamic routes that the TCP/IP stack can add to its routing table before issuing a warning message to indicate a potential high CPU consumption. Value must be in the range 1 to 65536. Default: IPMAXRT4(2000)

Reference:
See Minimizing the routing responsibility of z/OS Communications Server in *z/OS Communications Server: IP Configuration Guide*.

Messages:
This check issues the following exception messages:

- EZBH013E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_IPMAXRT6_*tcpipstackname*

Description:
Checks to see whether the total number of IPv6 indirect static and dynamic routes in the TCP/IP stack's routing table has exceeded the maximum threshold (default 2000). The *tcpipstackname* suffix is the job name of the TCP/IP stack to which this check applies. Use CSTCP_IPMAXRT6_* to reference this check for all stacks.

Reason for check:
A high number of routes added by OMPROUTE and the TCP/IP stack can potentially result in high CPU consumption from routing changes. A large routing table is considered to be inefficient in network design and operation. By default, this check is performed once at 30 minutes after stack initialization, at once whenever the total number of routes exceeds the threshold, and then will be repeated once at the specified intervals (default 168 hours for weekly).

z/OS releases the check applies to:
z/OS V1R12 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
ADDREPLACE POLICY STMT(IPMAXRT6)
UPDATE
CHECK(IBMCS,CSTCP_IPMAXRT6_tcpipstackname)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
PARM('IPMAXRT6(2000)')
ACTIVE
SEVERITY(LOW)
INTERVAL(168:00)
```

The following examples show how you can make runtime updates of CSTCP_IPMAXRT6_*tcpipstackname* default values:

```
F hzsproc,update,check=(ibmcs,cstcp_ipmaxrt6_*),parm='ipmaxrt6(1000)'
F hzsproc,update,check=(ibmcs,cstcp_ipmaxrt6_*),interval=02:00
F hzsproc,update,check=(ibmcs,cstcp_ipmaxrt6_*),interval=01:00,parm='ipmaxrt6(1500)'
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes - IPMAXRT6 is an integer value indicating the maximum threshold value for the number of IPv6 indirect static and dynamic routes that the TCP/IP

stack can add to its routing table before issuing a warning message to indicate a potential high CPU consumption. Value must be in the range 1 to 65536. Default: IPMAXRT6(2000)

Reference:
See Minimizing the routing responsibility of z/OS Communications Server in *z/OS Communications Server: IP Configuration Guide*.

Messages:
This check issues the following exception messages:

- EZBH013E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_SYSTCPIP_CTRACE_*tcpipstackname*

Description:
Checks if TCP/IP Event Trace (SYSTCPIP) is active with options other than the default options (MINIMUM, INIT, OPCMDS, or OPMSGS). The *tcpipstackname* suffix is the job name of the TCP/IP stack to which this check applies. Use CSTCP_SYSTCPIP_CTRACE_* to reference this check for all stacks.

Reason for check:
If problem documentation is not being gathered, only the default SYSTCPIP trace options (MINIMUM, INIT, OPCMDS, or OPMSGS) should be active. Leaving other options active can result in performance degradation.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
   CHECK(IBMCS,CSTCP_SYSTCPIP_CTRACE_tcpipstackname)
   DATE('date_of_the_change')
   REASON('Your reason for making the update.')
   ACTIVE
   SEVERITY(LOW)
   INTERVAL(24:00)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For information on initializing and modifying TCP/IP Event Trace options, see Specifying trace options in *z/OS Communications Server: IP Diagnosis Guide*.

Messages:
This check issues the following exception messages:

- EZBH002E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_SYSPLEXMON_RECOV_*tcpipstackname*

Description:
Checks to see whether:

- The IPCONFIG DYNAMICXCF or IPCONFIG6 DYNAMICXCF parameters have been specified **and**
- The GLOBALCONFIG SYSPLEXMONITOR RECOVERY parameter has been specified
-

The *tcpipstackname* suffix is the job name of the TCP/IP stack to which this check applies. Use CSTCP_SYSPLEXMON_RECOV_* to reference this check for all stacks.

Reason for check:
IBM suggests that you use GLOBALCONFIG SYSPLEXMONITOR RECOVERY when IPCONFIG DYNAMICXCF or IPCONFIG6 DYNAMICXCF is specified. This allows a TCP/IP stack in a sysplex to perform internal checks to determine if conditions are such that it is unhealthy. If so, it should remove itself from the sysplex, allowing a healthy backup TCP/IP stack to takeover the ownership of the DVIPA interfaces. This enables continued availability to applications.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSTCP_SYSPLEXMON_RECOV_tcpipstackname)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on GLOBALCONFIG SYSPLEXMONITOR RECOVERY, see GLOBALCONFIG in *z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- EZBH006E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_TCPMAXRCVBUFRSIZE_*tcpipstackname*

Description:
Checks if the configured TCP maximum receive buffer size is sufficient to provide optimal support to the z/OS Communications Server FTP Server. The *tcpipstackname* suffix is the job name of the TCP/IP stack to which this check applies. Use CSTCP_TCPMAXRCVBUFRSIZE_* to reference this check for all stacks.

Reason for check:
Optimally, the z/OS Communications Server FTP Server needs a buffer size of 180K for data connections. TCPMAXRCVBUFRSIZE should not be set below 180K if the z/OS Communications Server FTP Server is being used.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
   CHECK(IBMCS,CSTCP_TCPMAXRCVBUFRSIZE_tcpipstackname)
   DATE('date_of_the_change')
   REASON('Your reason for making the update.')
   PARM('MAXRCVBUFRSIZE(180K)')
   ACTIVE
   SEVERITY(LOW)
   INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes - MAXRCVBUFRSIZE is an integer value with optional suffix (K) indicating the maximum value an application can set as its receive buffer size (in bytes) using SETSOCKOPT(). Value must be in the range 256 to 512K. Default: MAXRCVBUFRSIZE(180K)

Reference:
For more information on TCPMAXRCVBUFRSIZE, see TCPCONFIG in *z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- EZBH004E

See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_CSM_STG_LIMIT

Description:
Checks if the maximum bytes of fixed storage dedicated to CSM use and the maximum amount of storage dedicated to ECSA CSM buffers is adequate to meet the needs of your system.

Reason for check:
The default values for IVTPRM00 are 100 MEG for both FIXED, and ECSA. It is suggested that they initially be coded at 100M MAX ECSA and 100M MAX FIXED. Then the system should be monitored for one week using the DISPLAY CSM command to determine peak usage. IVTPRM00 MAX ECSA and MAX FIXED values should then be adjusted to 1.5 times the highest value indicated in the DISPLAY CSM outputs.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMCS,CSVTAM_CSM_STG_LIMIT)
   DATE('date_of_the_change')
   REASON('Your reason for making the update.')
   ACTIVE
   SEVERITY(LOW)
   INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes:

- MAXFIX - Integer value with optional suffix (K,M) indicating the maximum bytes of fixed storage dedicated to CSM use. Default: MAXFIX(100M)
- MAXECSA - Integer value with optional suffix (K,M) indicating the maximum amount of storage dedicated to ECSA CSM buffers. Default: MAXECSA(100M)

Values for both parameters must be in the range between 1024K to 2048M.

Reference:
For more information on defining the maximum bytes of fixed storage dedicated to CSM use and the maximum amount of storage dedicated to ECSA CSM buffers, see IVTPRM00 (Communication Storage Manager) in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- ISTH017E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_T1BUF_T2BUF_EE

Description:
Checks to see whether the T1BUF and T2BUF buffer pool allocations for the system are adequate when Enterprise Extender (EE) are being used .

Reason for check:
The T1BUF and T2BUF buffer pools are used exclusively for Enterprise Extender (EE) functions that use QDIO or HyperSockets. When EE is being used with QDIO or HyperSockets DLCs, setting the T1BUF or T2BUF buffer pool allocations at their default values (16 for T1BUF and 8 for T2BUF) might not be optimal.

The T1BUF and T2BUF buffer pools should be monitored and tuned to minimize the number of expansions. Minimizing buffer pool expansions will decrease internal buffer overhead processing which should increase throughput while reducing CPU consumption. These buffer pools can be monitored using the D NET,BFRUSE,BUF=(T1,T2) command. Once the appropriate allocation values for the T1BUF and T2BUF buffer pools has been determined, you can change the T1BUF and T2BUF Start option allocation values.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSVTAM_T1BUF_T2BUF_EE)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on defining T1BUF and T2BUF parameters, see the Buffer Pool section of *z/OS Communications Server: SNA Resource Definition Reference*.

Messages:
This check issues the following exception messages:

- ISTH013E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_T1BUF_T2BUF_NOEE

Description:
Checks the T1BUF and T2BUF buffer pool allocations for the system when Enterprise Extender (EE) is not being used.

Reason for check:
The T1BUF and T2BUF buffer pool is used exclusively for Enterprise Extender (EE) functions that use QDIO or HyperSockets. If EE is not being used, the T1BUF or T2BUF buffer pool allocations are not optimal if set above their default values (16 for T1BUF and 8 for T2BUF).

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSVTAM_T1BUF_T2BUF_NOEE)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on defining T1BUF and T2BUF parameters, see the Buffer Pool section of *z/OS Communications Server: SNA Resource Definition Reference*.

Messages:
This check issues the following exception messages:

- ISTH016E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_VIT_DSPSIZE

Description:
Checks to see whether the VTAM® Internal Trace (VIT) dataspace table size is set to 5 (50 MB).

Reason for check:
IBM suggests a VIT dataspace table size of 5 (50 MB) to allow an optimal amount of trace information to be captured for serviceability.

z/OS releases the check applies to:
z/OS V1R9 through z/OS V1R12.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSVTAM_VIT_DSPSIZE)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on defining VTAM Internal Trace parameters, see TRACE for MODULE, STATE (with OPTION) or VTAM internal trace in *z/OS Communications Server: SNA Resource Definition Reference*.

Messages:
This check issues the following exception messages:

- ISTH008E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_VIT_OPT_ALL

Description:
'Check to see whether all VTAM Internal Trace (VIT) options are active. Having all VIT options active might not optimal for system performance.

Reason for check:
It might not be optimal for all VIT options to be active, unless this was requested by IBM service. In general, only a subset of all the VIT options needs to be made active to service a problem.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSVTAM_VIT_OPT_ALL)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on defining VTAM Internal Trace parameters, see TRACE for MODULE, STATE (with OPTION) or VTAM internal trace in *z/OS Communications Server: SNA Resource Definition Reference*.

Messages:
This check issues the following exception messages:

- ISTH010E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_VIT_OPT_PSSSMS

Description:
Checks to see whether the VTAM Internal Trace (VIT) options PSS and SMS are active.

Reason for check:
IBM suggests that the VIT PSS and SMS options always be activated, since they are almost always required when servicing a VTAM problem.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSVTAM_VIT_PSS_SMS)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on defining VTAM Internal Trace parameters, see TRACE for MODULE, STATE (with OPTION) or VTAM internal trace in *z/OS Communications Server: SNA Resource Definition Reference*.

Messages:
This check issues the following exception messages:

- ISTH006E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSVTAM_VIT_SIZE

Description:
Checks to see whether the VTAM Internal Trace (VIT) table size is set to the maximum value (999).

Reason for check:
A maximum table size of 999 for the VIT table allows the maximum amount of trace information to be captured for serviceability.

z/OS releases the check applies to:
z/OS V1R9 through z/OS V1R12.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,CSVTAM_VIT_SIZE)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on defining VTAM Internal Trace parameters, see TRACE for MODULE, STATE (with OPTION) or VTAM internal trace in *z/OS Communications Server: SNA Resource Definition Reference*.

Messages:
This check issues the following exception messages:

- ISTH004E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV2R1_CS_GATEWAY

Description:
Checks whether the GATEWAY statement is in use in a TCP/IP profile on this system. Support for the GATEWAY statement will be removed in a future release of IBM z/OS Communications Server.

If this check determines that a GATEWAY statement has been processed, it will continue to report this exception for the duration of this IPL, or as long as this migration check is active. When this exception condition is detected, message ISTM014E will be issued and will be followed by message ISTM900I which will indicate the date and time that the GATEWAY statement had been last processed, even if it has since been removed from all TCP/IP configuration files. Therefore, if this exception condition has been corrected (for example, GATEWAY statements have been removed from all TCP/IP profiles on this system) you can use message ISTM900I to determine whether a new use of GATEWAY statement has been detected, or whether the exception condition is related to the earlier detection of the GATEWAY configuration statement.

When the check is activated, at that point a TCP/IP stack with the GATEWAY statement may have not been processed yet. As a result, you may not see an exception for this check the first time the check is processed. However subsequent processing of the check (manually triggered or during subsequent intervals) will detect whether a TCP/IP configuration that uses the GATEWAY statement has been processed.

Reason for check:
Since the GATEWAY configuration statement will no longer be supported in the TCP/IP profile in a future release of z/OS Communications Server , IBM suggests that customers who currently use the GATEWAY statement migrate to the BEGINROUTES/ENDROUTES configuration block.

z/OS releases the check applies to:
z/OS V2R1

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMCS,ZOSMIGV2R1_CS_GATEWAY)
DATE('date_of_the_change')
REASON('Your reason for making the update.'))
INACTIVE
SEVERITY(LOW)
INTERVAL(24:00)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See the information on understanding GATEWAY statements in *z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- ISTM014E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV2R1_CS_LEGACYDEVICE

Description:
Checks whether any TCP/IP profile statements for legacy device types have been configured on this system. Support for the DEVICE and LINK profile statements for the following TCP/IP legacy device types will be eliminated in a future release of IBM z/OS Communications Server:

- ATM (Asynchronous Transfer Mode)
- CDLC (Channel Data Link Control)
- CLAW (Common Link Access to Workstation)
- HYPERchannel
- SNALINK (LU0 and LU6.2)
- X.25

Because support will be eliminated for the ATM device type, the following associated TCP/IP profile statements will no longer be supported:

- ATMARPSV
- ATMLIS
- ATMPVC

If this check determines that a TCP/IP profile statement has been processed for a legacy device type, it continues to report this exception for the duration of this IPL, or as long as this migration health check is active. When this exception is detected, message ISTM016E and then message ISTM900I are issued. Message ISTM900I indicates the date and time that the statement was last processed, even if this exception has been corrected. For example, the statement has been removed from all TCP/IP configuration files on the system. Therefore, if this exception has been corrected, you can use message ISTM900I to determine whether a new use of a legacy device type profile statement has been detected, or whether the exception is related to the earlier detection of a legacy device type profile statement.

When the check is activated, a TCP/IP stack with a legacy device type profile statement might not have been processed. Therefore, you might not see an exception the first time the check is processed. However, subsequent processing of the check, either manually triggered or during subsequent intervals, will detect whether a TCP/IP configuration that uses a legacy device type profile statement has been processed.

Reason for check:
Because the profile statements for legacy device types will not be supported in the TCP/IP profile in a future release of z/OS Communications Server, it is

recommended that users who currently use these statements migrate to a later interface type, such as OSA-Express QDIO or HiperSockets.

z/OS releases the check applies to:
z/OS V1R13 with the PTFs for APARs OA44669 and PI12977 applied, or z/OS V2R1 with the PTFs for APARs OA44671 and PI12981 applied.

User override of IBM values:
The following example shows the default keywords for the check. You can override the check defaults on either a POLICY statement in the HZSPRM*xx* parmlib member or on a MODIFY command.

```
CHECK(IBMCS,ZOSMIGV2R1_CS_LEGACYDEVICE)
DATE('date_of_the_change')
REASON('Your reason for making the update.'))
INACTIVE
SEVERITY(LOW)
INTERVAL(24:00)
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on using OSA-Express QDIO or HiperSockets interfaces, see Considerations for networking hardware attachment in *z/OS Communications Server: IP Configuration Reference*.

Messages:
This check issues the following exception messages:

- ISTM016E

See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Consoles checks (IBMCNZ)

CNZ_AMRF_Eventual_Action_Msgs

Description:
Checks that eventual action messages are not retained if the Action Message Retention Facility (AMRF) is active.

Reason for check:
Exclude eventual action messages from being retained when AMRF is active. Because AMRF causes messages to remain in storage, eventual action messages may exhaust storage needed to retain critical and immediate action messages.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
       DATE('date_of_the_change')
       REASON('Your reason for making the update.')
       ACTIVE
       SEVERITY(LOW)
       INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on AMRF, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0004I

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Console_MasterAuth_Cmdsys

Description:
Checks that there is an active console with MASTER authority that has command association to this system.

Reason for check:
Assign MASTER authority and proper command association to an MCS, EMCS or SMCS console. This console gives you the ability to control your system.

z/OS releases the check applies to:
z/OS V1R4 through V1R7

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
 UPDATE,
       DATE('date_of_the_change')
       REASON('Your reason for making the update.')
       ACTIVE
       SEVERITY(LOW)
       INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on MASTER authority and command association, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0002I

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Console_Mscope_And_Routcode

Description:
Checks that each MCS/SMCS/EMCS console is not defined with multi-system message scopes AND receiving all routing codes (or all except routing code 11).

Reason for check:
All MCS, SMCS, or EMCS consoles defined with multi-system message scope should only receive routing codes specific to that console's function. Conversely, all MCS, SMCS, EMCS consoles that are receiving all routing codes (or all except routing code 11) should be defined with single-system message scope. This reduces the number of messages sent to a console in the sysplex.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    DATE('date_of_the_change')
    REASON('Your reason for making the update.')
    ACTIVE
    SEVERITY(LOW)
    INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on message scope and routing codes, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0003I

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Console_Operating_Mode

Description:
This check identifies installations running in 'Shared' console service operating mode.

Reason for check:
'Distributed' mode is the preferred mode of operations and 'Shared' mode will be removed in a future release.

z/OS releases the check applies to:
z/OS 1.13

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
   DATE('date_of_the_change')
   REASON('Your reason for making the update.')
   INACTIVE
   SEVERITY(MEDIUM)
   INTERVAL(ONETIME)
```

Parameters accepted:
No

Reference:
For additional information about console service operating mode see *z/OS MVS Planning: Operations*.

For additional information on setting the console service operating mode see the CONSOLxx and IEASYSxx parmlib members in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- CNZHF0014E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Console_Routcode_11

Description:
Ensures that no MCS or SMCS console is receiving ROUTCODE 11 messages.

Reason for check:
All MCS/SMCS consoles should not be receiving messages issued with routing code 11. Messages issued with routing code 11 are intended to be sent to the programmer, not the operator console.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    DATE('date_of_the_change')
    REASON('Your reason for making the update.')
    ACTIVE
    SEVERITY(LOW)
    INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on routing codes, see*z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0005I

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_EMCS_Hardcopy_Mscope

Description:
Checks to see that each EMCS console defined with a multi-system message scope is not receiving the hardcopy message set.

Reason for check:
All EMCS consoles with multi-system message scopes should not receive the hardcopy message set. This can affect message processing times and console availability.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      DATE('date_of_the_change')
      REASON('Your reason for making the update.')
      ACTIVE
      SEVERITY(MED)
      INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on EMCS consoles, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0006E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_EMCS_Inactive_Consoles

Description:
Ensures that there are not an excessive number of inactive EMCS consoles.

Reason for check:
If the EMCS console is no longer needed, use the EMCS console removal service (IEARELEC) to remove the EMCS console definition. The number of inactive EMCS consoles in use in a sysplex can affect the time it takes for a system to join a sysplex.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      DATE('date_of_the_change')
      REASON('Your reason for making the update.')
      PARM(10000)
      ACTIVE
      SEVERITY(HI)
      INTERVAL(24:00)
```

Parameters accepted:
Yes. You can specify the number of inactive EMCS consoles that you deem excessive. PARM(10000) is the default.

Reference:
For more information on EMCS consoles, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0009E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_OBSOLETE_MSGFLD_AUTOMATION

Description:
Checks to see whether an obsolete version of Message Flood Automation is installed. z/OS R11 eliminates the use of message exit IEAVMXIT and the command exit CNZZCMXT. If either message exit IEAVMXIT or command exit CNZZCMXT are installed on a z/OS V1 R11 system, the check generates an exception.

Reason for check:
z/OS R11 eliminates the use of message exit IEAVMXIT and the command exit CNZZCMXT and integrates MFA into mainline message processing. Customers should remove the old code from their exits in order to ensure that old processing does not interfere with new processing.

z/OS releases the check applies to:
z/OS V1 R11 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMCNZ,CNZ_OBSOLETE_MSGFLD_AUTOMATION)
  SEVERITY(MED)INTERVAL(TIMER) HOURS(24) MINUTES(0)
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see:

- *z/OS MVS System Commands*
- *z/OS MVS Planning: Operations*

Messages:
This check issues the following exception messages:

- CNZHF0011E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The following shows output from the check :

```
CNZHR0011I The following components of an obsolete Message Flood
Automation installation were detected:

Message Exit (IEAVMXIT)
Command Exit (CNZZCMXT)

* Medium Severity Exception *

CNZHF0011E An obsolete version of Message Flood Automation is active

  Explanation:  One or more components of an obsolete version of Message
    Flood Automation were determined to be active. Report message
    CNZHR0011I identifies which components of the obsolete version of
    Message Flood Automation were detected.

    Obsolete versions of Message Flood Automation conflict with current
    Message Flood Automation processing.

  System Action:  The system continues processing.

  Operator Response:  N/A

  System Programmer Response:  Remove obsolete versions of Message Flood
    Automation from your installation's IEAVMXIT exit and MPFLSTxx .CMD
    statements.

  Problem Determination:  See CNZHR0011I in the message buffer that
    identifies which components of an obsolete version of Message Flood
    Automation were detected.

  Source:  Consoles (SC1CK)

  Reference Documentation:
      z/OS MVS Planning: Operations
      z/OS MVS System Commands
      z/OS Migration

  Automation:  N/A

  Check Reason:  Obsolete versions of Message Flood Automation must be
    removed from the system
```

CNZ_Syscons_Allowcmd

Description:
The check will compare the ALLOWCMD setting in the CONSOLxx parmlib member for the system console with the desired setting.

Reason for check:
To ensure that the system console is as available as possible to act as a console of last resort.

z/OS releases the check applies to:
z/OS V1R12 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      DATE('date_of_the_change')
      REASON('YOUR REASON FOR MAKING THE UPDATE.')
      ACTIVE
      SEVERITY(LOW)
      INTERVAL(24:00)
```

Parameters accepted:
Yes:

PARM('Y/N')
Indicates which setting is preferred for ALLOWCMD on the system console.

- PARM('Y') indicates that ALLOWCMD should be specified for the system console.
- PARM('N') indicates that commands should only be allowed from the system console if it's in PD-mode.

Reference:

- For more information on systems consoles, see *z/OS MVS Planning: Operations*
- For more information on ALLOWCMD in the CONSOLxx parmlib member, see*z/OS MVS Initialization and Tuning Reference*

Messages:
This check issues the following exception messages:

- CNZHF0015E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Syscons_Mscope

Description:
Ensures that the system console has a single-system message scope.

Reason for check:
The system console should only receive messages from the local system to avoid having to process large numbers of messages.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    DATE('date_of_the_change')
    REASON('Your reason for making the update.')
    ACTIVE
    SEVERITY(MED)
    INTERVAL(24:00)
```

Parameters accepted:
No.

Reference:
For more information on systems consoles, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0007E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Syscons_PD_Mode

Description:
Ensures that the system console is not in Problem Determination (PD) mode.

Reason for check:
The system console should not be running in PD mode during normal operations. The system console should only be in PD mode to perform necessary recovery operations in emergency situations.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
  UPDATE,
        DATE('date_of_the_change')
        REASON('Your reason for making the update.')
        ACTIVE
        SEVERITY(MED)
        INTERVAL(01:00)
```

Parameters accepted:
No

Reference:
For more information on system consoles, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0010E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Syscons_Routcode

Description:
Ensures that the system console is receiving the minimum set of routing codes (1, 2 and 10).

Reason for check:
The system console should be configured to receive, at a minimum, routing codes 1, 2, and 10. This is to ensure that the system console receives all important messages.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      DATE('date_of_the_change')
      REASON('Your reason for making the update.')
      ACTIVE
      SEVERITY(LOW)
      INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on systems consoles, see *z/OS MVS Planning: Operations*.

Messages:
This check issues the following exception messages:

- CNZHF0008I

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Task_Table

Description:
Reports the status of important tasks that run in the CONSOLE address space.

Reason for check:
Using the report generated from this check, installations can determine if there are (real or potential) problems with specific functions of the Consoles component.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      DATE('date_of_the_change')
      REASON('Your reason for making the update.')
      ACTIVE
      SEVERITY(LOW)
      INTERVAL(00:15)
```

Parameters accepted:
No

Reference:
N / A

Messages:
This check issues the following exception messages:

- None

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R13_CNZ_Cons_Oper_Mode

Description:
The check identifies installations that haven't explicitly identified their console service operating mode.

Reason for check:
For systems at a level prior to z/OS V1R13, the default console service operating mode is 'Shared', but beginning in z/OS 1.13 the new default is 'Distributed'. Simply accepting the default may leave the installation unaware that they are going to run in 'Distributed' mode.

z/OS releases the check applies to:
z/OS 1.11 and z/OS 1.12 only.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      DATE('date_of_the_change')
      REASON('Your reason for making the update.')
      ACTIVE
      SEVERITY(LOW)
      INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For additional information about console service operating mode see *z/OS MVS Planning: Operations*.

For additional information on setting the console service operating mode see the CONSOLxx and IEASYSxx parmlib members in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- CNZHF0013E

See the CNZHF messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Contents supervision checks (IBMCSV)

CSV_APF_EXISTS

Description:
Checks to see if data sets described by entries in the APF list are consistent with data sets that exist on the system.

Reason for check:
A potential system integrity risk exists when a data set cannot be allocated using the criteria specified in the system APF list.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
Yes

- MIGRATEDOK(NO) specifies that a migrated APF authorized data set is always an exception.
- MIGRATEDOK(YES) specifies that a migrated APF authorized data set is never an exception.
- MIGRATEDOK(SYSTEM) specifies that a migrated APF authorized data set is an exception unless the data set is SMS-managed (that is its catalog entry contains a storage class) and has an "SMS" APF entry. This is the default."

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMCSV,CSV_APF_EXISTS)
  PARM('MIGRATEDOK(SYSTEM)')
  SEVERITY(LOW) INTERVAL(04:00) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides additional error detail in debug mode. You can put a check into debug mode using any of the following:

- UPDATE,*filters*,DEBUG=ON parameters on either the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member

- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF.

Reference:
For more information, see:

- *z/OS MVS Programming: Authorized Assembler Services Guide*
- *z/OS MVS Initialization and Tuning Guide*

Messages:
This check issues the following exception messages:

- CSVH0957E

See the CSVH messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for MLS users:
SYSLOW

Output:
The report that CSV_APF_EXISTS produces is shown below:

```
VOLUME is the volume specified in the APF entry or *SMS*
DSNAME is the data set name specified in the APF entry
ERROR is the problem that was detected by the check


CSVH0955I A problem was found with each APF list entry displayed

VOLUME DSNAME                                   ERROR

TMPSTG ANY.ALIAS                                DS not found
*SMS*  ANY.DATASET                              DS not SMS-managed
BADVOL ANY.DATASET                              Volume not found
*SMS*  ANY.SMS.ALIAS                            DS is alias
ALL001 ANY.SMS.DATASET                          DS is SMS-managed
```

In the output:

VOLUME
The volume specified in the APF entry or *SMS*

DSNAME
The data set name specified in the APF entry

ERROR
The problem that was detected by the check

CSV_LNKLST_NEWEXTENTS

Description:
Checks to see if the number of extents in each data set of a LNKLST set has changed since the LNKLST was activated. All active LNKLST sets are checked.

Reason for check:
The system will recognize only the extents that existed when the LNKLST was made ACTIVE.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
The following parameters are supported to control WTOs produced by exception messages when a new extent is detected in the LNKLST set:

PARM(ALL)
Exceptions should be issued for all active LNKLST data sets for which new extents were created after the LNKLST was activated.

PARM('NEW(text value)')
Exceptions should only be issued for errors that are detected after this parameter is set.

The following are examples of PARMS specifications for CSV_LNKLST_NEWEXTENTS:

```
PARMS('NEW(yyyy/mm/dd hh:mm)')
PARMS('ALL')
```

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMCSV,CSV_LNKLST_NEWEXTENTS)
  PARM('ALL')
  SEVERITY(HIGH) INTERVAL(01:00)   DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
In debug mode, this check includes additional error information in the message buffer. You can put a check into debug mode using any of the following:

- UPDATE,*filters*,DEBUG=ON parameters on either the MODIFY command or in an HZSPRMxx parmlib member
- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF.

Verbose support:
Yes. When VERBOSE=YES is specified, the extent information for all data sets (not just those with new extents after activation) will be shown.

Reference:
For more information, see:

- *z/OS MVS Initialization and Tuning Guide*
- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS System Commands*

Messages:
This check issues the following exception messages:

- CSVH0970E

See the CSVH messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for MLS users:
SYSLOW

Output:
The report that CSV_LNKLST_NEWEXTENTS produces is shown below:

```
CSVH0977I  LNKLST set NEWLST

The error status is in column one:
C = Confirmed error    * = New error
```

```
 ORIG CURR VOLUME DSNAME
*   2    4 SIXPAK XESCT.SHARONP.LOADLIB
TOTAL EXTENTS ORIG: 130    CURR: 132
```

In the output:

ORIG
The number of extents that existed when the LNKLST was made ACTIVE.

CURR
The number of extents that existed the last time the check routine executed.

CSV_LNKLST_SPACE

Description:
Checks all active LNKLST sets on the system for PDS's that were created with secondary space defined.

Reason for check:
IBM suggests that partitioned data sets (PDS) in the LNKLST be defined with only primary spaces. Allocating a PDS with only primary space causes it to have one extent. That makes it easier to stay within the 255-extent limit of the LNKLST set and prevents new extents from being created if a data set is updated after the LNKLST is activated.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMCSV,CSV_LNKLST_SPACE)
  SEVERITY(LOW) INTERVAL(24:00)   DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
In debug mode, this check includes additional error information in the message buffer. You can put a check into debug mode using any of the following:

- UPDATE,*filters*,DEBUG=ON parameters on either the MODIFY command or in an HZSPRMxx parmlib member
- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF.

Reference:
For more information, see:

- *z/OS MVS Initialization and Tuning Guide*
- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS System Commands*

Messages:
This check issues the following exception messages:

- CSVH0980E

See the CSVH messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for MLS users:
SYSLOW

Output:
The report that CSV_LNKLST_SPACE produces is shown below:

```
CSVH0981I LNKLST set LNKLST00 data sets allocated with secondary

VOLUME DSNAME

ZOS17B SYS1.LINKLIB
ZOS17B SYS1.MIGLIB
ZOS17B SYS1.CSSLIB
ZOS17B SYS1.CMDLIB
```

In the output:

VOLUME
The volume serial number of a data set in the LNKLST

DSNAME
The name of a data set in the LNKLST

CSV_LPA_CHANGES

Description:
This check compares the current IPL's LPA to the previous IPL's LPA, providing information about modules that have changed in size (or been added or removed), along with summaries of the storage deltas for each of the LPA sub-areas (PLPA, MLPA, FLPA, device support, dynamic LPA), and totals for each of the sub-areas. In both cases, the display will differentiate between the below-16M area and the above-16M area.

Reason for check:
An increase in the amount of LPA could mean that the private regions size might soon be, or has been, reduced which could cause application failures. Running the system in exception has no consequence. The exception is intended to alert to the possibilities.

z/OS releases the check applies to:
z/OS V1R9 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMCSV,CSV_LPA_CHANGES),
      INTERVAL(ONETIME),
      SEVERITY(LOW),
      PARM(
           'PLPAD(32K),EPLPAD(1M)',
           'MLPAD(32K),EMLPAD(1M)',
           'FLPAD(32K),EFLPAD(1M)',
           'DEVSUPD(32K),EDEVSUPD(1M)',
           'DLPAD(32K),EDLPAD(1M)',
```

```
        'LPAD(64K),ELPAD(1M)'
      ),
DATE('20060424')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:

- PLPAD(n), for PLPA Delta, specifies an integer 0-2G. If the delta for PLPA exceeds n, an exception message is issued. The default is 32K.
- MLPAD(n), for MLPA Delta, specifies an integer 0-2G. If the delta for MLPA exceeds n, an exception message is issued. The default is 32K.
- FLPAD(n), for FLPA Delta, specifies an integer 0-2G. If the delta for FLPA exceeds n, an exception message is issued. The default is 32K.
- DEVSUPD(n), for Device Support LPA Delta, specifies an integer 0-2G. If the delta for Device Support LPA exceeds n, an exception message is issued. The default is 32K.
- DLPAD(n), for Dynamic LPA Delta, specifies an integer 0-2G. If the delta for dynamic LPA exceeds n, an exception message is issued. The default is 32K.
- LPAD(n), for LPA Delta, specifies an integer 0-2G. If the delta for LPA (the sum of the PLPA, MLPA, FLPA, DEVSUP LPA, and DLPA deltas) exceeds n, an exception message is issued. The default is 64K.
- EPLPAD(n), for Extended PLPA Delta, specifies an integer 0-2G. If the delta for extended PLPA exceeds n, an exception message is issued. The default is 1M.
- EMLPAD(n), for Extended MLPA Delta, specifies an integer 0-2G. If the delta for extended MLPA exceeds n, an exception message is issued. The default is 1M.
- EFLPAD(n), for Extended FLPA Delta, specifies an integer 0-2G. If the delta for extended FLPA exceeds n, an exception message is issued. The default is 1M.
- EDEVSUPD(n), for Device Support Extended LPA Delta, specifies an integer 0-2G. If the delta for Device Support extended LPA exceeds n, an exception message is issued. The default is 1MK.
- EDLPAD(n), for Extended Dynamic LPA Delta, specifies an integer 0-2G. If the delta for extended dynamic LPA exceeds n, an exception message is issued. The default is 1M.
- ELPAD(n), for Extended LPA Delta, specifies an integer 0-2G. If the delta for Extended LPA (the sum of the EPLPA, EMLPA, EFLPA, DEVSUP ELPA, and EDLPA deltas) exceeds n, an exception message is issued. The default is 1M.

Reference:

- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS Initialization and Tuning Guide*

Messages:
This check issues the following exception messages:

- CSVH1001E

See the CSVH messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

DAE checks (IBMDAE)

DAE_SHAREDSN

Description:
Checks that in a sysplex environment DAE data set is shared between systems (DAE option SHARE(DSN) is in effect).

This check is only applicable for a system in a Parallel Sysplex environment. Note, however, that this check is **not** applicable and will not run on a geographically dispersed Parallel Sysplex (GDPS®) controlling system (K-sys) in a GDPS environment.

Reason for check:
In a Parallel Sysplex environment, if the DAE data set is not being shared, it is possible that the system requests a dump for a problem occurred on another system in the sysplex. To prevent duplicate dumps on all systems in the Sysplex, IBM recommends that you share the same ADYSETxx parameter values in each system.

This check will detect if the DAE data set is being shared in a Parallel Sysplex environment.

z/OS releases the check applies to:
z/OS V1R11 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMDAE,DAE_SHAREDSN)
ACTIVE
SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:

Reference:
For more information, see:

- *z/OS MVS Diagnosis: Tools and Service Aids*.

Messages:
This check issues the following exception messages:

- ADYH011E

See the ADYH messages in *z/OS MVS System Messages, Vol 1 (ABA-AOM)*.

SECLABEL recommended for MLS users:
SYSLOW

DAE_SUPPRESSING

Description:
Checks that DAE is active and has the configuration recommended by IBM.

Reason for check:
The system requests dumps that you may not need. Dump Analysis and Elimination (DAE) is designed to suppress duplicate dumps and to help prevent the system from using unneeded resources by capturing diagnostic data repeatedly for the same problems. This check will detect whether DAE is active and your settings match those recommended by IBM.

z/OS releases the check applies to:
z/OS V1R11 and later.

Parameters accepted:
The following parameters are supported to identify the expected suppression options.

SVCDUMP(SUPPRESSALL | SUPPRESS | NONE)
Specifies that the check compare the current SVCDUMP suppression option (specified in the SVCDUMP statement of the ADYSETxx parmlib member) to the specified value and write exception message ADYH004E upon mismatch.

Default: SUPPRESSALL

SYSMDUMP(SUPPRESSALL | SUPPRESS | NONE | ANY)
Any choice other than ANY indicates that the check should compare the current SYSMDUMP suppression option (specified in the SYSMDUMP statement of the ADYSETxx parmlib member) to the specified value and write exception message ADYH005E upon mismatch.

ANY indicates that any option is acceptable so that there is never a mismatch.

Default: ANY

The following is an example of PARMS specifications for DAE_SUPPRESSING:

```
PARM('SVCDUMP(SUPPRESSALL),SYSMDUMP(ANY)')
```

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMDAE,DAE_SUPPRESSING)
ACTIVE
PARM('SVCDUMP(SUPPRESSALL),SYSMDUMP(ANY)')
SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:

Reference:
For more information, see:

- *z/OS MVS Diagnosis: Tools and Service Aids*.

Messages:
This check issues the following exception messages:

- ADYH0001E

See the ADYH messages in *z/OS MVS System Messages, Vol 1 (ABA-AOM)*.

SECLABEL recommended for MLS users:
SYSLOW

Device Manager checks (IBMDMO)

DMO_TAPE_LIBRARY_INIT_ERRORS

Description:
This check reports any tape library initialization errors that were detected during IPL. This is a local check, which will run once per the life of the IPL.

Reason for check:
Ensures that tape library HCD definitions agree with the tape library hardware definitions.

z/OS releases the check applies to:
z/OS V1R13.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK( DMO_TAPE_LIBRARY_INIT_ERRORS)
INTERVAL(ONETIME),
SEVERITY(LOW)
DATE('date_of_the_change')
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For additional information, refer to messages IEA437I and IEA438I in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

Messages:
This check issues the following exception messages:

- DMOH0104E

See the DMOH messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

DFSMS OPEN/CLOSE/EOV checks (IBMOCE)

OCE_XTIOT_CHECK

Description:
This check looks to see whether XTIOTs are enabled for non-VSAM data sets. You can enable XTIOTs for non-VSAM data by specifying NON_VSAM_XTIOT=YES sets in the DEVSUPxx parmlib member. NON_VSAM_XTIOT=YES has to be specified in every DEVSUPxx member that is identified by the current DEVSUPxx suffix list (as specified by DEVSUP=(xx,...,zz) in IEASYSxx). Otherwise the NON_VSAM_XTIOT value might get reset to its default of NO by one of the DEVSUPxx members which do not list NON_VSAM_XTIOT=YES explicitly.

Reason for check:
Enabling XTIOTs for non-VSAM data sets decreases the chances of running out of virtual storage when allocating and concurrently opening many sequential and partitioned data sets.

z/OS releases the check applies to:
z/OS V2R1 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMOCE, OCE_XTIOT_CHECK)
INTERVAL(ONETIME),
SEVERITY(LOW)
DATE('date_of_the_change')
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For additional information, see the DEVSUPxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IEC0H0101E

See the IEC0H messages in *z/OS MVS System Messages, Vol 7 (IEB-IEE)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:
The following shows sample output from the check exception:

```
CHECK(IBMOCE,OCE_XTIOT_CHECK)
START TIME: 08/29/2011 12:52:06.852243
CHECK DATE: 20110410  CHECK SEVERITY: LOW

* Low Severity Exception *

IEC0H0101E XTIOTs for non vsam data sets is not active
IBM suggests setting NON_VSAM_XTIOT=YES
in the DEVSUPxx member of SYS1.PARMLIB.

  Explanation:  IBM suggests setting NON_VSAM_XTIOT=YES in the DEVSUPxx
    member of SYS1.PARMLIB to decrease the chances of running out of
    virtural storage when allocating and concurrently opening many
    sequential and partitioned data sets.

  System Action:  The system continues processing.

  Operator Response:  N/A

  System Programmer Response:  Encourage the use of XTIOT allocations.

  Problem Determination:  N/A

  Source:  DFSMS OPEN/CLOSE/EOV

  Reference Documentation:  For additional information see:

    z/OS V1R12.0 MVS Initialization and Tuning Reference


  Automation:  N/A

  Check Reason:  Check whether XTIOTs for non VSAM is enabled.

END TIME: 08/29/2011 12:52:06.869138  STATUS: EXCEPTION-LOW
```

Global Resource Serialization checks (IBMGRS)

GRS_AUTHQLVL_SETTING

Description:

This check reports on whether the system is running with the recommended AUTHQLVL setting.

Reason for check:

If the AUTHQLVL parameter is not set to the maximum level, certain requests may be susceptible to denial of service attacks.

z/OS releases the check applies to:

z/OS V1R13 and later.

User override of IBM values:

The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
            CHECK(IBMGRS, GRS_AUTHQLVL_SETTING)
            ACTIVE
            SEVERITY(LOW)
            INTERVAL(ONETIME)
            DATE('date_of_the_change')
            PARM('2')
            REASON('Your reason for making the update.');
```

Parameters accepted:
Yes, you can specify the AUTHQLVL value you want the check to look for, either 1 or 2. For example, `PARM('1')`

Default: 2

Reference:
For more information on GRS, see *z/OS MVS Planning: Global Resource Serialization*.

Messages:
This check issues the following exception messages:

- ISGH0322E
- ISGH0323E

And the following information messages:

-
- ISGH0102I
- ISGH0321I

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

GRS_Mode

Description:
Checks the mode of the Global Resource Serialization complex.

Reason for check:
A STAR configuration is recommended because it provides better availability, real storage consumption, processing capacity, and response time.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMGRS,GRS_MODE)
       SEVERITY(LOW) INTERVAL(ONETIME) PARM('STAR') DATE('date_of_the_change')
       REASON('Your reason for making the update.')
```

Parameters accepted:
Yes, you can specify the mode required, either STAR, RING, or NONE. For example, `PARM('STAR')`

Default : STAR

Reference:
For more information on GRS, see *z/OS MVS Planning: Global Resource Serialization*.

Messages:
This check issues the following exception messages:

- ISGH0301E
- ISGH0303E

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

GRS_SYNCHRES

Description:
Checks whether GRS synchronous reserve processing is enabled.

Reason for check:
Enabling GRS synchronous reserve processing can prevent deadlock conditions.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMGRS,GRS_SYNCHRES)
       SEVERITY(LOW) INTERVAL(001:00) DATE('date_of_the_change')
       REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information on GRS synchronous reserve processing, see *z/OS MVS Planning: Global Resource Serialization*.

Messages:
This check issues the following exception messages:

- ISGH0305E

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

GRS_CONVERT_RESERVES

Description:
Whether RESERVEs are being converted to global ENQs in STAR mode

Reason for check:
Converting RESERVEs to global ENQs can help avoid deadlocks and improve reliability, availability, and serviceability.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMGRS,GRS_CONVERT_RESERVES)
     SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
     REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Parameters accepted:
No

Reference:
For more information on GRS Reserve Conversion, see *z/OS MVS Planning: Global Resource Serialization*.

Messages:
This check issues the following exception messages:

- ISGH0307E

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

GRS_EXIT_PERFORMANCE

Description:
Checks to see if there are GRS dynamic exits in use that could degrade system performance.

Reason for check:
The use of certain GRS dynamic exits can degrade system performance. In some cases, removing an exit module or changing it to use a different exit point can help improve performance.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMGRS,GRS_EXIT_PERFORMANCE)
      SEVERITY(LOW) INTERVAL(024:00) DATE(20050105)
      REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Reference:
For more information on GRS installation exits, see *z/OS MVS Planning: Global Resource Serialization* and*z/OS MVS Installation Exits*.

Messages:
This check issues the following exception messages:

- ISGH0309E
- ISGH0310E
- ISGH0311E
- ISGH0312E
- ISGH0313E

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

GRS_GRSQ_SETTING

Description:
Examines the system's GRSQ setting. The check generates an exception if the GRSQ setting is not set to CONTENTION in a GRS=STAR mode environment. This is a one-time check that is also run during a migration from GRS=RING to GRS=STAR.

Reason for check:
Having a GRSQ setting of CONTENTION will shorten the amount of time required for SVC Dump processing.

z/OS releases the check applies to:
z/OS V1R8 and later.

Type of check:
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMGRS,GRS_GRSQ_SETTING)
SEVERITY(LOW) INTERVAL(ONETIME) DATE(20050202)
REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on GRSQ, see *z/OS MVS Planning: Global Resource Serialization*.

Messages:
This check issues the following exception messages:

- ISGH0315E

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

GRS_RNL_IGNORED_CONV

Description:
Searches the RESERVE Conversion RNL for entries that will be ignored because of a matching or equivalent entry in the SYSTEMS Exclusion RNL.

Reason for check:
There should be no duplicate entries between the RESERVE Conversion RNL and SYSTEMS Exclusion RNL. Duplicate entries may result in undesired serialization of a resource.

z/OS releases the check applies to:
z/OS V1R8 and later.

Type of check:
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMGRS,GRS_RNL_IGNORED_CONV)
SEVERITY(LOW) INTERVAL(ONETIME) DATE(20050202)
REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on Global Resource Serialization RNL's, see*z/OS MVS Planning: Global Resource Serialization*.

Messages:
This check issues the following exception messages:

- ISGH0317E

See the ISGH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

HSM checks (IBMHSM)

HSM_CDSB_BACKUP_COPIES

Description:
Determines if DFSMShsm is configured to maintain a critical level of Control Data Set (CDS) backups. The check determines the number of CDS backup copies that have been specified to DFSMShsm via the SETSYS CDSVERSIONBACKUP(BACKUPCOPIES(x)) command.

This check is registered with IBM Health Checker for z/OS when the MAIN DFSMShsm host initializes on a z/OS image for the first time since IPL.

The check will run after a MAIN DFSMShsm host initializes, the CDS backup function completes on a MAIN or AUX host, a SETSYS CDSVERSIONBACKUP is issued to a MAIN or AUX DFSMShsm host (SETSYS CDSVERSIONBACKUP commands processed during MAIN host initializatin will not cause the checks to run).

This check requires a MAIN DFSMShsm host to be active on the z/OS image. If this check runs while a MAIN DFSMShsm host is not active, the check will be disabled until a MAIN host initializes.

Reason for check:
It is recommended that DFSMShsm maintain a minimum of 4 CDS backups. This practice greatly minimizes the exposure of all valid backups rolling off over the course of an extended off-shift period such as a three-day weekend.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMHSM,HSM_CDSB_BACKUP_COPIES)
SEVERITY(MED) INTERVAL(24:00) PARM('CRITVAL(4)')
DATE(20071031)
REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Parameters accepted:
Yes. The critical value of backups to check for can be adjusted. For example, PARM('CRITVAL(10)').

Default: 4 Maximum: 9999

Note that specifying a value of 0 means the check will run but will not issue an exception.

Verbose support:
No

Debug support:
No

Reference:
For more information on DFSMShsm CDS Backup, see the following:

- Defining the Backup Environment for Control Data Sets in *z/OS DFSMShsm Implementation and Customization Guide*.
- *z/OS DFSMSdfp Storage Administration*

Messages:
This check issues the following exception messages:

- ARCHC0108E

See the ARCH messages in *z/OS MVS System Messages, Vol 2 (ARC-ASA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

HSM_CDSB_DASD_BACKUPS

Description:
If DFSMShsm Control Data Set backups are created on DASD, this check will ensure all required DASD backup data sets are in place.

This check is registered with IBM Health Checker for z/OS when the MAIN DFSMShsm host initializes on a z/OS image for the first time since IPL.

The check will run after a MAIN DFSMShsm host initializes, the CDS backup function completes on a MAIN or AUX host, a SETSYS CDSVERSIONBACKUP is issued to a MAIN or AUX DFSMShsm host(SETSYS CDSVERSIONBACKUP commands processed during MAIN host initializatin will not cause the checks to run).

This check requires a MAIN DFSMShsm host to be active on the z/OS image. If this check runs while a MAIN DFSMShsm host is not active, the check will be disabled until a MAIN host initializes.

Reason for check:
When backing up CDS's to DASD, DFSMShsm requires all backup data sets to be pre-allocated. If one or more data set becomes unavailable, the CDS Backup function may fail. A failure during CDS Backup may lead to critical functions to be held within DFSMShsm.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMHSM,HSM_CDSB_DASD_BACKUPS)
SEVERITY(HI) INTERVAL(24:00) DATE(20071031)
REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Parameters accepted:
No

Verbose support:
No

Debug support:
No

Reference:
For more information on DFSMShsm CDS Backup, see the following:

- Defining the Backup Environment for Control Data Sets in *z/OS DFSMShsm Implementation and Customization Guide*.
- *z/OS DFSMSdfp Storage Administration*

Messages:
This check issues the following exception messages:

- ARCHC0111E

See the ARCH messages in *z/OS MVS System Messages, Vol 2 (ARC-ASA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

HSM_CDSB_VALID_BACKUPS

Description:
Determines if number of valid Control Data Set (CDS) backups has fallen below a critical level.

This check is registered with IBM Health Checker for z/OS when the MAIN DFSMShsm host initializes on a z/OS image for the first time since IPL.

The check will run after a MAIN DFSMShsm host initializes, the CDS backup function completes on a MAIN or AUX host, a SETSYS CDSVERSIONBACKUP is issued to a MAIN or AUX DFSMShsm host(SETSYS CDSVERSIONBACKUP commands processed during MAIN host initializatin will not cause the checks to run).

This check requires a MAIN DFSMShsm host to be active on the z/OS image. If this check runs while a MAIN DFSMShsm host is not active, the check will be disabled until a MAIN host initializes.

For multi-cluster CDS's, all cluster backups for a version must be valid before the version is considered a valid backup.

Reason for check:
It is recommended that DFSMShsm maintain a minimum of 4 CDS backups. This practice greatly minimizes the exposure of all valid backups rolling off over the course of an extended off-shift period such as a three-day weekend.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMHSM,HSM_CDS_VALID_BACKUPS)
  SEVERITY(MED) INTERVAL(24:00) PARM('CRITVAL(4)')
  DATE(20071031)
  REASON('YOUR REASON FOR MAKING AN UPDATE.')
```

Parameters accepted:
Yes. The critical value of backups to check for can be adjusted. For example, PARM('CRITVAL(10)').

Default: 4 Maximum: 9999

Note that specifying a value of 0 means the check will run but will not issue an exception.

Verbose support:
No

Debug support:
No

Reference:
For more information on DFSMShsm CDS Backup, see the following:

- Defining the Backup Environment for Control Data Sets in *z/OS DFSMShsm Implementation and Customization Guide*.
- *z/OS DFSMSdfp Storage Administration*

Messages:
This check issues the following exception messages:

- ARCHC0108E
- ARCHC0115E

See the ARCH messages in *z/OS MVS System Messages, Vol 2 (ARC-ASA)*. .

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSF checks (IBMICSF)

ICSF_COPROCESSOR_STATE_NEGCHANGE

Description:

Detects a degradation in the state of any cryptographic coprocessor or accelerator on the system. This check is activated when ICSF is started. This check will be performed on a daily basis.

Reason for check:

A degradation in the state of any cryptographic coprocessor or accelerator on the system can have a possible negative impact on the operation of ICSF and the dependant cryptographic workload. This checks allows for the detection of degradation in the state of any cryptographic coprocessor or accelerator on the system.

z/OS releases the check applies to:

ICSF FMID HCR7790 or later running on z/OS V1R12, z/OS V1R13 or z/OS V2R1

Type of check (local or remote):

Local

User override of IBM values:

The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF,ICSF_COPROCESSOR_STATE_NEGCHANGE)
ACTIVE
SEVERITY(MEDIUM) INTERVAL(DAILY) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:

No.

Verbose support:

No

Debug support:

No

Reference:

For more information see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Messages:

This check issues the following exception messages:

- CSFH0010E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:

SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSF_MASTER_KEY_CONSISTENCY

Description:

Detects inconsistencies in the states of the coprocessor master keys. This check

is activated when ICSF is started and is performed on a daily basis. The check determines when the state of a master key on at least one coprocessor is not in accord with the state of the other coprocessors.

Reason for check:
The check is instituted to assist in maintaining master key functionality. The coprocessor activation algorithm maximizes the number of active cryptographic coprocessors.

z/OS releases the check applies to:
ICSF FMID HCR77A0 or later running on z/OS V1R12, z/OS V1R13 or z/OS V2R1

Type of check (local or remote):
Local

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF, ICSF_MASTER_KEY_CONSISTENCY)
ACTIVE
SEVERITY(MEDIUM) INTERVAL(DAILY) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:
No.

Verbose support:
No

Debug support:
No

Reference:
For more information see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Messages:
This check issues the following exception messages:

- CSFH0015E
- CSFH0016E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSFMIG_DEPRECATED_SERV_WARNINGS

Description:
Detects the use of services which will not be supported in subsequent releases of ICSF. This check is inactive when ICSF is started it must be activated to perform the check. Once activated the check will be performed on a daily basis.

Reason for check:
Subsequent releases of ICSF will not support the use of certain callable services. This check provides a way to detect if any applications are currently using these services.

The deprecated services checked in ICSF FMID HCR77A0 (z/OS V2R1) are listed below. These are not supported after System z 900 hardware.

- CSFAEGN
- CSFAKEX
- CSFAKIM
- CSFAKTR
- CSFATKN
- CSFCTT
- CSFCTT1
- CSFTCK
- CSFUDK
- CSFPKSC

z/OS releases the check applies to:
ICSF FMID HCR77A0 or later running on z/OS V1R12, z/OS V1R13 or z/OS V2R1

Type of check (local or remote):
Local

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF,ICSFMIG_DEPRECATED_SERV_WARNINGS)
INACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:
No.

Verbose support:
No

Debug support:
No

Reference:
For more information see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Messages:
This check issues the following exception messages:

- CSFH0011I

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSFMIG7731_ICSF_RETAINED_RSAKEY

Description:

Detects the existence of retained RSA private keys on a PCICC or PCIXCC/CEX2C cryptographic card.

This check is inactive by default - in order to use this check you must activate it. You should run the check periodically, when the events occur that affect check results. For example, run the check dynamically when:

- The ICSF product release level is being upgraded to any new ICSF release level.
- The z/OS product release level is being upgraded and ICSF is an exploited feature for that z/OS image.

Reason for check:

A PCICC or PCIXCC/CEX2C card may possess the only copy of a retained RSA private key. Customers that run applications and middleware that utilize the retained key functionality of these cards are exposed to the loss of keys upon hardware failure, which may result from a problem as simple as an exhausted or malfunctioning card battery. Lost retained keys have the further implication of lost data, for retained key management keys, and an inability to verify signatures, for retained signature keys. Starting with the Cryptographic Support for z/OS V1R7-V1R9 and z/OS.e V1R7-V1R8 Web deliverable (ICSF FMID HCR7750), you no longer have the ability to store new private RSA keys intended for key management usage in a cryptographic coprocessor. Existing applications will continue to be able to use the retained keys and to delete them from the cryptographic coprocessor cards.

z/OS releases the check applies to:

z/OS V1R9 and later.

Type of check (local or remote):

Local

User override of IBM values:

The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF, ICSFMIG7731_ICSF_RETAINED_RSAKEY)
INACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:

No.

Verbose support:

No

Debug support:

No

Reference:

For more information see *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Messages:

This check issues the following exception messages:

- CSFH0003E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSFMIG7731_ICSF_PKDS_TO_4096BIT

Description:
Verifies that the PKDS size in an ICSF pre-HCR7750 environment is sufficiently allocated to support 4096-bit RSA keys.

This check is inactive by default - in order to use this check you must activate it. You should run the check periodically, when the events occur that affect check results. For example, run the check dynamically when:

- The ICSF product release level is being upgraded to HCR7750 or later.
- The z/OS product release level is being upgraded and ICSF is an exploited feature for that z/OS image.

Reason for check:
ICSF FMID HCR7750 introduces support for 4096-bit RSA keys, which requires a larger PKDS than prior ICSF releases needed. If a customer at a pre-HCR7750 FMID ICSF level migrates to HCR7750 without first reallocating the PKDS for 4096-bit key support, ICSF at HCR7750 will fail to start. This ICSF migration check will detect the case where the currently active PKDS is not sufficiently allocated for the HCR7750 environment and inform the customer that a PKDS reallocation action is necessary.

z/OS releases the check applies to:
z/OS V1R8 (or ICSF FMID HCR7731) and z/OS V1R9 (or ICSF FMID HCR7740).

Type of check (local or remote):
Local

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF, ICSFMIG7731_ICSF_PKDS_TO_4096BIT)
INACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Parameters accepted:
No.

Verbose support:
No

Debug support:
No

Reference:
For more information see the following:

- *z/OS Cryptographic Services ICSF Administrator's Guide*
- *z/OS Cryptographic Services ICSF System Programmer's Guide*

Messages:
This check issues the following exception messages:

- CSFH0005E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSFMIG77A1_COPROCESSOR_ACTIVE

Description:
Detects cryptographic coprocessors that will not become active when starting HCR77A1. This checks compares the coprocessor master keys against the CKDS and PKDS.

This check is inactive by default – in order to use this check you must activate it. You should run this check on your system before installing the HCR77A1 release of ICSF.

Reason for check:
A coprocessor that has master keys that do not match the CKDS and PKDS will not become active when ICSF FMID HCR77A1 is started. This will affect the availability of coprocessors for cryptographic work. The method to decide which coprocessors become active changed for HCR77A1 and later.

z/OS releases the check applies to:
ICSF FMID HCR7770 or later running on z/OS V1R9, z/OS V1R10, z/OS V1R11, z/OS V1R12, z/OS V1R13 or z/OS V2R1 with PTFs for APAR OA42011 applied.

Type of check (local or remote):
Local

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF,ICSFMIG77A1_COPROCESSOR_ACTIVE)
INACTIVE
SEVERITY(MEDIUM) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:
No.

Verbose support:
No

Debug support:
No

Reference:
For more information see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Messages:
This check issues the following exception messages:

- CSFH0020E

- CSFH0021E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSFMIG77A1_TKDS_OBJECT

Description:
Detects any TKDS object that is too large to allow the TKDS to be read into storage during ICSF initialization starting with ICSF FMID HCR77A1.

This check is inactive by default – in order to use this check you must activate it. You should run this check on your system before installing the HCR77A1 release of ICSF.

Reason for check:
In ICSF FMID HCR77A1, ICSF introduces new common KDS record format for CCA key tokens and PKCS #11 tokens and objects. This new format of the record adds new fields for key utilization and metadata. Because of the size of the new fields, some PKCS #11 objects in the TKDS may cause ICSF to fail to start. This check will detect any TKDS object that is too large to allow the TKDS to be loaded when ICSF is started.

z/OS releases the check applies to:
ICSF FMID HCR7770 or later running on z/OS V1R9, z/OS V1R10, z/OS V1R11, z/OS V1R12, z/OS V1R13 or z/OS V2R1 with PTFs for APAR OA42011 applied.

Type of check (local or remote):
Local

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF, ICSFMIG77A1_TKDS_OBJECT)
INACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:
No.

Verbose support:
No

Debug support:
No

Reference:
For more information see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Messages:
This check issues the following exception messages:

- CSFH0025E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ICSFMIG77A1_UNSUPPORTED_HW

Description:
Detects if system is supported by ICSF FMID HCR77A1.

This check is inactive by default – in order to use this check you must activate it. You should run this check on your system before installing the HCR77A1 release of ICSF.

Reason for check:
ICSF FMID HCR77A1 does not support IBM Eserver System z 800 and 900 systems.

z/OS releases the check applies to:
ICSF FMID HCR7770 or later running on z/OS V1R9, z/OS V1R10, z/OS V1R11, z/OS V1R12, z/OS V1R13 or z/OS V2R1 with PTFs for APAR OA42011 applied.

Type of check (local or remote):
Local

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFYcommand. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMICSF, ICSFMIG77A1_UNSUPPORTED_HW)
INACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.'))
```

Parameters accepted:
No.

Verbose support:
No

Debug support:
No

Reference:
For more information see *z/OS Cryptographic Services ICSF Overview*.

Messages:
This check issues the following exception messages:

- CSFH0022E

See in *z/OS Cryptographic Services ICSF Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Infoprint Server checks (IBMINFOPRINT)

INFOPRINT_PRINTWAY_MODE

Description:
Checks the type of Infoprint (IP) PrintWay™ mode that is being used. Generates an exception if it detects use of Infoprint Basic mode.

Reason for check:
Infoprint Basic mode was stabilized in z/OS V1R5 and no further enhancements will be done. In future releases, IBM will make enhancements only to IP PrintWay extended mode. IP PrintWay extended mode uses the SYSOUT Application Programming Interface (SAPI) to obtain output data sets from the JES spool.

IP PrintWay extended mode provides better performance, improved usability, and additional functions.

z/OS releases the check applies to:
z/OS V1R8 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMINFOPRINT, INFOPRINT_PRINTWAY_MODE)
  SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
No.

Verbose support:
No.

Reference:
For more information, see:

- *z/OS Infoprint Server Customization*
- *z/OS Infoprint Server User's Guide*
- *z/OS Infoprint Server Introduction*
- *z/OS Infoprint Server Operation and Administration*
- *z/OS Migration*

Messages:
This check issues the following exception messages:

- ANFH001E

See *z/OS Infoprint Server Messages and Diagnosis*.

SECLABEL recommended for MLS users:
SYSLOW

INFOPRINT_V2DB_CHECK

Description:
The INFOPRINT_V2DB_CHECK provides information for customers who have migrated to z/OS V1R12 and then fallen back to an earlier z/OS release. The health check examines the Printer Inventory base directory (default is /var/Printsrv) looking for Version 2 Printer Inventory files that were created by Infoprint Server on z/OS V1R12. Infoprint Server on z/OS V1R12 uses only the Version 2 Printer Inventory files. Both the Version 1 and Version 2 Printer Inventory files may exist. However, Infoprint Server on z/OS V1R10 or V1R11 uses only the Version 1 Printer Inventory files. If the Version 2 Printer Inventory files are found, the health check issues an exception to warn the customer that the files exist but are not being updated. Before migrating back to z/OS V1R12 again, IBM suggests that you remove file master.v2db file from the base directory so that Infoprint Server creates a new Version 2 Printer Inventory. Any changes that the administrator has made in the Version 1 Printer Inventory after falling back to z/OS V1R10 or V1R11 will be included in the new Version 2 Printer Inventory. However, if you want to keep any changes that the administrator made in the Version 2 Printer Inventory before falling back to z/OS V1R10 or V1R11, do not remove file master.v2db. If file master.v2db exists, Infoprint Server on z/OS V1R12 does not create a new Version 2 Printer Inventory.

Reason for check:
The INFOPRINT_V2DB_CHECK provides information for customers who have migrated to z/OS V1R12 and then fallen back to an earlier z/OS release. It warns the customer that the Version 2 Printer Inventory files may become stale.

z/OS releases the check applies to:
z/OS V1R10 and V1R11.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMINFOPRINT, INFOPRINT_V2DB_CHECK)
  USS(YES)
  VERBOSE(NO)
  SEVERITY(LOW)
  INTERVAL(ONETIME)
  ACTIVE
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides additional detail in debug mode. You can put a check into debug mode using any of the following: UPDATE, filters, DEBUG=ON parameters on either the MODIFY command or in policy statement in an HZSPRMxx parmlib member.

Verbose support:
No.

Reference:
For more information, see:

- *z/OS Migration*

Messages:

This check issues the following exception messages:

- AOPH1501E
- AOPH1503E
- AOPH1504E

See *z/OS Infoprint Server Messages and Diagnosis*.

SECLABEL recommended for MLS users:

SYSLOW

Output:

The following shows sample output of the exception because Version 2 Printer Inventory files were found in the base directory.

```
CHECK(IBMINFOPRINT,INFOPRINT_V2DB_CHECK)
START TIME: 05/05/2010 15:30:10.317783
CHECK DATE: 20100301  CHECK SEVERITY: LOW
             Printer Inventory Master.V2DB Existence Report
  Inv  Base
S Name Directory Name
- ---- ----------------------------------------------------------------
E AOP1 /var/Printsrv

* Low Severity Exception *
  AOPH1501E The Infoprint Server base directory contains Version 2 Printer
Inventory files.

  Explanation:  File master.v2db was found in the Infoprint Server base
    directory. This file indicates that Infoprint Server has reformatted
    the Version 1 Printer Inventory and created a Version 2 Printer
    Inventory on z/OS V1R12. The Version 1 Printer Inventory and the
    Version 2 Printer Inventory both exist in the Infoprint Server base
    directory. However, Infoprint Server on z/OS V1R10 or V1R11 uses
    only the Version 1 Printer Inventory.

    Before you start Infoprint Server on z/OS V1R12 again, IBM suggests
    that you remove file master.v2db so that Infoprint Server creates a
    new Version 2 Printer Inventory. Any changes that the administrator
    has made in the Version 1 Printer Inventory after falling back to
    z/OS V1R10 or V1R11 will be included in the new Version 2 Printer
    Inventory. However, if you want to keep any changes that the
    administrator made in the Version 2 Printer Inventory before falling
    back to z/OS V1R10 or V1R11, do not remove file master.v2db. If file
    master.v2db exists, Infoprint Server on z/OS V1R12 does not create a
    new Version 2 Printer Inventory.

  System Action:  Processing continues.

  Operator Response:  Not applicable.

  System Programmer Response:  Examine the report that this check
    produced. An "E" in the "S"(Status) column indicates that the
    master.v2db file exists in the indicated base directory. If desired,
    remove file master.v2db from the base directory. To remove
    master.v2db, you must have an effective UID of 0 or be a member of
    the RACF AOPADMIN group.

  Problem Determination:  Not applicable

  Source:   Infoprint Server

  Reference Documentation:  See the z/OS Migration Guide.
```

```
  Automation:

  Check Reason:  Warn if .v2db inventory files exist in the base
    directory.  Infoprint Server releases prior to V1R12 will ignore
    these files.

END TIME: 05/05/2010 15:30:10.324205  STATUS: EXCEPTION-LOW
================================================================================
```

ZOSMIGV1R12_INFOPRINT_INVSIZE

Description:
This check verifies that Infoprint Server has sufficient space to create Version 2 Printer Inventory files.

Before activating this check, verify that Infoprint Server is active.

Reason for check:
When first started on z/OS V1R12, the Infoprint Server will create Version 2 Printer Inventory files from the Version 1 Printer Inventory files. Before starting the conversion, Infoprint Server verifies that there is sufficient space to successfully complete. If there is not sufficient space, Infoprint Server halts and issues a message that there is insufficient space. The minimum available space that is required is two times the space used by the Version 1 Printer Inventory files. (Version 1 Printer Inventory files have the extension .db.) This check determines if there is sufficient space for the conversion.

z/OS releases the check applies to:
z/OS V1R10 and V1R11.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
   CHECK(ZOSMIGV1R12_INFOPRINT_INVSIZE)
   USS(YES)
   VERBOSE(NO)
   SEVERITY(MED)
   INTERVAL(ONETIME)
   INACTIVE
   DATE('date_of_the_change')
   REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides additional detail in debug mode. You can put a check into debug mode using any of the following: UPDATE, filters, DEBUG=ON parameters on either the MODIFY command or in policy statement in an HZSPRMxx parmlib member.

Verbose support:
No.

Reference:
For more information, see:

- *z/OS Migration*

Messages:
This check issues the following exception messages:

- AOPH1511E
- AOPH1503E
- AOPH1504E

See *z/OS Infoprint Server Messages and Diagnosis*.

SECLABEL recommended for MLS users:
SYSLOW

Output:
The following shows sample output of the exception due to insufficient space to create the Version 2 Printer Inventory files:

```
CHECK(IBMINFOPRINT,ZOSMIGV1R12_INFOPRINT_INVSIZE)
START TIME: 05/05/2010 09:29:49.720815
CHECK DATE: 20100301  CHECK SEVERITY: MEDIUM
                Printer Inventory Migration Space Report
  Inv  File                                          (MB)
S Name System Name                                   Avail  Needed  Used
- ---- ------------------------------------------- ------ ------ ------
E AOP1 VAR.PRINTSR1.ZFS                               1470   2049   2341
  /var/Printsrv

* Medium Severity Exception *

AOPH1511E The Infoprint Server file system has insufficient space to
reformat the Printer Inventory when you migrate to z/OS V1R12.

  Explanation:  The Infoprint Server file system does not have enough
    available space for the Version 2 Printer Inventory. The first time
    you start Infoprint Server on z/OS V1R12, Infoprint Server attempts
    to reformat the Version 1 Printer Inventory and create a Version 2
    Printer Inventory. If insufficient space exists, Infoprint Server
    does not start.  The minimum available space that is required is 2
    times the space that the Version 1 Printer Inventory files currently
    use. (Version 1 Printer Inventory files have extension .db.)  The
    health check produced a report that identifies the Infoprint Server
    file system with its total available space and used space plus the
    minimum available space required to create the Version 2 Printer
    Inventory.

 System Action:  Processing continues. However, if you do not increase
   the available space, Infoprint Server cannot start on z/OS V1R12.

 Operator Response:  Report this problem to the system programmer.

 System Programmer Response:  Examine the report that the health check
   produced.

   o An "E" in the "S"(Status) column indicates that the file system
   does not have enough available space.

   o An "N" in the "S"(Status) column indicates that file master.v2db
   exists. The amount of available space was not checked because
   Version 2 Printer Inventory files already exist.

   After you increase the space in the file system, run this check
    again to verify that enough available space exists.

  Problem Determination:  Not applicable

  Source:   Infoprint Server

  Reference Documentation:  z/OS Migration Guide

  Automation:

  Check Reason:  Verify there is enough space to create the .v2db
    inventory files from the .db files when migrating to V1R12.

END TIME: 05/05/2010 09:29:49.735897  STATUS: EXCEPTION-MED
============================================================================
```

IOS checks (IBMIOS)

IOS_CAPTUCB_PROTECT

Description:
This check verifies that captured UCB protection is active on the system. Captured UCB protection is suggested.

Reason for check:
UCBs (Unit Control Blocks) are control blocks in storage that define the characteristics of devices. Legacy software may require a subset of these to reside in the first 16 megabytes of storage. To accommodate 24-bit addressability there is a service to capture the UCB and temporarily put the UCB in the 24-bit addressable area. Captured UCB Protection places the UCBs temporarily below the line in write protected storage, so legacy software cannot interfere with the state of the devices. IBM recommends that Captured UCB Protection is active.

You can verify the state of captured UCB protection using the following console command:

```
DISPLAY IOS,CAPTUCB
```

In response to this command, the system issues message IOS088I to display the state of captured UCB protection:

```
IOS088I 12.12.00 CAPTURED UCB DATA 243
CAPTURED UCB PROTECTION IS DISABLED
```

To change the state, see the SETIOS command in *z/OS MVS System Commands*.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
Yes, the following parameters are accepted:

PARM('PROTECT(*state*)')
Since, there may be a rare instance where captured UCB protection should not be enabled, you can set the expected *state* of captured UCB protection the check expects to either:

- PARM('PROTECT(YES)'), which is the default, indicates that the expected captured UCB protection state is enabled.
- PARM('PROTECT(NO)') indicates that the expected captured UCB protection state is disabled.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
 UPDATE
        CHECK(IBMIOS,IOS_CAPTUCB_PROTECT )
        INACTIVE
        SEVERITY(MED) INTERVAL(ONETIME) DATE('date_of_the_change')
```

Debug support:
No.

Verbose support:
No.

Reference:
For more information, see information on the SETIOS command in *z/OS MVS System Commands*.

Messages:
This check issues the following exception messages:

- IOSHC101E

See IOSHC messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

IOS_CMRTIME_MONITOR

Description:
Detects if any control units in the system are reporting inconsistent average initial command response (CMR) time for their attached channel paths. The check issues an exception if at least one control unit in the system has a path with an average CMR time that is the highest among the other paths to the control unit and meets the following conditions:

- The average CMR time for this path is greater than the THRESHOLD check parameter value.
- The average CMR time for this path is significantly higher (as defined by the RATIO check parameter) than the path with the lowest average CMR time for this control unit. That is, the average CMR time for this path is at least '*x*' times the lowest average CMR time for this control unit (where '*x*' is the RATIO check parameter value).

  Example:

  ```
  Path 1, average CMR time = 11ms
  Path 2, average CMR time = 3 ms
  Path 3, average CMR time = 2 ms
  Path 4, average CMR time = 4 ms
  ```

  If THRESHOLD is 3ms and RATIO is 5, the check issues an exception because path 1 has a CMR time (11ms) that is greater than 3ms and is also greater than 5 times the CMR time for path 3 (2ms).

Reason for check:
Initial Command Response (CMR) time is a component of Response time and measures the round trip delay of the fabric alone with minimal channel and control unit involvement and thus can be a symptom of potential problems in the fabric. By monitoring this measurement alone and comparing it among the paths to a control unit, fabric problems like hardware errors, misconfiguration and congestion may be more easily detected.

z/OS releases the check applies to:
z/OS V1R10 and later.

Parameters accepted:
Yes, the following parameters are accepted:

`PARM('THRESHOLD(`*threshold*`),RATIO(`*x*`),XTYPE(`*devtype*`),XCU(`*cu1*`,`*cu2*`,...,`*cux*`)')`

THRESHOLD(*threshold***)**
THRESHOLD defines the value in milliseconds that is used in conjunction with the RATIO parameter to determine whether an exception exists. If the path with the highest average CMR time is greater than the THRESHOLD value, then the RATIO value is used to further determine if an exception exists.

A THRESHOLD value of 0 means the highest average CMR time can be any value and exceptions will be declared as defined by the RATIO value alone.

Range: 0 to 100

Default: 3

RATIO(*x*)
RATIO defines the value used to determine if the path with the highest average CMR time is significantly higher than the path with the lowest average CMR time for this control unit using a factor of '*x*'. This is used to determine if an exception exists only after the THRESHOLD condition has been met.

If the THRESHOLD condition has been met and if the path with the highest average CMR time is at least '*x*' times greater than the path with the lowest average CMR time, an exception will be declared for the control unit.

Range: 2 to 100

Default: 5

XTYPE(*devtype*)
devtype is the device type of control units that will be excluded from the check and not reported on.

Supported device types: DASD,TAPE

Default: no value

XCU(*cu1*,*cu2*,...,*cux*)
XCU defines a list of specific control units that will be excluded from the check and will not be reported on. Each control unit in this list is a hexadecimal value representing the control unit number. This parameter takes up to 40 different control unit numbers.

Range: 0 to FFFE

Default: no value

Note that if any parameter is changed, the check results may not reflect these changes for several minutes because the check must gather a few minutes worth of data before performing analysis using the new parameters.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
       CHECK(IBMIOS,IOS_CMRTIME_MONITOR)
       ACTIVE
       VERBOSE(NO)
       INTERVAL(00:05)
       SEVERITY(MED)
       DATE('date_of_the_change')
       PARM('THRESHOLD(3),RATIO(5),XCU(),XTYPE()')
       REASON('Your reason for making the update')
```

Debug support:
No.

Verbose support:
Yes, if VERBOSE(YES) is specified on the check, the control units that were excluded via the XTYPE and XCU parameters will be displayed in the report if exceptions were found for them. This allows an easy way to temporarily obtain information on ALL control units with an exception without the need for a change to the XCU and XTYPE parameters.

Reference:
For more information on interpreting initial command response (CMR) time for the affected control units, see "IOQUEUE - I/O Queuing Activity Report" in *z/OS RMF Report Analysis*.

Messages:
This check issues the following exception messages:

- IOSHC112E

See IOSHC messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

IOS_FABRIC_MONITOR

Description:
Reports if any switches which support CUP diagnostics capabilities have indicated unusual health conditions on connected channel paths.

Typically, a fabric network requires careful planning, implementation, and maintenance in order to provide healthy operation. When a switch determines that there is a performance problem with a path, it will signal to z/OS that such a problem has been found and that diagnostic data is available. This action results in the route being monitored by z/OS. At regular intervals, z/OS will obtain health diagnostic data. This health check report contains the formatted diagnostic data.

The check issues an exception if a switch has indicated to z/OS there is a possible health problem. The report will be updated at regular intervals with the latest diagnostic data until two hours after the switch stops reporting a health problem.

Example behavior:

A switch with CUP diagnostics support signals to z/OS that there is a health problem. z/OS initiates monitoring for the path, requesting diagnostic data from the switch at regular intervals. The problem may require intervention, i.e. MVS system commands or I/O configuration. Once no errors or health issues have been detected by the switch for at least 2 hours, the monitoring of the route is stopped and will no longer appear in the report.

Report information:

The report will indicate path information, the time of the exception, and the diagnostic data provided by the switch.

Reason for check:
While z/OS has historically provided sophisticated I/O monitoring and recovery, this report will expose newly available diagnostic data provided directly from the switch. This health check may be able to provide insight into their fabric problems such as hardware errors, I/O mis-configurations, or congestion.

z/OS releases the check applies to:
z/OS V1R12 and later with apar OA40548 and supporting switch hardware.

Parameters accepted:
Yes, the following parameters are accepted:

```
PARM('LOG(NO|YES),SHOW(LATEST|ALL)')
```

LOG(NO|YES)
LOG determines if z/OS should request that a diagnostic log should be generated by the switch. The diagnostic log will only be generated when the first unusual condition is detected if the YES option us used. The switch log is not generated if the NO option is used.

The switch log may be useful in diagnosing the fabric problem and its contents are defined by the switch vendor.

Supported values: NO and YES

Default: NO

SHOW(LATEST|ALL)
SHOW determines how many diagnostic records should be formatted in the health check report. If LATEST is used, only the most recent diagnostic data is reported. If ALL is used, all diagnostic records since monitoring began will be provided.

The historical data may be useful in diagnosing the problem.

Supported values: LATEST and ALL

Default: LATEST

Note: If any parameter is changed, the check results may not reflect these changes for several minutes because the check must gather a few minutes worth of data before performing analysis using the new parameters.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMIOS,IOS_FABRIC_MONITOR)
    ACTIVE
    VERBOSE(NO)
    INTERVAL(00:30)
    SEVERITY(MED)
    DATE('date_of_the_change')
    PARM('SHOW(LATEST),LOG(NO)')
    REASON('Your reason for making the update')
```

Debug support:
No.

Verbose support:
No.

Reference:
For more information on interpreting switch diagnostic data, please consult your hardware vendor's documentation.

Messages:
This check issues the following exception messages:

- IOSHC119E

See IOSHC messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

IOS_IORATE_MONITOR

Description:
Detects if any control units in the system are reporting inconsistent I/O rates for their attached channel paths.

Typically, I/Os are distributed equally across all paths for a control unit. When the system determines that there is a performance problem with a path, it will direct I/Os away from that path. This action, taken by the system to correct the performance problem, results in inconsistent I/O rates across the paths.

The check issues an exception if at least one control unit in the system has a total I/O rate across all of its channel paths that exceeds the THRESHOLD check parameter value, and at least one path with an I/O rate significantly lower (as defined by the RATIO check parameter) than that of the channel path with the highest I/O rate for the control unit.

Example:

```
Path 1,  I/O rate = 600 I/Os per second
Path 2,  I/O rate = 250 I/Os per second
Path 3,  I/O rate = 300 I/Os per second
```

If THRESHOLD is 800 and RATIO is 2, the check issues an exception because the total I/O rate of 1150 exceeds the threshold value, and path 2, (the path with the lowest I/O rate of 250) is less than half the I/O rate for path 1 (the path with the highest I/O rate).

Reason for check:
I/O rate measures the number of I/Os started down the channel path per second. A lower than average I/O rate can be a symptom of potential problems in the fabric. By monitoring this measurement alone and comparing it among the paths to a control unit, fabric problems like hardware errors, misconfiguration and congestion may be more easily detected.

z/OS releases the check applies to:
z/OS V1R12 and later with apar OA40548 on a zEC12 or later processor.

Parameters accepted:
Yes, the following parameters are accepted:

```
PARM('THRESHOLD(threshold),RATIO(x),XTYPE(devtype),XCU(cu1,cu2,...,cux)')
```

THRESHOLD(*threshold*)
THRESHOLD defines the value in number of I/Os per second that is used in conjunction with the RATIO parameter to determine whether an exception exists. If the total I/O rate of all of the paths to the control unit exceed the THRESHOLD value, then the RATIO value is used to further determine if an exception exists.

Range: 10 to 1000

Default: 100

RATIO(*ratio*)
RATIO defines the value used to determine if the I/O rate of the path with the lowest I/O rate is significantly lower than the path with the highest

I/O rate for this control unit, using a factor of 'ratio'. This is used to determine if an exception exists only after the THRESHOLD condition has been met.

If the THRESHOLD condition has been met and if the path with the lowest I/O rate is at least a factor of 'ratio' less than the path with the highest I/O rate, an exception will be declared for the control unit.

Range: 2 to 100

Default: 2

XTYPE(*devtype*)
devtype is the device type of control units that will be excluded from the check and not reported on.

Supported device type values: DASD,TAPE

Default: no value

XCU(*cu1*,*cu2*,...,*cux*)
XCU defines a list of specific control units that will be excluded from the check and will not be reported on. Each control unit in this list is a hexadecimal value representing the control unit number. This parameter takes up to 40 different control unit numbers.

Range: 0 to FFFE

Default: no value

Note: If any parameter is changed, the check results may not reflect these changes for several minutes because the check must gather a few minutes worth of data before performing analysis using the new parameters.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMIOS,IOS_IORATE_MONITOR)
    ACTIVE
    VERBOSE(NO)
    INTERVAL(00:05)
    SEVERITY(MED)
    DATE('date_of_the_change')
    PARM('THRESHOLD(100),RATIO(2),XCU(),XTYPE()')
    REASON('Your reason for making the update')
```

Debug support:
No.

Verbose support:
Yes, if VERBOSE(YES) is specified on the check, the control units that were excluded via the XTYPE and XCU parameters will be displayed in the report if exceptions were found for them. This allows an easy way to temporarily obtain information on ALL control units with an exception without the need for a change to the XCU and XTYPE parameters.

Reference:
For more information on interpreting initial command response (CMR) time for the affected control units, see "IOQUEUE - I/O Queuing Activity Report" in z/OS RMF Report Analysis.

Messages:
This check issues the following exception messages:

- IOSHC132E

See IOSHC messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

IOS_MIDAW

Description:
This check verifies that the modified indirect addressing word (MIDAW) facility is enabled. MIDAWs are a more efficient way to issue I/O commands.

Reason for check:
MIDAW is a extension to I/O that allows better use of I/O bandwidth.

You can verify whether the MIDAW facility is enabled using the following console command:

```
DISPLAY IOS,MIDAW
```

In response to this command, the system issues message IOS097I to display whether MIDAW is active:

```
IOS097I 12.27.47 MIDAW FACILITY 279
MIDAW FACILITY IS ENABLED
```

To change MIDAW enablement, see the SETIOS command in *z/OS MVS System Commands*.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
Yes, the following parameters are accepted:

PARM('MIDAW(*state*)')
Since, there may be a rare instance when MIDAW should not be enabled, you can set the expected *state* of MIDAW enablement the check expects to either:

- PARM('MIDAW(YES)'), which is the default, indicates that the expected state the check expects is MIDAW enabled.
- PARM('MIDAW(NO)') indicates that the state expected by the check is MIDAW disabled.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMIOS,IOS_MIDAW)
    INACTIVE
    SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Debug support:
No.

Verbose support:
No.

Reference:
For more information, see information on the SETIOS command in *z/OS MVS System Commands*.

Messages:
This check issues the following exception messages:

- IOSHC105E

See IOSHC messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

IOS_STORAGE_IOSBLKS

Description:
This check verifies that control blocks used in IOS can reside in 31-bit addressable storage.

Reason for check:
Control blocks used to initiate I/O were obtained in storage addressable in the first 16 megabytes of storage so that 24-bit AMODE legacy software could perform scans on them. However, forcing all I/O control blocks below the 16 megabyte line creates a constraint on the amount of storage below the line. IBM recommends that the control blocks be allowed to be in 31-bit addressable storage.

You can verify the state of IOS blocks using the following console command:

```
DISPLAY IOS,STORAGE
```

In response to this command, the system issues message IOS089I to display the state of captured UCB protection:

```
IOS089I 12.21.54 STORAGE DATA 246
IOS BLOCKS RESIDE IN 31 BIT STORAGE
```

To change the state of the IOS blocks, see the SETIOS command in *z/OS MVS System Commands*.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
Yes, the following parameters are accepted:

PARM('IOSBLKS(*state*)')
Since, there may be a rare instance where 31-bit IOS Blocks should not be enabled, you can set the expected state of 31-bit IOS blocks the check expects to either:

- PARM('IOSBLKS(31)') which is the default, indicates that the expected state for IOS blocks is 31–bit.
- PARM('IOSBLKS(24)') indicates that the expected state for IOS blocks is 24–bit.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
     CHECK(IBMIOS,IOS_STORAGE_IOSBLKS)
     INACTIVE
     SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
     REASON('Your reason for making the update.')
```

Debug support:
No.

Verbose support:
No.

Reference:
For more information, see information on the SETIOS command in *z/OS MVS System Commands*.

Messages:
This check issues the following exception messages:

- IOSHC103E

See IOSHC messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

JES2 checks (IBMJES2)

JES2_Z11_Upgrade_CK_JES2

Description:
This check determines if the system is ready to upgrade the JES2 checkpoint to z11 mode. If the necessary preconditions are already satisfied on the customer's system, then a message is given recommending that the system be upgraded to z11 mode. In the case where prerequisite conditions have not been satisfied, an exception message is given indicating that the system is not ready to upgrade to checkpoint level z11.

Reason for check:
IBM suggests that installations use JES2 z11 mode because it upgrades the JES2 checkpoint and enables the following functions:

- SAPI (SSI 79) and Extended Status (SSI 80) can now support selection by transaction job name and/or transaction job id.
- Extended Status (SSI 80) can now return transaction information (job name, job id and owner) within the terse SYSOUT section.
- JOE data area extensions supported by BERTs. Initial JOE data area extension includes transaction job name and transaction job ID.
- Fixed JOE extension termed the JOX. JOX only exists in artificial JOE returned by the new $DOGJOE service.
- Increased the size of the JQX by 32 bytes.
- Increased limits for JQEs, JOEs and BERTs. New limits are as follows:
  - JQEs = 400,000
  - JOEs = 1,000,000
  - BERTs = 1,000,000

  These new limits allow for more jobs and pieces of output to be handled by the Job Entry Subsystem.

z/OS releases the check applies to:
z/OS V1R11 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE   CHECK(IBMJES2,JES2_Z11_Upgrade_CK_JES2)
  SEVERITY(LOW) INTERVAL(168:00) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
No.

Verbose support:
No.

Reference:
For more information, see:

- *z/OS JES2 Messages*
- *z/OS JES2 Initialization and Tuning Guide*
- *z/OS JES2 Commands*

Messages:
This check issues the following exception messages:

- HASP022E
- HASP028E

See *z/OS JES2 Messages*.

SECLABEL recommended for MLS users:
SYSLOW

Loadwait/Restart checks (IBMSVA)

SVA_AUTOIPL_DEFINED

Description:
Checks if the customer environment is capable of supporting an AutoIPL policy and if it is, determines whether the AutoIPL policy is active.

Reason for check:
IBM suggests that you define the AutoIPL policy using the DIAGxx parmlib member to minimize z/OS system downtime. AutoIPL can re-IPL MVS, take a Stand Alone Dump (SADMP), or take a SADMP and have SADMP re-IPL MVS when it has finished.

AutoIPL function requires the Program-Directed IPL feature. AutoIPL is not appropriate in GDPS environment. If the check determines that the customer does not have the hardware feature or that the AutoIPL policy is not active, the check stops running.

z/OS releases the check applies to:
z/OS V1R11

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMSVA,SVA_AUTOIPL_DEFINED)
ACTIVE
SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Parameters accepted:
No.

Reference:

- For information about Exploiting the Automatic IPL Function, see *z/OS MVS Planning: Operations*.
- For information about DIAGxx (Control common storage tracking and GFS trace) see *z/OS MVS Initialization and Tuning Reference*

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- BLWH0001E
- BLWH0011E

See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:
The following report is generated by the SVA_AUTOIPL_DEV_VALIDATION check when the device validation fails for devices specified in the AutoIPL policy:

```
AutoIPL  Device     Error
action   Address    Description

-------  ---------  ------------------------------------------------------------
SADMP    03A0       Device is not DASD

In the output:
AutoIPL action =  The AutoIPL action (SADMP or MVS).
Device Address =  The address of the device failing the device validation.
Error Description = The description of the problem
```

SVA_AUTOIPL_DEV_VALIDATION

Description:
Performs device validation for device(s) specified in the AutoIPL policy for SADMP and/or MVS when an AutoIPL policy exists. Reports problems if the device validation fails. If the check determines that there is no AutoIPL policy defined, or that the customer does not have the appropriate hardware feature, the check stops running.

Reason for check:
If an AutoIPL policy exists all of the following conditions must be met for the device to pass device validation:

- The device must exist
- The device must be accessible
- The device must be DASD
- The device must not be specified as a secondary device in a Metro Mirror pair. Use the report generated from this check to determine if there are problems with device(s) specified in the AutoIPL policy.

z/OS releases the check applies to:
z/OS V1R11

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMSVA,SVA_AUTOIPL_DEV_VALIDATION)
ACTIVE
SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Parameters accepted:
No.

Reference:

- For information about Exploiting the Automatic IPL Function, see *z/OS MVS Planning: Operations*.
- For information about DIAGxx (Control common storage tracking and GFS trace) see *z/OS MVS Initialization and Tuning Reference*

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- BLWH0002E

See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

PDSE checks (IBMPDSE)

PDSE_SMSPDSE1

Description:
The PDSE_SMSPDSE1 check returns the current status of the SMSPDSE1 address space.

Reason for check:
IBM recommends that SMSPDSE1 address be set to active to prevent possible PDSE related problems.

z/OS releases the check applies to:
z/OS V1R6 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
   CHECK(IBMPDSE,PDSE_SMSPDSE1),
   SEVERITY(LOW),
       INTERVAL(ONETIME),
       DATE('date_of_the_change')
```

Debug support:
No.

Verbose support:
No

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGWPH0101E

See the IGWPH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

Predictive failure analysis checks (IBMPFA)

Predictive failure analysis (PFA) is described in *z/OS Problem Management*, including the following:

- The PFA checks are described in Predictive Failure Analysis checks .
- General information about PFA is described as follows:
  - Predictive failure overview and installation
  - Managing PFA checks

RACF checks (IBMRACF)

Write your own RACF resource checks!

You can create your own RACF installation-defined resource checks to see if your resources have the security characteristics you want. Do the following for each check you wish to create:

1. Define a RACF profile containing a list of the resources you want your RACF installation-defined resource check to look at, along with the maximum allowable general user access you want for each resource.

The check raises an exception if the profile which covers the resource allows more than the specified access or there is no profile covering the resource and default return code from the class is not 8. If you would like to prevent the exception, define a profile which allows less access than indicated in the installation-defined check. You can use a generic profile.

The format of each member list entry in the profile is as follows:

className/resourceName/volume/maxUacc

className
The class of the resource which is to be checked. Valid values are DATASET and any RACF general resource class which is defined on the system.

Note that if the general resource class is a member/grouping class, the class must already be RACLISTed.

resourceName
The name of the resource which is to be checked.

volume
If the *className* is DATASET then this is volume upon which the data set resides. This parameter is optional. If it is not specified, then the catalog is searched to find the volume serial for the dataset.

If the *className* is **not** DATASET, do not specify a volume. If you specify a volume for a *className* other than DATASET, you will receive an error message.

maxUacc
The maximum allowed general user access to the resource.

The following shows an example of a profile for a RACF installation-defined resource check:

```
RDEFINE  RACFHC MY_RESOURCE_LIST
         ADDMEM(DATASET/PROD.VALUABLE.DATA/ZDR17B/NONE
         DATASET/SEC.FILING.FORMS//NONE
         DATASET/PUBLIC.REPORTS/REGVOL/READ
         RACFHC/MY_RESOURCE_LIST//NONE)
```

ADDMEM member list entry considerations:

- You can specify any number of resource names up to the maximum amount of data which can be placed into the member list portion of a profile using the ADDMEM operand.
- Only the following types of data sets are allowed to be specified as resources: Sequential, partitioned, library, or VSAM data sets.

Special values you can use in ADDMEM: To make defining your profile easier, you can also use the following special values in ADDMEM:

Value	Description
IRR_APFLIST	Examines all of the entries in the current APF list.
IRR_LINKLIST	Examines all of the entries in the current link list
IRR_PARMLIB	Examines all of the entries in the current PARMLIB
IRR_RACFDB	Examines the current primary and backup RACF databases
IRR_SYSREXX	Examines all of the SYSREXX data sets
IRR_ICHAUTAB	Examines the entries in ICHAUTAB

If you specify one of these special ADDMEM values, you cannot specify any other value, such as *className*, *resourceName*, *volume*, or *maxUacc* on that entry.

Note that the system does not validate the content of your profile when you add or alter it. The system verifies the profile only when the check runs. The system processes ADDMEM values in the following order:

- The syntax of the entire member list is validated
- The reports are processed.
- The individual resource names are processed

Possible profile errors reported when your RACF profile is validated:

Basic parameter errors: The system validates the following

- If the required RESOURCELIST keyword has not been specified, the system issues message HZS1201E.
- If the RESOURCELIST value is greater than 128 characters, the system issues message HZS1213E.
- If the RESOURCELIST value has not been specified, then the system issues message HZS1201E.
- If the USER value is greater than 8 characters, the system issues message HZS1213E.
- If the USER value has not been specified, then the system issues message HZS1201E.

The system also issues messages for situations such as unexpected parameter. In this case, the system issues message IRRH231I in addition to one of the messages above, and the check is placed in the "parameter error" state.

The profile specified as the RESOURCELIST does not exist: If the profile does not exist or cannot be retrieved, the system issues messages IRRH232I and HZS1001E and the check is placed in the "parameter error" state. The check will not run until the installation corrects the situation by defining the profile or modifying the value specified in the PARM statement on the check on the check registration to point to a properly defined profile and the reactivating the check.

The profile exists, but the profile does not have any ADDMEM value: If there is no member list, the system issues message IRRH233I and HZS1001E and the check is placed into "parameter error" state. The check will not run until the installation corrects the situation by adding a correct member list or modifying the value specified in the PARM statement on the check on the check registration to point to a properly defined profile and the reactivating the check.

The profile exists with ADDMEM values, but there is an error in the ADDMEM member list entry, such as :

- The specified class does not exist.
- The length of the resource name does not match the maximum value allowed for the class
- A volume serial was specified for a class other than data set
- The volume serial value is greater than six (6) characters
- The maximum "general user" access level a value other than "NONE", "READ", "UPDATE", "ALTER", or "CONTROL"

For any ADDMEM entry with any of these errors, the system issues messages IRRH234I and HZS1001E and places the check into "parameter error" state. The check will not run until the installation corrects the situation by adding a correct member list or modifying the value specified in the PARM statement on the check on the check registration to point to a properly defined profile and

the reactivating the check. Message IRRH234I is issued once for each member list entry which is in error. The message contains the entry number of the incorrect member entry.

2. Choose a name for your RACF installation-defined resource check, and using this name, define the check to IBM Health Checker for z/OS in an HZSPRMxx parmlib member. The following shows an example of registering a RACF installation-defined resource check in HZSPRMxx:

```
ADD CHECK(USER01,MY_INSTALLATION_HEALTH_CHECK)
    CHECKROUTINE(IRRHCR00)
    MESSAGETABLE(IRRHCM00)
    ENTRYCODE(100)
    PARM('USER(USER01)  RESOURCELIST(MY_RESOURCE_LIST)')
    DATE('date_of_the_change')
    REASON('Your reason for making the update.')
    GLOBAL
    ACTIVE
    SEVERITY(HIGH)
    INTERVAL(08:00)
```

Reason for check:
Installation defined.

z/OS releases the check applies to:
z/OS V1R10 and later.

Parameters accepted:
Yes, the following parameters are accepted for an RACF installation-defined resource check:

PARM('USER(*userid*)')
Optional parameter specifies an individual user ID whose the authority to the resources listed in the profile the check will examine.

PARM('RESOURCELIST(*resourcelist_profile)')
Required parameter specifies the resource list profile name defined for this check. The check then examines the authority levels for the resources listed in the profile named in this parameter.

The following shows an example of a PARM statement specified in a RACF installation-defined resource check:

```
PARM('USER(USER01)  RESOURCELIST(MY_RESOURCE_LIST)')
```

User override of IBM values:
There are no IBM default values for a RACF installation-defined resource check. You can override the check values you defined with either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command.

Debug support:
Yes

Verbose support:
No

Reference:
For more information on storage increments, see *z/OS Security Server RACF System Programmer's Guide*.

Messages:
This check issues the following exception messages:

- IRRH237E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The following shows output from a RACF installation-defined resource check:

```
CHECK(USER01,MY_INSTALLATION_HEALTH_CHECK)
START TIME: 01/10/2008 14:35:34.674057
CHECK DATE: 20070425  CHECK SEVERITY: HIGH
CHECK PARM: USER(USER01)  RESOURCELIST(MY_RESOURCE_LIST)


                  Resource List from MY_RESOURCE_LIST

S Resource Name                   Class    Vol    UACC Warn ID*  User
- ------------------------------ -------- ------ ---- ---- ---- ----
V PROD.VALUABLE.DATA              DATASET  ZDR17B
V SEC.FILING.FORMS                DATASET
V PUBLIC.REPORTS                  DATASET  REGVOL
  MY_RESOURCE_LIST                RACFHC          None No   ****

* High Severity Exception *

IRRH237E The MY_INSTALLATION_HEALTH_CHECK check has found
one or more potential errors in the
security controls for the installation-defined
resources specified in this check.

  Explanation:  The RACF security configuration check has found one or
    more potential errors with the protection mechanisms for the
    resources specified for this check.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator and the system auditor.

...
...
...


  Check Reason:  My sensitive resources

END TIME: 01/10/2008 14:35:34.701104  STATUS: EXCEPTION-HIGH
```

RACF_AIM_STAGE

Description:
The RACF_AIM_STAGE check examines the RACF database application identity mapping (AIM) to see whether it is at AIM stage 3, which is recommended. Your system programmer can convert your RACF database to AIM stage 3 using the IRRIRA00 conversion utility.

Reason for check:
AIM stage 3 allows RACF to more efficiently handle authentication and authorization requests from applications such as z/OS UNIX and is required to use some RACF function. You should assign a unique UNIX UID for each user and a unique GID for each group that needs access to z/OS UNIX functions and resources. Assigning unique IDs rather than shared IDs improves overall security and increases user accountability. However, if you have a large number of users without OMVS segments who need access to z/OS UNIX

services, such as FTP, you might choose not to assign UNIX identities in advance of their need to use the services. In these cases, when your RACF database has been converted to AIM stage 3, you can enable RACF to automatically assign unique UNIX UIDs and GIDs at the time they are needed.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
No

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,RACF_AIM_STAGE)
SEVERITY(MED),INTERVAL(24:00),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:

- For information on running the IRRIRA00 conversion utility, see *z/OS Security Server RACF System Programmer's Guide*.
- For information about enabling RACF for automatic assignment of unique UNIX identities, see *z/OS Security Server RACF Security Administrator's Guide*.

Messages:
This check issues the following exception messages:

- IRRH501E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

- The following shows the output from a RACF_AIM_STAGE check that finds the system at stage 3:

```
CHECK(IBMRACF,RACF_AIM_STAGE)
START TIME: 05/06/2011 10:51:02.926675
CHECK DATE: 20110101  CHECK SEVERITY: MEDIUM

IRRH500I The RACF database is at the suggested stage of application
identity mapping (AIM). The database is at AIM stage 03.


END TIME: 05/06/2011 10:51:02.927390  STATUS: SUCCESSFUL
```

- The following shows the output from an exception for RACF_AIM_STAGE:

```
CHECK(IBMRACF,RACF_AIM_STAGE)
START TIME: 05/06/2011 11:06:27.618944
CHECK DATE: 20110101  CHECK SEVERITY: MEDIUM

* Medium Severity Exception *

IRRH501E The RACF database is not at the suggested stage of application
```

```
identity mapping (AIM). The database is at AIM stage 00.

  Explanation:  The RACF_AIM_STAGE check has determined that the RACF
    database is not at the suggested stage of application identity
    mapping (AIM). Your system programmer can convert your RACF database
    using the IRRIRA00 conversion utility. See z/OS Security Server RACF
    System Programmer's Guide for information about running the IRRIRA00
    conversion utility.

    Stage 3 of application identity mapping allows RACF to more
    efficiently handle authentication and authorization requests from
    applications such as z/OS UNIX and is required to use some RACF
    function. You should assign a unique UNIX UID for each user and a
    unique GID for each group that needs access to z/OS UNIX functions
    and resources. Assigning unique IDs rather than shared IDs improves
    overall security and increases user accountability. However, if you
    have a large number of users without OMVS segments who need access
    to z/OS UNIX services, such as FTP, you might choose not to assign
    UNIX identities in advance of their need to use the services. In
    these cases, when your RACF database has been converted to AIM stage
    3, you can enable RACF to automatically assign unique UNIX UIDs and
    GIDs at the time they are needed. See z/OS Security Server RACF
    Security Administrator's Guide for information about enabling RACF
    for automatic assignment of unique UNIX identities.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator.

  System Programmer Response:  If you want to use RACF function such as
    support for automatically assigning unique UNIX UIDs and GIDs at the
    time that they are needed, run the IRRIRA00 utility to advance the
    RACF database to application identity mapping stage 3. For details
    about using the IRRIRA00 utility, see z/OS Security Server RACF
    System Programmer's Guide.

  Problem Determination:

  Source:

  Reference Documentation:
    z/OS Security Server RACF System Programmer's Guide
    z/OS Security Server RACF Security Administrator's Guide

  Automation:  None.

  Check Reason:  AIM Stage 3 is suggested.

END TIME: 05/06/2011 11:06:27.620454  STATUS: EXCEPTION-MED
```

RACF_CERTIFICATE_EXPIRATION

Description:

The RACF_CERTIFICATE_EXPIRATION check:

- Extracts each certificate from the RACF database.
- Examines the ending date of the certificate and lists the certificate in the check output if the ending date is equal to or less than the warning date. The warning date is the current date adjusted by the "warning period" that the installation has specified as a parameter to the check.
- If the certificate is either a TRUST or HIGHTRUST then the certificate is marked as an exception.

The RACF_CERTIFICATE_EXPIRATION check has the following columns in its report:

Table 45. RACF_CERTIFICATE_EXPIRATION report columns

Column	Description
s	The status of the certificate. This column contains an "E" if the certificate is marked as an exception.
Cert Owner	This column contains the "anchor point" for the certificate. Valid values are "SITE", "CERTAUTH", and "ID(user-ID)."
Certificate Label	This is the label that has been assigned to the certificate.
End Date	The end date assigned to the certificate. This is the date after which the certificate is not valid.
Trust	The trust status of the certificate. Valid values are "No", "Yes", and "High".
Rings	The number of rings with which this certificate is associated.

If there are no certificates selected for inclusion in the report, then only the title and headers are presented, along with the "No exceptions found" message.

Note: The check end date and the current date are evaluated as follows:

- If the CERTEND date/time is earlier than the current date/time, then the certificate is considered "expired".
- If the CERTEND date/time is not earlier than the current date/time, then the current date/time value is subtracted from the CERTEND date/time and the result converted to minutes. This value is compared to the number of days in the warning period multiplied by the number of minutes in a day (1440).

The RACF_CERTIFICATE_EXPIRED check is registered with these attributes:

Table 46. RACF_CERTIFICATE_EXPIRED attributes

Attribute	Setting
Severity	Medium
State	Active
Interval	Run once a day on each system
Date	20111010
Reason	Operational certificates should not be allowed to expire.
Parameter	DAYS(nnn), where "nnn" is between 0 and 366, with a default of 60 if DAYS is not specified explicitly.

Reason for check:
RACF_CERTIFICATE_EXPIRATION allows RACF to identify all certificates which have expired, identify all certificates which are going to expire within the next few days, and ensures that the user has defined a proper baseline set of protections within the z/OS environment.

z/OS releases the check applies to:
z/OS V2R1 and later.

Parameters accepted:
The value of DAYS(nnn), where "nnn" is between 0 and 366.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE
CHECK(IBMRACF,RACF_CERTIFICATE_EXPIRATION)
ACTIVE
SEVERITY(MED)
DATE('20111010')
REASON('Operational certificates should not expire.')
INTERVAL(24:00)
```

Debug support:
No

Verbose support:
No

Reference:

- For information on running the IRRIRA00 conversion utility, see *z/OS Security Server RACF System Programmer's Guide*.
- For information about enabling RACF for automatic assignment of unique UNIX identities, see *z/OS Security Server RACF Security Administrator's Guide*.

Messages:
This check issues the following exception messages:

- IRRH276E
- IRRH277I

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:
The following shows the output from a RACF_CERTIFICATE_EXPIRATION check:

```
CHECK(IBMRACF,RACF_CERTIFICATE_EXPIRATION)
START TIME: 01/23/2012 08:10:01.603497
CHECK DATE: 20111010 CHECK SEVERITY: MEDIUM
                    Certificates Expiring in 60 Days
S Cert Owner  Certificate Label                 End Date  Trust Rings
- ------------ -------------------------------- ---------- ----- -----
IRRH277I No exceptions are detected. Expired certificates that are not
trusted or are associated with only a virtual key ring are not
exceptions.
END TIME: 01/23/2012 08:10:01.643285  STATUS: SUCCESSFUL
```

RACF_GRS_RNL

Description: Check evaluates whether the RACF ENQ names are in either the installation system exclusion resource name list (SERNL) or the system inclusion resource name list (SIRNL).

During its normal course of processing, RACF performs numerous serialization requests using the Global Resource Serialization (GRS) RESERVE, ENQ, and DEQ services. These serialization requests allow RACF to ensure that changes to the RACF database and RACF control blocks are done in a consistent manner, maintaining the integrity of RACF data.

Depending on the type of the serialization that RACF requires, RACF serializes at either the address space (SCOPE=STEP), single MVS image (SCOPE=SYSTEM) or multiple MVS image/Sysplex level (SCOPE=SYSTEMS). GRS identifies a serialization request by an eight character QNAME (or major name) and RNAME (or minor name) of up to 255 characters.

GRS allows installations to tailor the processing of RESERVE, ENQ, and DEQ requests through the use of Resource Name Lists (RNLs). RNLs allow an installation to influence the scope of RESERVE, ENQ, and DEQ processing. GRS supports three types of RNLs:

- The System Inclusion RNL (SIRNL), which promotes a local ENQ (SCOPE=SYSTEM) to a global ENQ (SCOPE=SYSTEMS
- The System Exclusion RNL (SERNL), which demotes a global ENQ (SCOPE=SYSTEMS) to a local ENQ (SCOPE=SYSTEM)
- The Reserve Conversion (RCRNL), which suppresses a hardware RESERVE, in effect allowing it to be a global (SCOPE=SYSTEMS) ENQ

The RACF service team has debugged several customer problems and outages and found that the problem or outage was caused by a customer's RNL changing the scope of a RACF serialization request. With z/OS V1R6, GRS introduced an enhanced ISGQUERY service which allows an application to specify the QNAME and RNAME of an ENQ and determine if the ENQ name is on an RNL.

RACF's ENQ names fall into three general categories:

- Names which consist of constant values, such as the SYSZRACF/RACF ENQ
- Names which consist of values, which the check can easily determine, such as SYSZRACF/*racf_data_set_name* or SYSZRAC2/RACGLIST_*classname*
- Names which consist of values which the check cannot easily determine, such as SYSZRAC2/IRRDPI08*hhhh* where *hhhh* is a hexadecimal address. However, since ISGQUERY supports wildcard characters when searching for entries in the RNL, many of these cases can be detected.

The RACF_GRS_RNL check produces a report which identifies the RACF ENQs would have their scope changed by an entry in a GRS RNL. For SYSTEMS level ENQs, the RACF_GRS_RNL check flags as error that match entries in the SERNL. For a SYSTEM level ENQ, the RACF_GRS_RBL check flags as errors RACF ENQ names which matches entries in the SIRNL.

When it runs, the RACF_GRS_RNL check calls the GRS ISGQUERY service for each of the ENQ names documented in Table 47 and Table 48 on page 477. If one or more ENQs are on an RNL that affects the scope of the ENQ, then the RACF_GRS_RNL check identifies the ENQs that have their scope changed.

Table 47. **Systems** *Level ENQs that RACF_GRS_RNL checks*

Major Name	Minor Name
SYSZRACF	*racf_data_set_name* RACF data set names are derived from the data set name table on which the check executes. The check looks at all of the data sets in the primary RACF data base as well as all of the data sets in the backup RACF data base.
SYSZRACF	SETROPTS
SYSZRACF	DSDTDSDTDSDT...DSDT The minor name is the string DSDT repeated twelve times.

Table 47. **Systems** *Level ENQs that RACF_GRS_RNL checks (continued)*

Major Name	Minor Name
SYSZRACF	DSDTPREP...DSDTPREP The minor name is the string DSDTPREP repeated six times.
SYSZRAC2	IRRCV05
SYSZRAC2	RACGLIST_*class_name* *class_name* is derived from the list of classes defined on the system upon which the check executes.
SYSZRAC2	GLOBALGLOBALGLOBAL
SYSZRAC2	PROGRAMPROGRAMPROGRAM
SYSZRAC2	TEMPLATE-LOCK
SYSZRAC4	BPX.NEXT.USER
SYSZRAC5	ALIAS
SYSZRAC5	IRRIRA00

Table 48. **System** *Level ENQs that RACF_GRS_RNL checks*

Major Name	Minor Name
SYSZRAC2	SSTABLE1
SYSZRAC2	SSTABLE2
SYSZRACF	RACF
SYSZRACF	CNSTGNLP**class_name* *class_name* is derived from the list of classes defined on the system upon which the check executes.
SYSZRACF	CNSTRCLP**class_name* *class_name* is derived from the list of classes defined on the system upon which the check executes.
SYSZRACF	*racf_data_set_name* RACF data set names are derived from the data set name table on which the check executes. The check looks at all of the data sets in the primary RACF data base as well as all of the data sets in the backup RACF data base.
SYSZRACF	DSDTDSDTDSDT...DSDT The minor name is the string DSDT repeated twelve times.
SYSZRAC2	IRRCV05
SYSZRACF	CNSTRCLP**class_name* *class_name* is derived from the list of classes defined on the system upon which the check executes.
SYSZRACF	CNSTRCLP**class_name* *class_name* is derived from the list of classes defined on the system upon which the check executes.
SYSZRAC2	DSDTABPT0000
SYSZRAC2	ICHSEC00

Table 48. **System** *Level ENQs that RACF_GRS_RNL checks (continued)*

Major Name	Minor Name
SYSZRAC2	IRRDPI80000
SYSZRAC2	RCVTDPTB000
SYSZRAC2	XMCAXMCA...XMCA The minor name is the string XMCA repeated twelve times.
SYSZRAC2	CONNECT...CONNECT The minor name is the string CONNECT repeated six times.

Reason for check:
Installations that convert RACF SYSTEM ENQs to SYSTEM ENQs can corrupt the RACF data base and experience outages.

z/OS releases the check applies to:
z/OS V1R5 and later.

Parameters accepted:
No

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,RACF_GRS_RNL)
SEVERITY(HI),INTERVAL(08:00),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides output displays all the ENQ names being looked at plus additional error detail in debug mode. You can put a check into debug mode using any of the following:

- UPDATE,filters,DEBUG=ON parameters on either the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member
- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF..

Verbose support:
Yes, the check output displays all the ENQ names being looked at in verbose mode. You can put a check into verbose mode using the UPDATE,filters,VERBOSE=ON parameters on either the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.

Reference:
For more information on storage increments, see *z/OS MVS Planning: Global Resource Serialization* and *z/OS Security Server RACF System Programmer's Guide*.

Messages:
This check issues the following exception messages:

- IRRH202E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The report that RACF_GRS_RNL produces is shown below. The columns in this report are as follows:

S Status. An E in this column indicates an exception.

Major
The major name of the ENQ

Minor
The minor name of the ENQ

Type
The type of the ENQ. SERNL indicates that the ENQ is a SYSTEMS-level ENQ and that it was found on the system exclusion resource name list, which would change its scope to SYSTEM-level and potentially destroy RACF's serialization.

Qname
The QNAME of the RNL entry

Rname
The RNAME of the RNL entry

Type
The type of the RNL entry. The values are SPEC for specific and GEN for generic

RACF_GRS_RNL check report with exceptions:

```
START TIME: 11/10/2004 10:13:10.341622  IBMRACF, RACF_GRS_RNL
OWNER DATE: 20040703



                          RACF_GRS_RNL Report

 S Major    Minor                Type  QName    Rname              Type
 - -------- -------------------- ----- -------- ------------------ ----
 E SYSZRACF SETROPTS             SERNL SYSZRACF SETROPTS           SPEC
 E SYSZRAC2 IRRCRV05             SERNL SYSZRAC2 IRRCRV05           SPEC
 E SYSZRAC2 IRRCRV05             SIRNL SYSZRAC2 IRRCRV05           SPEC
 E SYSZRAC5 ALIAS                SERNL SYSZRAC5 AL                 GEN
* High severity Exception *

 IRRH202E   One or more RACF ENQ names were found in a GRS Resource Name
     List.

   Explanation:
     The RACF RACF_GRS_RNL check has detected that a RACF resource
     is covered by an entry in the specified GRS resource name list
     (RNL). RACF resource names should not be in either the system
     inclusion RNL (SIRNL) or the system exclusion RNL (SERNL).

   System Action:
     The check continues processing. There is no effect on the system.

   Operator Response:
     Report this problem to the system programmer.

 System Programmer Response:
    Ensure that the RACF resource names are removed from the specified
    resource name list (RNL).

  Problem Determination:
    See "MVS Planning: Global Resource Serialization" for details on
    resource name lists (RNLs). Ensure that the RACF ENQ names do not
    match any of your resource name list entries. A list of the RACF
    ENQ names may be found in the RACF Systems Programmer's Guide.
```

```
 Source:
   RACF Systems Programmer's Guide

 Reference documentation:
   RACF Systems Programmer's Guide MVS Planning: Global Resource
   Serialization
  Automation:
    None.

  IBMRACF Reason: None of the RACF ENQ names should be in RNLs.
  Check parameters: N/A

END TIME: 01/08/2005 20:47:54.819710 RESULT: 0000000C DIAG:
00000000_00000000
```

RACF_GRS_RNL check report without exceptions:

```
START TIME: 11/14/2004 23:11:39.610978  IBMRACF, RACF_GRS_RNL
OWNER DATE: 20040703

                        RACF_GRS_RNL Report

 S Major    Minor                Type  QName    Rname             Type
 - -------- -------------------- ----- -------- ----------------- ----

IRRH203I   No RACF ENQ names were found in the GRS Resource Name List.



END TIME: 11/14/2004 23:11:39.613687 RC: 00000000 RSN: 00000000
```

RACF_GRS_RNL check report in debug mode:

```
START TIME: 11/14/2004 23:17:12.648857  IBMRACF, RACF_GRS_RNL
OWNER DATE: 20040703

                         RACF_GRS_RNL Report

 S Major    Minor                Type  QName    Rname             Type
 - -------- -------------------- ----- -------- ----------------- ----
   SYSZRACF SETROPTS             SERNL
   SYSZRACF DSDTDSDTDSDTDSDTDSDT SERNL
   SYSZRACF DSDTPREPDSDTPREPDSDT SERNL
   SYSZRACF RACF                 SIRNL
   SYSZRACF DSDTDSDTDSDTDSDTDSDT SIRNL
   SYSZRAC2 IRRCRV05             SERNL
   SYSZRAC2 GLOBALGLOBALGLOBAL   SERNL
   SYSZRAC2 TEMPLATE-LOCK        SERNL
   SYSZRAC2 PROGRAMPROGRAMPROGRA SERNL
   . . .
```

RACF_GRS_RNL check report in a GRS=NONE environment:

```
START TIME: 11/18/2004 22:29:54.701040  IBMRACF, RACF_GRS_RNL
OWNER DATE: 20040703


IRRH201I   The RACF check RACF_GRS_RNL cannot be executed in a
    GRS=NONE environment.



HZS1004E (IBMRACF,RACF_GRS_RNL)
    THE CHECK IS NOT APPLICABLE IN THE CURRENT SYSTEM ENVIRONMENT.

END TIME: 11/18/2004 22:29:54.861360 RC: 00000000 RSN: 00000000
```

RACF_ICHAUTAB_NONLPA

Description:
The RACF_ICHAUTAB_NONLPA check examines the RACF Authorized Caller Table (ICHAUTAB) and reports if there are any non-LPA entries in it. The output format is similar to the report format for the ICHAUTAB Report in RACF_SENSITIVE_RESOURCES, with the exception that LPA-resident modules are not listed.

Reason for check:
IBM recommends that installations have no entries in the ICHAUTAB table.

z/OS releases the check applies to:
z/OS V1R10 and later.

Type of check:
Local

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMRACF,RACF_ICHAUTAB_NONLPA)
  SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
z/OS Security Server RACF System Programmer's Guide

Messages:
This check issues the following exception messages:

- IRRH240E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for MLS users:
SYSLOW

Output: The following shows ICHAUTAB Non-LPA report:

- Successful case:

```
CHECK(IBMRACF,RACF_ICHAUTAB_NONLPA)
START TIME: 03/14/2008 15:52:22.756461
CHECK DATE: 20070411  CHECK SEVERITY: MEDIUM

                        ICHAUTAB Non-LPA Report

S Module   REQUEST= REQUEST= Location
           VERIFY   LIST
- -------- -------- -------- --------
```

```
IRRH239I There are no ICHAUTAB programs on this system.

END TIME: 03/14/2008 15:52:22.762403  STATUS: SUCCESSFUL
```

- Exception case:

```
START TIME: 11/13/2007 18:42:44.876179
CHECK DATE: 20070411  CHECK SEVERITY: MEDIUM

                        ICHAUTAB Non-LPA Report

S Module   REQUEST= REQUEST= Location
           VERIFY   LIST
- -------- -------- -------- --------
  TRESPOND YES      YES      NON-LPA

* Medium Severity Exception *

IRRH240E The RACF_ICHAUTAB_NONLPA check has found
one or more non-LPA ICHAUTAB entries.
non-LPA ICHAUTAB entries. IBM recommends that ICHAUTAB contain no
entries. An entry in ICHAUTAB represents a program whose access
should be controlled using PROGRAM CONTROL and restricted to a known
set of trusted users or trusted started tasks.

LPA-resident ICHAUTAB entries are listed in the
RACF_SENSITIVE_RESOURCES check.

System Action:  The check continues processing. There is no effect on
the system.

Operator Response:  None.

System Programmer Response:  If the modules in ICHAUTAB are no longer
in use, they should be deleted from ICHAUTAB. If the modules are
still in use and the privileges granted by ICHAUTAB are still
required, the modules should be protected using PROGRAM CONTROL and
their use should be restricted to a known set of trusted users or
trusted started tasks.

Problem Determination:

Source:

Reference Documentation:
  IBM Health Checker for z/OS: User's Guide
  z/OS Security Server RACF Security Administrator's Guide

Automation:  None.

Check Reason:  ICHAUTAB entries must be protected.
END TIME: 11/13/2007 18:42:44.885582  STATUS: EXCEPTION-MED
```

RACF_SENSITIVE_RESOURCES

Description: The RACF_SENSITIVE_RESOURCES check examines the security characteristics of several system-critical data sets and general resources other than data sets. The output of this check is a list of exceptions flagged.

For each of these, the check examines:

- For system-critical data sets, that the data set exists on the expected volume. If the data set does not exist on the volume, a V (volume exception) is placed in the Status (S) column.

- That the resource has baseline protection. For example, APF data sets can have a general access as high as READ, while the data sets which comprise the RACF data base must have a general access of NONE.

The check verifies the protection of each resource by extracting its profile and examining the UACC, WARNING status, and the ID(*) entry in the access list if one exists. In addition, if there is no profile protecting a data set, then if NOPROTECTALL or PROTECTALL(WARN) is in effect, the check flags the data set as an exception. The customer can optionally specify a user ID to the check which, if specified, is used to perform a RACF authorization check for the next higher access authority after the highest expected general access authority.

The RACF_SENSITIVE_RESOURCES check offers the following resources:

Table 49. Additional Discrete General Resources for RACF_SENSITIVE_RESOURCES

Class	Resource	Maximum Public Access
FACILITY	BPX.DEBUG	NONE
FACILITY	BPX.WLMSERVER	NONE
FACILITY	IEAABD.DMPAKEY	NONE
FACILITY	IEAABD.DMPAUTH	NONE
OPERCMDS	MVS.SLIP	READ
UNIXPRIV	SUPERUSER.PROCESS.GETPSENT	NONE
UNIXPRIV	SUPERUSER.PROCESS.KILL	NONE
UNIXPRIV	SUPERUSER.PROCESS.PTRACE	NONE

The resources above are all “discrete resources”, that is, the resource name is a predictable value. There are other “sensitive” resource names which contain a variable value, often in the form of a data set name or module name. This support enhances the RACF_SENSITIVE_RESOURCES check to examine these resources for a proper baseline protection. These are shown in “Additional “Generic” General Resources for RACF_SENSITIVE_RESOURCES.”

Table 50. Additional “Generic” General Resources for RACF_SENSITIVE_RESOURCES

Class	Resource	Maximum Public Access
FACILITY	CSVAPF.data_set_name (excluding CSVAPF.MVS.SETPROG.FORMAT. DYNAMIC)	READ
FACILITY	CSVDYLPA.ADD.module_name	READ
FACILITY	CSVDYLPA.DELETE.module_name	READ
FACILITY	CSVDYNEX.exit_name.function. modname (excluding CSVDYNEX.LIST, CSVDYNEX.exit_name.RECOVER, and CSVDYNEX.exit_name.CALL)	READ
FACILITY	CSVDYNL.lnklstname. function (excluding CSVDYNL.lnklstname.DEFINE and CSVDYNL.lnklstname.UNDEFINE)	READ

The RACF_SENSITIVE_RESOURCE health check will report on each resource name that it finds, flagging exceptions in a manner consistent with the existing exception flagging. No new messages are planned.

The RACF_SENSITIVE_RESOURCE check will not validate any portion of the "variable" portion of the resource name.

The RACF_SENSITIVE_RESOURCE check will evaluate only the names which begin with the specific high level qualifier. Profiles which contain generic qualifiers or RACF variables in the high level qualifier will not be flagged.

Because of the change in the resources that are checked in the RACF_SENSITIVE_RESOURCES check, the date associated with the check is changed to '20120106' (6 January 2012).

Reason for check:
The system is critically exposed if these resources are not properly protected.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
Yes, you can specify a user ID as a parameter. The following example shows keywords you can use to specify an user ID (GENUSER) in the PARM field for RACF_SENSITIVE_RESOURCES. You can specify the following keywords on either HZSPRMxx or on a MODIFY command:

```
CHECK(RACF_SENSITIVE_RESOURCES)
OWNER(IBMRACF)
DATE('date_of_the_change')
PARM(GENUSER)
REASON('Your reason for making the update.')
```

The check verifies that the specified user ID is a syntactically valid user ID, that the user ID exists, and that the user ID is active and has not been revoked. If any of these conditions is not true, an error message is written to the IBM Health Checker for z/OS log and the check continues processing as if no parameter had been specified to the check.

User override of IBM values:
The following shows keywords you can use to override RACF check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES),
SEVERITY(HI),INTERVAL(08:00),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides additional error detail in debug mode. You can put a check into debug mode using any of the following:

- UPDATE,filters,DEBUG=ON parameters on either the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member
- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF.

Verbose support:
No.

Reference:
For more information on storage increments, see *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF Auditor's Guide*.

Messages:
This check issues the following exception messages:

- IRRH204E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The report that RACF_GRS_RNL produces is shown below. The columns in this report are as follows:

S Status. An E in this column indicates that the check found an exception and that there is excessive access authority allowed to the data set. A V in this column indicates that the data set is not on the volume. A U in this column indicates that the check did not complete because the dataset was in use by another user.

Data set name
The name of the data set

Vol
The volume upon which the data set resides

UACC
The UACC of the profile that covers the data set

WARN
The WARNING attribute of the profile that covers the data set

ID(*)
The access level assigned to the * user ID on the access list

User
If the installation specified a user ID in the PARMLIB entry for the RACF_SENSITIVE_RESOURCES check PARMLIB, the **User** column contains the string *>xxxx*, where *xxxx* is either Read or None.

RACF_SENSITIVE_RESOURCES report with exceptions, without a user ID:

```
1CHECK(IBMRACF,RACF_SENSITIVE_RESOURCES)
 SYSPLEX:    LOCAL     SYSTEM: RACFR21
 START TIME: 05/24/2013 13:13:46.412092
 CHECK DATE: 20120106  CHECK SEVERITY: HIGH

                          APF Dataset Report

 S Data Set Name                          Vol    UACC Warn ID*  User
 - --------------------------------------- ------ ---- ---- ---- ----
   ASM.SASMMOD1                           ZDR21  Read No   ****
   CBC.SCCNCMP                            ZDR21  Read No   ****
   CBC.SCLBDLL                            ZDR21  Read No   ****
   CBC.SCLBDLL2                           ZDR21  Read No   ****
   CEE.SCEERUN                            ZDR21  Read No   ****
   CEE.SCEERUN2                           ZDR21  Read No   ****
   CSF.SCSFMOD0                           ZDR21  Read No   ****
   EOY.SEOYLOAD                           ZDR21  Read No   ****
   FFST.V120ESA.SEPWMOD1                  ZDR21
   FFST.V120ESA.SEPWMOD2                  ZDR21
   GDDM.SADMMOD                           ZDR21  Read No   ****
   GIM.SGIMLMD0                           ZDR21  Read No   ****
   ISF.SISFLINK                           ZDR21  Read No   ****
   ISF.SISFLOAD                           ZDR21  Read No   ****
   ISP.SISPLOAD                           ZDR21  Read No   ****
   ISP.SISPLPA                            ZDR21  Read No   ****
   RACFDRVR.ATC.AUTHLIB                   D79PK5
   RACF321.MIGLIB                         D97107
   SYS1.CMDLIB                            ZDR21  Read No   ****
   SYS1.DFQLLIB                           ZDR21  Read No   ****
   SYS1.DGTLLIB                           ZDR21  Read No   ****
   SYS1.LINKLIB                           ZDR21  Read No   ****
   SYS1.SBDTLIB                           ZDR21  Read No   ****
```

```
  SYS1.SBDTLINK                           ZDR21  Read No   ****
  SYS1.SCBDHENU                           ZDR21  Read No   ****
  SYS1.SERBLINK                           ZDR21  Read No   ****
  SYS1.SHASLNKE                           ZDR21  Read No   ****
  SYS1.SHASMIG                            ZDR21  Read No   ****
  SYS1.SIATLIB                            ZDR21  Read No   ****
  SYS1.SIATLINK                           ZDR21  Read No   ****
  SYS1.SIATLPA                            ZDR21  Read No   ****
  SYS1.SIATMIG                            ZDR21  Read No   ****
  SYS1.SICELINK                           ZDR21  Read No   ****
  SYS1.SIEALNKE                           ZDR21  Read No   ****
  SYS1.SIOALMOD                           ZDR21  Read No   ****
  SYS1.SISTCLIB                           ZDR21  Read No   ****
  SYS1.SVCLIB                             ZDR21  Read No   ****
  SYS1.VTAMLIB                            ZDR21  Read No   ****
  TCPIP.SEZADSIL                          ZDR21  Read No   ****
  TCPIP.SEZALNK2                          ZDR21  Read No   ****
  TCPIP.SEZALOAD                          ZDR21  Read No   ****
  TCPIP.SEZATCP                           ZDR21  Read No   ****

                        RACF Dataset Report

S Data Set Name                           Vol    UACC Warn ID*  User
- ---------------------------------------- ------ ---- ---- ---- ----
  RACFDRVR.RACF31D                        RDB31D None No   ****

                       PARMLIB Dataset Report

S Data Set Name                           Vol    UACC Warn ID*  User
- ---------------------------------------- ------ ---- ---- ---- ----
  RACFDRVR.PARMLIB.ZR10                   D94RF4 Read No   ****
  RACFDRVR.PARMLIB.ZR11                   D94RF4 Read No   ****
  RACFDRVR.PARMLIB.ZR12                   D94RF4 Read No   ****
  RACFDRVR.PARMLIB.ZR13                   D94RF4 Read No   ****
  RACFDRVR.PARMLIB.Z21                    D94RF4 Read No   ****
  SYS1.PARMLIB                            ZDR21  Read No   ****
  SYS1.PARMLIB.INSTALL                    ZDR21
  SYS1.PARMLIB.POK                        ZDR21

                  Current Link List Dataset Report

S Data Set Name                           Vol    UACC Warn ID*  User
- ---------------------------------------- ------ ---- ---- ---- ----
  ASM.SASMMOD1                            ZDR21  Read No   ****
  CBC.SCLBDLL                             ZDR21  Read No   ****
  CEE.SCEERUN                             ZDR21  Read No   ****
  CEE.SCEERUN2                            ZDR21  Read No   ****
  COMMON.LOOKFEEL.LINKLIB                 ZDR21
  CSF.SCSFMOD0                            ZDR21  Read No   ****
  EOY.SEOYLOAD                            ZDR21  Read No   ****
  GDDM.SADMMOD                            ZDR21  Read No   ****
  GIM.SGIMLMD0                            ZDR21  Read No   ****
  ISF.SISFLINK                            ZDR21  Read No   ****
  ISF.SISFLOAD                            ZDR21  Read No   ****
  ISF.SISFMIG                             ZDR21  Read No   ****
  ISF.SISFMOD1                            ZDR21  Read No   ****
  ISP.SISPLOAD                            ZDR21  Read No   ****
  RACF321.MIGLIB                          D97107
  RACF321.SIEALNKE                        D97107
  SYS1.CMDLIB                             ZDR21  Read No   ****
  SYS1.CSSLIB                             ZDR21  Read No   ****
  SYS1.DFQLLIB                            ZDR21  Read No   ****
  SYS1.DGTLLIB                            ZDR21  Read No   ****
  SYS1.LINKLIB                            ZDR21  Read No   ****
  SYS1.MIGLIB                             ZDR21  Read No   ****
  SYS1.SERBLINK                           ZDR21  Read No   ****
  SYS1.SHASLNKE                           ZDR21  Read No   ****
  SYS1.SHASMIG                            ZDR21  Read No   ****
  SYS1.SIATLIB                            ZDR21  Read No   ****
  SYS1.SIATLINK                           ZDR21  Read No   ****
  SYS1.SIATLPA                            ZDR21  Read No   ****
  SYS1.SIATMIG                            ZDR21  Read No   ****
  SYS1.SICELINK                           ZDR21  Read No   ****
  SYS1.SIEALNKE                           ZDR21  Read No   ****
  SYS1.SIEAMIGE                           ZDR21  Read No   ****
  SYS1.SIOALMOD                           ZDR21  Read No   ****
  SYS1.SORTLIB                            ZDR21  Read No   ****
  SYS1.VTAMLIB                            ZDR21  Read No   ****
  SYS2.CSSLIB                             ZDR21
  SYS2.LINKLIB                            ZDR21
```

```
  SYS2.MIGLIB                              ZDR21
  SYS2.SIEALNKE                            ZDR21
  SYS2.SIEAMIGE                            ZDR21
  TCPIP.SEZALOAD                           ZDR21  Read No   ****

                      System Rexx Dataset Report

S Data Set Name                            Vol    UACC Warn ID*  User
- ----------------------------------------- ------ ---- ---- ---- ----
  SYS1.SAXREXEC                            ZDR21  Read No   ****

                           ICSF Dataset Report

S Data Set Name                            Vol    UACC Warn ID*  User
- ----------------------------------------- ------ ---- ---- ---- ----
  SYSTEMA.PKDS
  SYSTEMA.CKDS

                   Sensitive General Resources Report

S Resource Name                            Class    UACC Warn ID*  User
- ----------------------------------------- -------- ---- ---- ---- ----
  BPX.DAEMON                               FACILITY None No   ****
  BPX.DEBUG                                FACILITY None No   ****
  BPX.FILEATTR.APF                         FACILITY None No   ****
  BPX.FILEATTR.PROGCTL                     FACILITY None No   ****
  BPX.SERVER                               FACILITY None No   ****
  BPX.SUPERUSER                            FACILITY None No   ****
  BPX.WLMSERVER                            FACILITY None No   ****
  CSVAPF.RACFDEV.**.LOAD                   FACILITY Read No   ****
  CSVDYLPA.ADD.MODULE01                    FACILITY Read No   ****
  CSVDYLPA.DELETE.MODULE01                 FACILITY Read No   ****
  CSVDYLPA.ADD.*                           FACILITY Read No   ****
  CSVDYLPA.DELETE.*                        FACILITY Read No   ****
  CSVDYNEX.EXITNAME_READ.MODNAME01         FACILITY Read No   ****
  CSVDYNEX.*.DEFINE                        FACILITY Read No   ****
  CSVDYNEX.*                               FACILITY Read No   ****
  CSVDYNL.ADD                              FACILITY None No   ****
  CSVDYNL.LINKLIST01.ADD                   FACILITY None No   ****
  CSVDYNL.LINKLIST01.DELETE                FACILITY None No   ****
  CSVDYNL.*.ADD                            FACILITY None No   ****
  CSVDYNL.*.DELETE                         FACILITY None No   ****
  IEAABD.DMPAKEY                           FACILITY None No   ****
  IEAABD.DMPAUTH                           FACILITY None No   ****
  ICHBLP                                   FACILITY None No   ****
  IRR.PASSWORD.RESET                       FACILITY None No   ****
  MVS.SET.PROG                             OPERCMDS Read No   ****
  MVS.SETPROG                              OPERCMDS Read No   ****
  MVS.SLIP                                 OPERCMDS Read No   ****
  ACCT                                     TSOAUTH  None No   ****
  CONSOLE                                  TSOAUTH  None No   ****
  OPER                                     TSOAUTH  None No   ****
  PARMLIB                                  TSOAUTH  None No   ****
  TESTAUTH                                 TSOAUTH  None No   ****
  SUPERUSER.FILESYS                        UNIXPRIV
  SUPERUSER.FILESYS.CHANGEPERMS            UNIXPRIV
  SUPERUSER.FILESYS.CHOWN                  UNIXPRIV
  SUPERUSER.FILESYS.MOUNT                  UNIXPRIV
  SUPERUSER.PROCESS.GETPSENT               UNIXPRIV
  SUPERUSER.PROCESS.KILL                   UNIXPRIV
  SUPERUSER.PROCESS.PTRACE                 UNIXPRIV


                             ICHAUTAB Report

S Module   REQUEST= REQUEST= Location
           VERIFY   LIST
- -------- -------- -------- --------

IRRH239I There are no ICHAUTAB programs on this system.

IRRH205I The RACF_SENSITIVE_RESOURCES check has not found any errors in
the security controls on this system.

END TIME: 05/24/2013 13:13:48.281732  STATUS: SUCCESSFUL
```

RACF_SENSITIVE_RESOURCES report with exceptions, with a user ID:

```
                          RACF Dataset Report

 S Data Set Name                          Vol    UACC Warn ID*  User
 - ---------------------------------------- ------ ---- ---- ---- ----
 E RACFDRVR.RACF317                       RDB317 None No   **** >None
* High severity Exception *
```

RACF_SENSITIVE_RESOURCES report without exceptions: Note that no user ID was specified for this report.

```
START TIME: 11/18/2004 16:54:09.533912  IBMRACF,
RACF_SENSITIVE_RESOURCES
 OWNER DATE:

                           APF Dataset Report


 S Data Set Name                          Vol    UACC Warn ID*  User
 - ---------------------------------------- ------ ---- ---- ---- ----
   MVSSTORE.SRVLIB.ZOS15.NUCLEUS          DRVPSL None No   ****
   SYS1.LINKLIB                           ZDR17B Read No   ****
   SYS1.NFSLIB                            ZDR17B Read No   ****
   SYS1.SIATLPA                           ZDR17B Read No   ****
   SYS1.SVCLIB                            ZDR17B **** **** ****


                          RACF Dataset Report


 S Data Set Name                          Vol    UACC Warn ID*  User
 - ---------------------------------------- ------ ---- ---- ---- ----
   RACFDRVR.RACF317                       RDB317 None No   ****

IRRH205I   The RACF check RACF_SENSITIVE_RESOURCES has not found
    any errors in the security controls on this system.
```

RACF_*classname*_ACTIVE

Description: Each of the RACF_*classname*_ACTIVE checks examine the status of a single RACF general resource class:

- RACF_UNIXPRIV_ACTIVE
- RACF_FACILITY_ACTIVE
- RACF_TAPEVOL_ACTIVE
- RACF_TEMPDSN_ACTIVE
- RACF_TSOAUTH_ACTIVE
- RACF_OPERCMDS_ACTIVE
- RACF_CSFKEYS_ACTIVE
- RACF_CSFSERV_ACTIVE

Reason for check:
An effective RACF implementation requires that the baseline group of RACF general resource classes listed above be active.

z/OS releases the check applies to:
z/OS V1R5 and later.

Parameters accepted:
No.

User override of IBM values:
The following shows keywords you can use to override RACF check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,RACF_classname_ACTIVE),
SEVERITY(MED),INTERVAL(24:00),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides additional error detail in debug mode. You can put a check into debug mode using any of the following:

- UPDATE,filters,DEBUG=ON parameters on either the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member
- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF.

Verbose support:
No.

Reference:
For more information on storage increments, see *z/OS Security Server RACF Security Administrator's Guide* .

Messages:
This check issues the following exception messages:

- IRRH229E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

RACF_*FACILITY*__ACTIVE check - no exception found:

```
1CHECK(IBMRACF,RACF_FACILITY_ACTIVE)
 START TIME: 03/02/2006 14:50:57.305795
 CHECK DATE: 20051111  CHECK SEVERITY: MEDIUM
 CHECK PARM: FACILITY


 IRRH228I The class FACILITY is active.

 END TIME: 03/02/2006 14:50:57.314865  STATUS: SUCCESSFUL
```

RACF_*TAPEVOL*_ACTIVE check - class inactive exception found:

```
1CHECK(IBMRACF,RACF_TAPEVOL_ACTIVE)
 START TIME: 03/02/2006 14:50:57.304859
 CHECK DATE: 20051111  CHECK SEVERITY: MEDIUM
 CHECK PARM: TAPEVOL


 * Medium Severity Exception *

 IRRH229E The class TAPEVOL is not active.

   Explanation:  The class is not active. IBM recommends that the
  security administrator evaluate the need for this class, define
  profiles in it as appropriate, and activate the class.

   System Action:  The check continues processing. There is no effect on
     the system.

   Operator Response:  Report this problem to the system security
     administrator and the system auditor.
```

```
   System Programmer Response:  None.

   Problem Determination:  See the RACF Security Administrator's Guide,
     the RACF Auditor's Guide and the RACF System Programmer's Guide.

   Source:
     RACF Security Administrator's Guide
     RACF Auditor's Guide
     RACF System Programmer's Guide

   Reference Documentation:
     RACF Security Administrator's Guide
     RACF Auditor's Guide
     RACF System Programmer's Guide

   Automation:  None.

   Check Reason:  IBM recommends activating this class

 END TIME: 03/02/2006 14:50:57.314816  STATUS: EXCEPTION-MED
```

RACF_*class_name*_ACTIVE check - no exceptions found (the class is active):

```
CHECK(IBMRACF,RACF_TSOAUTH_ACTIVE)
START TIME: 11/16/2005 13:17:30.931923
CHECK DATE: 20050820 CHECK SEVERITY: MEDIUM
CHECK PARM: TSOAUTH
IRRH228I The class TSOAUTH is active.
END TIME: 11/16/2005 13:17:30.945682 STATUS: SUCCESSFUL
```

RACF_IBMUSER_REVOKED

Description: Check looks to see if the IBMUSER user ID is still active.

Reason for check:
The IBMUSER user ID is intended for use only during the initial installation process. After installation, the IBMUSER user ID should be revoked so that it cannot be used by unauthorized users.

z/OS releases the check applies to:
z/OS V1R5 and later.

Parameters accepted:
No.

User override of IBM values:
The following shows keywords you can use to override RACF check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,RACF_IBMUSER_REVOKED),
SEVERITY(MED),INTERVAL(24:00),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
Yes, the check provides additional error detail in debug mode. You can put a check into debug mode using any of the following:

- UPDATE,filters,DEBUG=ON parameters on either the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member
- Overwrite the OFF value with the ON value in the DEBUG column of the CK panel in SDSF.

Verbose support:
No.

Reference:
For more information on storage increments, see *z/OS Security Server RACF Security Administrator's Guide* .

Messages:
This check issues the following exception messages:

- IRRH225E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

RACF_IBMUSER_REVOKED check - IBMUSER not revoked exception found:

```
CHECK(IBMRACF,RACF_IBMUSER_REVOKED)
START TIME: 12/02/2005 16:43:31.614417
CHECK DATE: 20050820 CHECK SEVERITY: MEDIUM
* Medium Severity Exception *
IRRH225E The user ID IBMUSER is not revoked.
Explanation: The user ID IBMUSER has not been revoked. IBM recommends
revoking IBMUSER.
System Action: The check continues processing. There is no effect on
the system.
Operator Response: Report this problem to the system security
administrator and the system auditor.
System Programmer Response: Revoke IBMUSER.
Problem Determination: See the RACF Auditor's Guide and the RACF
System Programmer's Guide.
Source:
RACF System Programmer's Guide
RACF Auditor's Guide
Reference Documentation:
RACF System Programmer's Guide
RACF Auditor's Guide
Automation: None.
Check Reason: IBMUSER should be revoked.
END TIME: 12/02/2005 16:43:31.653215 STATUS: EXCEPTION-MED
```

RACF_IBMUSER_REVOKED check - no exceptions found, IBMUSER has been revoked:

```
1CHECK(IBMRACF,RACF_IBMUSER_REVOKED)
 START TIME: 03/02/2006 14:50:57.307193
 CHECK DATE: 20051111  CHECK SEVERITY: MEDIUM

 IRRH224I The user ID IBMUSER is revoked.

 END TIME: 03/02/2006 14:50:57.315063  STATUS: SUCCESSFUL
```

RACF_UNIX_ID

Description:
z/OS V1R13 is the last release that supports default UNIX identities implemented using the BPX.DEFAULT.USER profile in the FACILITY class. To replace this function you can do one of the following:

- Use the replacement BPX.UNIQUE.USER profile function provided in z/OS R11 to enable RACF to automatically generate unique UIDs and GIDs.
- Define OMVS segments for all users and groups who require UNIX services.

The RACF_UNIX_ID check detects whether RACF is enabled to perform the best practice of automatically assigning unique UNIX identities when users

without OMVS segments access the system to use UNIX services. This determination is based on whether the BPX.UNIQUE.USER and BPX.DEFAULT.USER profiles are defined in the FACILITY class. The following table summarizes the actions of the check:

Table 51. RACF_UNIX_ID check actions based on whether the BPX.UNIQUE.USER and BPX.DEFAULT.USER profiles are defined in the FACILITY class

BPX.UNIQUE.USER defined in Facility	BPX.DEFAULT.USER defined in Facility	Check action
No	No	RACF is not enabled to assign z/OS UNIX identities to users or groups who do not have OMVS segments. The check issues informational message IRRH504I (see "“RACF_UNIX_ID” on page 491") and does not raise an exception, but you should use the best practice of assigning a unique UID and a unique GID to each user and group which needs access to z/OS UNIX functions and resources using either the BPX.UNIQUE.USER profile or by defining OMVS segments manually.
No	Yes	The presence of the BPX.DEFAULT.USER profile without the BPX.UNIQUE.USER profile indicates an intent to use default OMVS segment support, which is not recommended. The check raises a medium severity exception and issues error message IRRH505E. See "“RACF_UNIX_ID” on page 491".
Yes	Yes or No	The presence of the BPX.UNIQUE.USER profile (with or without BPX.DEFAULT.USER) indicates an intent to have RACF automatically generate unique UNIX UIDs and GIDs, as is recommended. The check issues informational message IRRH502I and then verifies requirements for the automatic generation of unique UNIX IDs. IRRH502I includes a report showing whether all requirements have been met. See a sample of IRRH502I in "“RACF_UNIX_ID” on page 491" The check's action then depends on whether it finds that requirements have been met or not: • If all requirements have been met, the check raises no exceptions and issues informational message IRRH506I. • If the check detects that not all requirements have been met, the check raises a medium severity exception and issues error message IRRH503E. Note that if both the BPX.UNIQUE.USER and BPX.DEFAULT.USER profiles are defined, RACF automatically assigns unique UNIX IDs. In this case, RACF does not use the BPX.DEFAULT.USER profile and, therefore does not do default OMVS segment processing.

Reason for check:
IBM recommends that a unique UNIX UID be assigned to each user and that a unique GID be assigned to each group that needs access to z/OS UNIX functions and resources. Assigning unique identities, rather than shared identities, improves overall security and increases user accountability.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
No

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,RACF_UNIX_ID)
SEVERITY(MED),INTERVAL(24:00),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
z/OS Security Server RACF Security Administrator's Guide

Messages:
This check issues the following exception messages:

- IRRH503E
- IRRH505E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

- The following shows the output from a RACF_UNIX_ID check that finds neither the BPX.UNIQUE.USER or BPX.DEFAULT.USER profiles are defined:

```
CHECK(IBMRACF,RACF_UNIX_ID)
START TIME: 05/11/2011 10:26:01.195890
CHECK DATE: 20110101  CHECK SEVERITY: MEDIUM

IRRH504I RACF is not enabled to assign UNIX IDs when users or groups
that do not have OMVS segments use certain z/OS UNIX services. If you
choose not to define UNIX IDs for each user of UNIX functions, you can
enable RACF to automatically generate unique UNIX UIDs and GIDs for you.

END TIME: 05/11/2011 10:26:01.201875  STATUS: SUCCESSFUL
```

- The following shows the output from an exception for RACF_UNIX_ID when the presence of the BPX.DEFAULT.USER profile without the BPX.UNIQUE.USER profile indicates an intent to use default OMVS segment support, which is not recommended:

```
CHECK(IBMRACF,RACF_UNIX_ID)
START TIME: 05/10/2011 16:02:41.125401
CHECK DATE: 20110101  CHECK SEVERITY: MEDIUM

* Medium Severity Exception *

IRRH505E The BPX.DEFAULT.USER profile in the FACILITY class
indicates that you want RACF to assign shared default UNIX
IDs when users or groups that do not have OMVS segments use
certain z/OS UNIX services.

  Explanation:  The RACF UNIX identity check has found the
    BPX.DEFAULT.USER profile in the FACILITY class. The presence of this
    profile indicates an intent to have RACF assign shared default UNIX
    UIDs and GIDs when users without OMVS segments access the system to
    use certain UNIX services.

    On z/OS V1R13 and below, you have the option of enabling RACF to
    assign default z/OS UNIX identities, however it is not suggested.
    You should either define OMVS segments for user and group profiles,
    with unique UIDs and GIDs, or you should enable RACF to
    automatically assign unique z/OS UNIX identities when users without
    OMVS segments access the system to use certain UNIX services.
    Assigning unique identities rather than shared identities improves
    overall security and increases user accountability.
```

```
    See z/OS Security Server RACF Security Administrator's Guide for
    more information about how to assign a user identifier (UID) to a
    RACF user and how to assign a group identifier (GID) to a RACF
    group. z/OS Security Server RACF Security Administrator's Guide also
    contains information about how to enable RACF to automatically
    assign unique UNIX identities.

    Note: z/OS V1R13 is the last release that supports default UNIX
    identities. After z/OS V1R13, users and groups that need to access
    z/OS UNIX functions and resources must be assigned unique UNIX UIDs
    and unique GIDs in advance of their need to access these services,
    or you must enable RACF to automatically assign unique z/OS UNIX
    identities when users without OMVS segments access the system to use
    certain UNIX services. The FACILITY class BPX.DEFAULT.USER profile
    will no longer be used and can be deleted.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator.

  System Programmer Response:  None.

  Problem Determination:

  Source:

  Reference Documentation:
    z/OS Security Server RACF Security Administrator's Guide

  Automation:  None.

  Check Reason:  Unique UNIX identities are recommended.

END TIME: 05/10/2011 16:02:41.126280  STATUS: EXCEPTION-MED
```

- The following shows the output from a RACF_UNIX_ID check that finds that the requirements for the automatic generation of unique UNIX IDs have been met:

```
CHECK(IBMRACF,RACF_UNIX_ID)
START TIME: 05/11/2011 09:54:50.971115
CHECK DATE: 20110101  CHECK SEVERITY: MEDIUM

IRRH502I RACF attempts to assign unique UNIX IDs when users or groups
that do not have OMVS segments use certain z/OS UNIX services.

Requirements for this support:

S Requirement
- -------------------------------------------------------------------
  FACILITY class profile BPX.UNIQUE.USER is defined
  RACF database is at the required AIM stage:
    AIM stage = 03
  UNIXPRIV class profile SHARED.IDS is defined
  UNIXPRIV class is active
  UNIXPRIV class is RACLISTed
  FACILITY class profile BPX.NEXT.USER is defined
  BPX.NEXT.USER profile APPLDATA is specified (not verified):
    APPLDATA = 1/0

IRRH506I The RACF UNIX identity check has detected no exceptions.

END TIME: 05/11/2011 09:54:50.972634  STATUS: SUCCESSFUL
```

- The following shows the output from a RACF_UNIX_ID check that finds that the requirements for the automatic generation of unique UNIX IDs have NOT been met and raises an exception:

```
CHECK(IBMRACF,RACF_UNIX_ID)
START TIME: 05/11/2011 09:44:58.682612
CHECK DATE: 20110101  CHECK SEVERITY: MEDIUM

IRRH502I RACF attempts to assign unique UNIX IDs when users or groups
that do not have OMVS segments use certain z/OS UNIX services.

Requirements for this support:

S Requirement
- ------------------------------------------------------------------
  FACILITY class profile BPX.UNIQUE.USER is defined
  RACF database is at the required AIM stage:
    AIM stage = 03
E UNIXPRIV class profile SHARED.IDS is not defined
E UNIXPRIV class is not active
E UNIXPRIV class is not RACLISTed
E FACILITY class profile BPX.NEXT.USER is not defined

* Medium Severity Exception *

IRRH503E RACF cannot assign unique UNIX IDs when users or groups that
do not have OMVS segments use certain z/OS UNIX services. One or more
requirements are not satisfied.

  Explanation:  The RACF UNIX identity check has determined that you
    want RACF to assign unique UNIX IDs when users or groups without
    OMVS segments use certain z/OS UNIX services. However, RACF is not
    able to assign unique UNIX identities for z/OS UNIX services because
    one or more of the following requirements are not satisfied:

    1. The RACF database is enabled for application identity mapping
    (AIM) stage 3.

    2. The UNIXPRIV class profile SHARED.IDS is defined and the UNIXPRIV
    class is active and RACLISTed.

    3. The FACILITY class profile BPX.NEXT.USER is defined and its
    APPLDATA field has valid ID values or ranges.

    4. The FACILITY class profile BPX.UNIQUE.USER is defined.

    See z/OS Security Server RACF Security Administrator's Guide for
    more information about enabling RACF for automatic assignment of
    unique UNIX identities.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator.

  System Programmer Response:  None.

  Problem Determination:  The check produces a report listing the
    requirements. An "E" in the "S" (Status) column indicates that a
    requirement is not satisfied. For example, if the RACF database has
    not been enabled for AIM stage 3, this requirement is flagged as an
    exception. If the "S" field is blank, the requirement is satisfied.
    One or more requirements are not satisfied and have been flagged as
    an exception in the Status column.

  Source:
```

```
                        Reference Documentation:
                          z/OS Security Server RACF Security Administrator's Guide

                        Automation:  None.

                        Check Reason:  Unique UNIX identities are recommended.

                      END TIME: 05/11/2011 09:44:58.740914  STATUS: EXCEPTION-MED
```

ZOSMIGV1R13_DEFAULT_UNIX_ID

Description:

This check determines whether a client is relying on RACF to assign default z/OS UNIX identities for users without OMVS segments who are accessing UNIX services. IBM recommends that a unique UNIX UID be assigned to each user and that a unique GID be assigned to each group that needs access to z/OS UNIX functions and resources.

Starting with z/OS V1R13, support for the default UNIX identity, implemented using the BPX.DEFAULT.USER profile in the FACILITY class, is no longer available, so a migration action may be required if you are using it. The need for a migration action is based on whether the BPX.UNIQUE.USER and BPX.DEFAULT.USER profiles are defined in the FACILITY class. The following table summarizes:

Table 52. ZOSMIGV1R13_DEFAULT_UNIX_ID check actions and migration actions

BPX.UNIQUE.USER defined in Facility	BPX.DEFAULT.USER defined in Facility	Check action and migration action required:
No	No	RACF is not enabled to assign z/OS UNIX identities to users or groups who do not have OMVS segments. The check issues informational message IRRH504I (see ""ZOSMIGV1R13_DEFAULT_UNIX_ID"") and does not raise an exception, but you should use the best practice of assigning a unique UID and a unique GID to each user and group which needs access to z/OS UNIX functions and resources using either the BPX.UNIQUE.USER profile or by defining OMVS segments manually. **Migration action**: Not required; the installation continues to perform as before.
No	Yes	The presence of the BPX.DEFAULT.USER profile without the BPX.UNIQUE.USER profile indicates an intent to use default OMVS segment support, which is not recommended. The check raises a low severity exception and issues error message IRRH505E. See ""ZOSMIGV1R13_DEFAULT_UNIX_ID"". **Migration action**: Required, because default OMVS segment support is not supported in z/OS V1R13 or later. Do one of the following: • Use the replacement BPX.UNIQUE.USER profile function provided in z/OS R11 to enable RACF to automatically generate unique UIDs and GIDs. • Define OMVS segments for all users and groups who require UNIX services.

Table 52. ZOSMIGV1R13_DEFAULT_UNIX_ID check actions and migration actions (continued)

BPX.UNIQUE.USER defined in Facility	BPX.DEFAULT.USER defined in Facility	Check action and migration action required:
Yes	Yes or No	The presence of the BPX.UNIQUE.USER profile (with or without BPX.DEFAULT.USER) indicates an intent to have RACF automatically generate unique UNIX UIDs and GIDs, as is recommended. The check issues informational message IRRH502I and then verifies requirements for the automatic generation of unique UNIX IDs. IRRH502I includes a report showing whether all requirements have been met. See a sample of IRRH502I in "“ZOSMIGV1R13_DEFAULT_UNIX_ID” on page 496" The check's action then depends on whether it finds that requirements have been met or not: • If all requirements have been met, the check raises no exceptions and issues informational message IRRH506I. • If the check detects that not all requirements have been met, it issues informational message IRRH507I and does not raise an exception **Migration action**: Not required - requirements for the automatic generation of unique UNIX IDs are an issue of enablement rather than migration.

Reason for check:
Starting with z/OS V1R13, support for the default UNIX identity, implemented using the BPX.DEFAULT.USER profile in the FACILITY class, is no longer available, so a migration action may be required if you are using it

z/OS releases the check applies to:
z/OS V1R12 and z/OS V1R13

Parameters accepted:
No

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command:

```
UPDATE,
CHECK(IBMRACF,ZOSMIGV1R13_DEFAULT_UNIX_ID)
SEVERITY(LOW),INTERVAL(ONETIME),DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
z/OS Security Server RACF Security Administrator's Guide

Messages:
This check issues the following exception messages:

- IRRH505E

See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

- The following shows the output from a ZOSMIGV1R13_DEFAULT_UNIX_ID check that finds neither the BPX.UNIQUE.USER or BPX.DEFAULT.USER profiles are defined:

```
CHECK(IBMRACF,ZOSMIGV1R13_DEFAULT_UNIX_ID)
START TIME: 05/11/2011 10:34:11.210824
CHECK DATE: 20110101  CHECK SEVERITY: LOW

IRRH504I RACF is not enabled to assign UNIX IDs when users or groups
that do not have OMVS segments use certain z/OS UNIX services. If you
choose not to define UNIX IDs for each user of UNIX functions, you can
enable RACF to automatically generate unique UNIX UIDs and GIDs for you.

END TIME: 05/11/2011 10:34:11.211004  STATUS: SUCCESSFUL
```

- The following shows the output from an exception for ZOSMIGV1R13_DEFAULT_UNIX_ID when the presence of the BPX.DEFAULT.USER profile without the BPX.UNIQUE.USER profile indicates an intent to use default OMVS segment support, which is not recommended:

```
CHECK(IBMRACF,ZOSMIGV1R13_DEFAULT_UNIX_ID)
START TIME: 05/11/2011 10:36:31.611960
CHECK DATE: 20110101  CHECK SEVERITY: LOW

* Low Severity Exception *

IRRH505E The BPX.DEFAULT.USER profile in the FACILITY class
indicates that you want RACF to assign shared default UNIX
IDs when users or groups that do not have OMVS segments use
certain z/OS UNIX services.

  Explanation:  The RACF UNIX identity check has found the
    BPX.DEFAULT.USER profile in the FACILITY class. The presence of this
    profile indicates an intent to have RACF assign shared default UNIX
    UIDs and GIDs when users without OMVS segments access the system to
    use certain UNIX services.

    On z/OS V1R13 and below, you have the option of enabling RACF to
    assign default z/OS UNIX identities, however it is not suggested.
    You should either define OMVS segments for user and group profiles,
    with unique UIDs and GIDs, or you should enable RACF to
    automatically assign unique z/OS UNIX identities when users without
    OMVS segments access the system to use certain UNIX services.
    Assigning unique identities rather than shared identities improves
    overall security and increases user accountability.

    See z/OS Security Server RACF Security Administrator's Guide for
    more information about how to assign a user identifier (UID) to a
    RACF user and how to assign a group identifier (GID) to a RACF
    group. z/OS Security Server RACF Security Administrator's Guide also
    contains information about how to enable RACF to automatically
    assign unique UNIX identities.

    Note: z/OS V1R13 is the last release that supports default UNIX
    identities. After z/OS V1R13, users and groups that need to access
    z/OS UNIX functions and resources must be assigned unique UNIX UIDs
    and unique GIDs in advance of their need to access these services,
    or you must enable RACF to automatically assign unique z/OS UNIX
    identities when users without OMVS segments access the system to use
    certain UNIX services. The FACILITY class BPX.DEFAULT.USER profile
    will no longer be used and can be deleted.

  System Action:  The check continues processing. There is no effect on
    the system.

  Operator Response:  Report this problem to the system security
    administrator.
```

```
  System Programmer Response:  None.

  Problem Determination:

  Source:

  Reference Documentation:
    z/OS Security Server RACF Security Administrator's Guide

  Automation:  None.

  Check Reason:  Migration check for BPX.DEFAULT.USER removal.

END TIME: 05/11/2011 10:36:31.612823  STATUS: EXCEPTION-LOW
```

- The following shows the output from a ZOSMIGV1R13_DEFAULT_UNIX_ID check that finds that the requirements for the automatic generation of unique UNIX IDs have been met:

```
CHECK(IBMRACF,ZOSMIGV1R13_DEFAULT_UNIX_ID)
START TIME: 05/11/2011 11:02:39.632614
CHECK DATE: 20110101  CHECK SEVERITY: LOW

IRRH502I RACF attempts to assign unique UNIX IDs when users or groups
that do not have OMVS segments use certain z/OS UNIX services.

Requirements for this support:

S Requirement
- -------------------------------------------------------------------
  FACILITY class profile BPX.UNIQUE.USER is defined
  RACF database is at the required AIM stage:
    AIM stage = 03
  UNIXPRIV class profile SHARED.IDS is defined
  UNIXPRIV class is active
  UNIXPRIV class is RACLISTed
  FACILITY class profile BPX.NEXT.USER is defined
  BPX.NEXT.USER profile APPLDATA is specified (not verified):
    APPLDATA = 1/0

IRRH506I The RACF UNIX identity check has detected no exceptions.

END TIME: 05/11/2011 11:02:39.634310  STATUS: SUCCESSFUL
```

- The following shows the output from a ZOSMIGV1R13_DEFAULT_UNIX_ID check that finds that the requirements for the automatic generation of unique UNIX IDs have NOT been met and raises an exception:

```
CHECK(IBMRACF,ZOSMIGV1R13_DEFAULT_UNIX_ID)
START TIME: 05/11/2011 11:05:26.315471
CHECK DATE: 20110101  CHECK SEVERITY: LOW

IRRH502I RACF attempts to assign unique UNIX IDs when users or groups
that do not have OMVS segments use certain z/OS UNIX services.

Requirements for this support:

S Requirement
- -------------------------------------------------------------------
  FACILITY class profile BPX.UNIQUE.USER is defined
  RACF database is at the required AIM stage:
    AIM stage = 03
  UNIXPRIV class profile SHARED.IDS is defined
E UNIXPRIV class is not active
  UNIXPRIV class is RACLISTed
  FACILITY class profile BPX.NEXT.USER is defined
  BPX.NEXT.USER profile APPLDATA is specified (not verified):
    APPLDATA = 1/0
```

```
IRRH507I RACF cannot assign unique UNIX IDs when users or groups that do
not have OMVS segments use certain z/OS UNIX services. One or more
requirements are not satisfied.

END TIME: 05/11/2011 11:05:26.317215  STATUS: SUCCESSFUL
```

Reconfiguration checks (IBMRCF)

RCF_PCCA_ABOVE_16M

Description:
Checks to see whether the residency mode (RMODE), specified for the PCCA control block in the CBLOC parameter of the DIAGxx parmlib member, is the expected value. The default RMODE for the PCCA control block is RMODE 31. The check will look for RMODE 31 for the PCCA control block unless you specify an RMODE of 24 in the RMODE parameter for the check.

Reason for check:
The suggested RMODE for the PCCA control block is RMODE 31.

z/OS releases the check applies to:
z/OS V1R12 and higher.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMRCF,RCF_PCCA_ABOVE_16M),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('CBLOC(31)'),
DATE('date_of_the_change')
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:

- CBLOC(31), which is the default, specifies that you want the check to generate an exception if it finds that IHAPCCA had been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member.
- CBLOC(24) specifies that you want the check to generate an exception if it finds that IHAPCCA either:
  - Had been specified within the CBLOC VIRTUAL31 parameter of the DIAGxx parmlib member.
  - Had not been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member (because CBLOC VIRTUAL31 is the default for the PCCA).

Reference:
See the CBLOC parameter in the DIAGxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IEAVEH101E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R12_RCF_PCCA_ABOVE_16M

Description:
Checks to see whether the residency mode (RMODE), specified for the PCCA control block in the CBLOC parameter of the DIAGxx parmlib member, is the expected value. The default RMODE for the PCCA control block on z/OS systems at the pre-z/OS R12 level is RMODE 24. The check will look for RMODE 24 for the PCCA control block unless you specify an RMODE of 31 in the RMODE parameter for the check.

Reason for check:
The default RMODE for the PCCA control block is changing from RMODE 24 to RMODE 31 in z/OS V1R12.

z/OS releases the check applies to:
z/OS V1R10 and V1R11.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMSUP,ZOSMIGV1R12_RCF_PCCA_ABOVE_16M),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('CBLOC(31)'),
DATE('date_of_the_change')
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:

- CBLOC(31), which is the default, specifies that you want the check to generate an exception if it finds that IHAPCCA had been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member.
- CBLOC(24) specifies that you want the check to generate an exception if it finds that IHAPCCA either:
  - Had been specified within the CBLOC VIRTUAL31 parameter of the DIAGxx parmlib member.

– Had not been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member (because CBLOC VIRTUAL24 is the default for the PCCA).

Reference:
See the CBLOC parameter in the DIAGxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IEAVEH101E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RMM checks (IBMRMM)

ZOSMIGV1R10_RMM_REJECTS_DEFINED

Description:
The purpose of this check is to ensure that an installation knows they are affected by the change in DFSMSrmm default processing when moving from systems below z/OS V1R10.

The RACF userid used for the System REXX check, (REXX checks run under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*.), issues the RMM LISTCONTROL subcommand and so needs one of the following:

- CONTROL access to STGADMIN.EDG.LISTCONTROL in the FACILITY class
- CONTROL access to STGADMIN.EDG.MASTER in the FACILITY class

Reason for check:
The check is intended to identify when a migration action is needed when migrating from a release below z/OS V1R10.

z/OS releases the check applies to:
z/OS V1.8 and z/OS V1.9

Type of check (local, remote, or REXX):
REXX

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMRMM,ZOSMIGV1R10_RMM_REJECTS_DEFINED)
 INTERVAL(ONETIME)
 SEVERITY(MED)
 DATE('date_of_the_change')
 INACTIVE
 REASON('Your reason for making the update.')
```

Debug support:
Yes, additional information is written to REXXOUT data set.

Verbose support:
Yes, when also used with DEBUG, additional information is written to REXXOUT data set.

Parameters accepted:
No

Reference:
For additional information about partitioning and open rules see Using the Parmlib Member EDGRMMxx in *z/OS DFSMSrmm Implementation and Customization Guide*.

Messages:
This check issues the following exception messages:

- EDGH8001E

See the EDGH messages in *z/OS MVS System Messages, Vol 5 (EDG-GFS)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R10_RMM_VOL_REPLACE_LIM

Description:
Use this check to determine whether the hard coded volume replacement limit, LIMIT=1, value has changed and to show how to set the same limit at z/OS V1R10 and later releases using MEDINF NAME(IBM) REPLACE(PERM(value)).

This check uses the AMASPZAP tool to verify contents of a load module in LINKLIB. The RACF userid used for the System REXX check, (REXX checks run under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*), needs READ access to the LINKLIB library, and if AMASPZAP is under program control, the userid also needs READ access to AMASPZAP.

Reason for check:
The check is intended to identify when a migration action is needed when migrating from a release below z/OS V1R10.

z/OS releases the check applies to:
z/OS V1.8 and z/OS V1.9

Type of check (local, remote, or REXX):
REXX

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMRMM,ZOSMIGV1R10_RMM_VOL_REPLACE_LIM)
 INTERVAL(ONETIME)
 SEVERITY(MED)
 DATE('date_of_the_change')
 INACTIVE
 REASON('Your reason for making the update.')
 PARM('LINKLIB(SYS1.LINKLIB)')
```

Debug support:
Yes, additional information is written to REXXOUT data set.

Verbose support:
Yes, when also used with DEBUG, additional information is written to REXXOUT data set.

Parameters accepted:
Optional PARM('LINKLIB(*linklib_dsname*)') is supported. *linklib_dsname* is a fully qualified data set name, specified without quotes, where load modules targeted to the LINKLIB DDDEF name reside. The load module EDGMUPD must be in this data set.

When no PARM value is provided, the check uses SYS1.LINKLIB.

Reference:
For additional information about use of EDGRMMxx parmlib parameter MEDINF with REPLACE, see Defining Media Information: MEDINF in *z/OS DFSMSrmm Implementation and Customization Guide*.

Messages:
This check issues the following exception messages:

- EDGH8002E

See the EDGH messages in *z/OS MVS System Messages, Vol 5 (EDG-GFS)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R10_RMM_VRS_DELETED

Description:
This check ensures that an installation has no VRSes which conflict with the new 'DELETED' VRS support. If your installation has VRSes with either DSNAME('DELETED') or JOBNAME(DELETED) you should be aware of the conflict with new 'DELETED' VRS support when migrating from a release below z/OS V1R10.

The check should be run once for each DFSMSrmm CDS. The RACF userid used for the System REXX check, (REXX checks run under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*), issues the RMM SEARCHVRS subcommand and so needs READ access to STGADMIN.EDG.VRS in the FACILITY class.

Reason for check:
This check identifies when a migration action is needed when migrating from a release below z/OS V1R10.

z/OS releases the check applies to:
z/OS V1.8 and z/OS V1.9

Type of check (local, remote, or REXX):
REXX

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMRMM,ZOSMIGV1R10_RMM_VRS_DELETED)
 INTERVAL(ONETIME)
 SEVERITY(MED)
 DATE('date_of_the_change')
 INACTIVE
 REASON('Your reason for making the update.')
```

Debug support:
Yes, additional information is written to REXXOUT data set.

Verbose support:
Yes, when also used with DEBUG, additional information is written to REXXOUT data set.

Parameters accepted:
Optional PARM('LINKLIB(*linklib_dsname*)') is supported. *linklib_dsname* is a fully qualified data set name, specified without quotes, where load modules targeted to the LINKLIB DDDEF name reside. The load module EDGMUPD must be in this data set.

When no PARM value is provided, the check uses SYS1.LINKLIB.

Reference:
For additional information about deleted tape data set support see Considerations for Retaining Data Sets and Volumes in *z/OS DFSMSrmm Implementation and Customization Guide*.

Messages:
This check issues the following exception messages:

- EDGH8003E

See theEDGH messages in *z/OS MVS System Messages, Vol 5 (EDG-GFS)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R11_RMM_DUPLICATE_GDG

Description:
This check is used to determine if an installation has changed the hard coded switches in EDGVREC load module which control VRSEL GDG processing, and to show them how to enable the same limit at z/OS V1R11 and later releases using OPTION GDG.

This is a System REXX check, which means that it runs under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*. The RACF user ID used for the System REXX check must have READ access to the LINKLIB library, and if AMASPZAP is under program control, the userid also needs READ access to AMASPZAP.

Reason for check:
The check is intended to identify when a migration action is needed when migrating from a release below z/OS V1R11.

z/OS releases the check applies to:
z/OS V1.9 and z/OS V1.10

Type of check (local, remote, or REXX):
REXX

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMRMM,ZOSMIGV1R11_RMM_DUPLICATE_GDG)
 INTERVAL(ONETIME)
 SEVERITY(MED)
 DATE('date_of_the_change')
 INACTIVE
 REASON('Your reason for making the update.')
 PARM('LINKLIB(SYS1.LINKLIB)')
```

Debug support:
Yes, additional information is written to REXXOUT data set.

Verbose support:
Yes, when also used with DEBUG, additional information is written to REXXOUT data set.

Parameters accepted:
An optional PARM('LINKLIB(*linklib_dsname*)') is supported. *linklib_dsname* is a fully qualified data set name, specified without quotes, where load modules targeted to the LINKLIB DDDEF name reside. The load module EDGVREC must be in this data set.

When no PARM is provided the check uses SYS1.LINKLIB.

Reference:
For additional information about use of EDGRMMxx parmlib OPTION GDG see Defining System Options: OPTION in *z/OS DFSMSrmm Implementation and Customization Guide*.

Messages:
This check issues the following exception messages:

- EDGH8004E
- EDGH8005E

See the EDGH messages in *z/OS MVS System Messages, Vol 5 (EDG-GFS)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R11_RMM_REXX_STEM

Description:
This check is used to determine if installation written Rexx EXECs that issued RMM TSO subcommands do either of the following:

- Use stem variables which are removed in z/OS V1R11 and later systems.
- Reference one or more key or special stem .0 variables that are now only created for specific RMM TSO subcommands

The check uses ISRSUPC tool to verify the contents of each EXEC. The RACF userid used for the System Rexx check, (REXX checks run under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*), needs READ access to the LIBRARY data set.

Reason for check:
The check is intended to identify when a migration action might be needed when migrating from a release below z/OS V1R11.

z/OS releases the check applies to:
z/OS V1.9 and z/OS V1.10

Type of check (local, remote, or REXX):
REXX

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMRMM,ZOSMIGV1R11_RMM_REXX_STEM)
 INTERVAL(ONETIME)
 SEVERITY(MED)
 DATE('date_of_the_change')
 INACTIVE
 REASON('Your reason for making the update.')
 PARM()
```

Debug support:
Yes, additional information is written to REXXOUT data set.

Verbose support:
Yes, when also used with DEBUG, additional information is written to REXXOUT data set.

Parameters accepted:
A required PARM('LIBRARY(*library_dsname*)') is supported. *library_dsname* is a fully qualified data set name, specified without quotes, which contains one or more members that use RMM Rexx variables.

Reference:
For additional information about using ISRSUPC, *z/OS ISPF User's Guide Vol II*.

Messages:
This check issues the following exception messages:

- EDGH8006E
- EDGH8008E

See the EDGH messages in *z/OS MVS System Messages, Vol 5 (EDG-GFS)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R11_RMM_VRSEL_OLD

Description:
This check is used to determine if an installation is still using OPTION VRSEL(OLD) despite migration actions in earlier z/OS releases.

DFSMSrmm must be active for this check to be run. The check should be run once for each system to enable each of the parmlib settings to be verified.

The RACF userid used for the System REXX check, (REXX checks run under the security assigned to the IBM Health Checker for z/OS procedure, *hzsproc*.), issues the RMM LISTCONTROL subcommand and so needs one of the following:

- CONTROL access to STGADMIN.EDG.LISTCONTROL in the FACILITY class
- CONTROL access to STGADMIN.EDG.MASTER in the FACILITY class

Reason for check:
The check is intended to identify when a migration action is needed when migrating from a release below z/OS V1R11.

z/OS releases the check applies to:
z/OS V1.9 and z/OS V1.10

Type of check (local, remote, or REXX):
REXX

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
 CHECK(IBMRMM,ZOSMIGV1R11_RMM_VRSEL_OLD)
 INTERVAL(ONETIME)
 SEVERITY(MED)
 DATE('date_of_the_change')
 INACTIVE
 REASON('Your reason for making the update.')')
```

Debug support:
Yes, additional information is written to REXXOUT data set.

Verbose support:
Yes, when also used with DEBUG, additional information is written to REXXOUT data set.

Parameters accepted:
None.

Reference:
For additional information about migration to VRSEL(NEW) see http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DGT2R370/5.2.4?SHELF=EZ2ZO10L&DT=20080515141920.

Messages:
This check issues the following exception messages:

- EDGH8007E

See the EDGH messages in *z/OS MVS System Messages, Vol 5 (EDG-GFS)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS checks (IBMRRS)

RRS_ArchiveCFStructure

Description:
The check evaluates the coupling facility structure in which the RRS Archive log resides.

Reason for check:
IBM recommends that each RRS log stream reside in its own coupling facility

structure. This is particularly important for the archive log. Allowing the RRS archive log stream to share its coupling facility structure with another log stream is likely to result in sub-optimal use of the storage in the coupling facility structure, which could affect system performance.

z/OS releases the check applies to:
z/OS V1R8 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_ARCHIVECFSTRUCTURE)
    SEVERITY(LOW),INTERVAL(8:00),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- ATRH010E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_RMDataLogDuplexMode

Description:
The duplexing scheme used to protect the RM Data log stream is evaluated.

Reason for check:
Choose a duplexing scheme more reliable than local buffer duplexing for the RM Data log stream. For example, choose to use staging data sets. Why? Because local buffer duplexing can result in a loss of data in the log stream if both the CF and the local buffers are on the same machine. A loss of data in the RRS RM Data log stream will eventually require an RRS cold start to repair the log stream and may also require a cold start of any resource manager using RRS at the time of the RRS cold start. For more details on protecting log streams, see *z/OS MVS Programming: Resource Recovery*.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_RMDATALOGDUPLEXMODE)
    SEVERITY(MEDIUM),INTERVAL(8:00),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH01E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_RMDOffloadSize

Description:
The check evaluates the size of the RM Data log's offload data set.

Reason for check:
The size of the RM Data log's offload data set should be at least as large as the space allocated for the log stream's CF structure. Why? Because a small offload dataset may cause multiple offload data sets to be created for each offload of the CF. The increased overhead in allocating datasets can degrade offload performance and the performance of RRS when reading the log stream.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_RMDOFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH02E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_DUROffloadSize

Description:
The check evaluates the size of the Delayed UR log's offload data set.

Reason for check:
The size of the Delayed UR log's offload data set should be at least as large as the space allocated for the log stream's CF structure. Why? Because a small offload data set may cause multiple offload data sets to be created for each offload of the CF. The increased overhead in allocating data sets can degrade offload performance and the performance of RRS when reading the log stream.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_DUROFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH02E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_MUROffloadSize

Description:
The check evaluates the size of the Main UR log's offload data set.

Reason for check:
The size of the Main UR log's offload data set should be at least as large as the space allocated for the log stream's CF structure. Why? Because a small offload dataset may cause multiple offload data sets to be created for each offload of the CF. The increased overhead in allocating datasets can degrade offload performance and the performance of RRS when reading the log stream.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_MUROFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH02E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_RSTOffloadSize

Description:
The check evaluates the size of the Restart log's offload data set.

Reason for check:
The size of the Restart log's offload data set should be at least as large as the space allocated for the log stream's CF structure. Why? Because a small offload dataset may cause multiple offload data sets to be created for each offload of the CF. The increased overhead in allocating datasets can degrade offload performance and the performance of RRS when reading the log stream.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_RSTOFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH02E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_Storage_NumLargeLOGBlks

Description:
Monitor the level of virtual storage usage in the RRS address space to prevent a terminating failure.

Reason for check:
If the count of large log buffer blocks in RRS grows too big then RRS might encounter a terminating failure.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
      CHECK(IBMRRS,RRS_STORAGE_NUMLARGELOGBLKS)
      ACTIVE
      SEVERITY(HI),INTERVAL(0:05),DATE('date_of_the_change')
      PARM('1000')
      REASON('Your reason for making the update.')
```

Parameters accepted:
The threshold for number of large message blocks in use by RRS; in the range of '0' to '99999999'. The threshold for number of transaction related blocks in use by RRS; in the range of '0' to '99999999'. The default is '1000'.

Debug support:
No

Verbose support:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH020E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Ouptut:

```
ATR020E - The current number of large log buffer blocks in RRS
is currblks which exceeds current threshold of maxblks

Explanation: The number of large message blocks being processed
within RRS at this time has exceeded the threshold specified in
the health check.  This can be an indication of a potential
storage usage failure in RRS.

System Program Response:
```

```
Use the available RRS data collection techniques (panels,
console display command, or batch program) to assess the level
of transaction activity in RRS and determine if it is unusual or
unexpected.

If the level of activity is determined to be a problem then use
the data collection methods to determine if it is a problem with
a specific work manager then check with that work manager
function for problems.

If not a work manager problem then use the data collection
methods to determine if it is a problem with a specific resource
manager.

If it appears to be neither a specific work manager nor a
specific resource manager problem then monitor RRS using this
health check until either the exception is resolved or the count
continues to grow.
```

RRS_Storage_NumLargeMSGBlks

Description:
Monitor the level of virtual storage usage in the RRS address space to prevent a terminating failure.

Reason for check:
If the count of large message blocks in RRS grows too big then RRS might encounter a terminating failure.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
      CHECK(IBMRRS,RRS_STORAGE_NUMLARGEMSGBLKS)
      ACTIVE
      SEVERITY(HI),INTERVAL(0:05),DATE('date_of_the_change')
      PARM('10000')
      REASON('Your reason for making the update.')
```

Parameters accepted:
The threshold for number of large message blocks in use by RRS; in the range of '0' to '99999999'. The threshold for number of transaction related blocks in use by RRS; in the range of '0' to '99999999'. The default is '1000'.

Debug support:
No

Verbose support:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH018E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Ouptput:

```
ATR018E - The current number of large message blocks in RRS is
currblks which exceeds current threshold of maxblks

Explanation: The number of large message blocks being processed
within RRS at this time has exceeded the threshold specified in
the health check.  This can be an indication of a potential
storage usage failure in RRS.

System Program Response:
Use the available RRS data collection techniques (panels,
console display command, or batch program) to assess the level
of transaction activity in RRS and determine if it is unusual or
unexpected.

If the level of activity is determined to be a problem then use
the data collection methods to determine if it is a problem with
a specific work manager then check with that work manager
function for problems.

If not a work manager problem then use the data collection
methods to determine if it is a problem with a specific resource
manager.

If it appears to be neither a specific work manager nor a
specific resource manager problem then monitor RRS using this
health check until either the exception is resolved or the count
continues to grow.
```

RRS_Storage_NumServerReqs

Description:
Monitor the level of virtual storage usage in the RRS address space to prevent a terminating failure.

Reason for check:
If the count of server requests within RRS grows too big then RRS might be encountering a hang situation.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_STORAGE_NUMSERVERREQS)
    ACTIVE
    SEVERITY(HI),INTERVAL(0:05),DATE('date_of_the_change')
    REASON('Your reason for making the update.')
```

Parameters accepted:
No

Debug support:
No

Verbose support:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH016E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Ouptut:

```
ATR016E The current number of server task requests in RRS is
curreqs which exceeds the threshold

Explanation: The number of server task requests in RRS has
exceeded the manageable threshold and could be an indication of
a potential problem in RRS.  Please monitor the level of
activity in RRS and the associated resource managers and see if
anything indicates a slow down or complete halt to transaction
processing.

System Program Response:
Use the available RRS data collection techniques (panels,
console display command, or batch program) to assess the level
of transaction activity in RRS and determine if it is unusual or
unexpected.

If the level of activity is determined to be a problem then use
the data collection methods to determine if it is a problem with
a specific work manager then check with that work manager
function for problems.
```

RRS_Storage_NumTransBlks

Description:
Monitor the level of virtual storage usage in the RRS address space to prevent a terminating failure.

Reason for check:
If the count of transactions that RRS is managing grows too big then RRS might encounter a terminating failure.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
    CHECK(IBMRRS,RRS_STORAGE_NUMTRANSBLKS)
    ACTIVE
    SEVERITY(HI),INTERVAL(0:05),DATE('date_of_the_change')
    PARM('10000')
    REASON('Your reason for making the update.')
```

Parameters accepted:
The threshold for number of transaction related blocks in use by RRS; in the range of '0' to '99999999'. The default is '10000'.

Debug support:
No

Verbose support:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
This check issues the following exception messages:

- ATRH014E

See the ATRH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Ouptput:

```
ATR014E The current number of active RRS transactions is
currtrans which exceeds the current threshold of maxtrans

Explanation:
The number of transactions being managed by RRS at the current
time has exceeded the threshold specified in the health check.
This can be an indication of a potential storage usage failure
in RRS.

System Program Response:
Use the available RRS data collection techniques (panels,
console display command, or batch program) to assess the level
of transaction activity in RRS and determine if it is unusual or
unexpected.

If the level of activity is determined to be a problem then use
the data collection methods to determine if it is a problem with
a specific work manager then check with that work manager
function for problems.

If not a work manager problem then use the data collection
methods to determine if it is a problem with a specific resource
manager.

If it appears to be neither a specific work manager nor a
specific resource manager problem then monitor RRS using this
health check until either the exception is resolved or the count
continues to grow.
```

RSM checks (IBMRSM)

RSM_HVSHARE

Description:
Checks the configured size and current allocation of the high virtual shared area (HVSHARE in IEASYSxx). This check will issue a warning when the allocation of high virtual storage exceeds a predetermined threshold, and/or when the size of the high virtual shared area is less than the default minimum.

Reason for check:
The HVSHARE setting controls the size of the shared area above 2GB, directly affecting how much virtual storage may be shared by jobs on the system. Setting this value too low may cause jobs relying on shared high virtual storage to fail. The default suggested value for this area is 510T.

z/OS releases the check applies to:
z/OS V1R5 and later in z/Architecture® mode only.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMRSM,RSM_HVSHARE),
INTERVAL(00:15),
SEVERITY(LOW),
PARM('THRESHOLD(80%),SIZE(510T)'),
DATE('date_of_the_change')
```

Parameters accepted:
Yes:

- An integer, 0-100, indicating the warning threshold percent (keyword: THRESHOLD, percent sign optional)
- Number of bytes with optional suffix (K,M,G,T,P,E), indicating shared area size (keyword: SIZE)

Default: THRESHOLD(80%),SIZE(510T)

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IARH110E

See the IARH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RSM_MEMLIMIT

Description:
Checks the MEMLIMIT parameter in SMFPRMxx, which affects the amount of high virtual storage available to jobs on the system.

Reason for check:
IBM suggests that jobs requiring virtual storage above 2G use the MEMLIMIT option on the associated JCL EXEC statement to control high virtual storage usage. Additionally, IBM suggests that the IEFUSI exit be used as a secondary limit on the allocation of high virtual storage. Finally, a system wide default MEMLIMIT should be set in SMFPRMxx. This check will issue an exception when the MEMLIMIT setting in SMFPRMxx has been set to zero.

z/OS releases the check applies to:
z/OS V1R4 and later in z/Architecture mode only.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    CHECK(IBMRSM,RSM_MEMLIMIT),
    INTERVAL(ONETIME),
    SEVERITY(LOW),
    DATE('date_of_the_change')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*, *z/OS MVS Programming: Extended Addressability Guide*, and *z/OS MVS Installation Exits*.

Messages:
This check issues the following exception messages:

- IARH109E

See the IARH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RSM_MAXCADS

Description:
The setting of MAXCADS in IEASYSxx, and the number of in-use common area data spaces. A warning will be issued if the number of common area dataspaces exceeds a predetermined threshold.

Reason for check:
Once the number of in use common area dataspaces reaches the value specified in MAXCADS, no more common area dataspaces can be created. This may adversely affect starting new jobs, or the continued operation of jobs already running. This check will help to ensure that the MAXCADS setting is adequate.

z/OS releases the check applies to:
z/OS V1R4 and later in z/Architecture mode only.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    CHECK(IBMRSM,RSM_MAXCADS),
    INTERVAL(00:15),
    SEVERITY(MED),
    PARM('THRESHOLD(80%)'),
    DATE('date_of_the_change')
```

Parameters accepted:
An integer, 0-100, indicating the warning threshold percent (keyword: THRESHOLD, percent sign optional)

Default: THRESHOLD(80%)

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IARH108E

See the IARH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RSM_AFQ

Description:
Whether available frame queue threshold values used for reclaiming storage frames are too low.

Reason for check:
To avoid situations where the system does not start to reclaim storage frames soon enough, you should evaluate the values for storage. If you are running in ESA mode, both the MCCAFCTH and the MCCAECTH values are used. If you are running in z/Architecture mode, only the MCCAFCTH value is used. For migrations to a 64-bit environment, this check is critical because using the same value that was used in ESA mode could introduce problems. IBM suggests that the IEAOPTxx parameters are set as follows:

- MCCAFCTH specifies the low and the OK threshold values for central storage. The lowvalue indicates the number of frames on the available frame queue when stealing begins. The okvalue indicates the number of frames on the available frame queue when stealing ends. You can monitor actual conditions on the RMF Paging Activity Report (RMF Monitor 1) or a equivalent performance monitoring product and adjust accordingly.
- MCCAECTH specifies the low and the OK threshold values for expanded storage. The lowvalue indicates the number of frames on the available frame queue when real storage manager (RSM) frame stealing begins. The okvalue indicates the number of frames on the available frame queue when stealing ends. You can monitor actual conditions on the RMF Paging Activity Report (RMF Monitor 1) or equivalent performance monitoring product and adjust accordingly.

In 31–bit mode, the defaults are sufficient. For these two parameters, the defaults are MCCAFCTH=(50,100), and MCCAECTH=(150,300). The OK point for available frames in a 31-bit mode implementation is 400 frames, 100 from central storage and 300 from expanded storage.

For 64–bit mode (after the installations of APARs OW55902 and OW55729), the default values for MCCAFCTH are (400,600). These are IBM's minimum suggested settings. Although IBM suggests using the defaults, higher values are acceptable.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    CHECK(IBMRSM,RSM_AFQ),
    INTERVAL(ONETIME),
    SEVERITY(HI),
    PARM('AFQLOW(400),AFQOK(600)'),
    DATE('date_of_the_change')
```

Parameters accepted:

1. The number of frames for the MCCAFCTH LOW threshold (keyword: AFQLOW)
2. The number of frames for the MCCAFCTH OK threshold (keyword: AFQOK)
3. The number of frames for the MCCAECTH LOW threshold (ESA only, keyword: EXPLOW)
4. The number of frames for the MCCAECTH OK threshold (ESA only, keyword: EXPOK)

Reference:

For more information on MCCAFCTH and MCCAECTH IEAOPTxx parameters, see *z/OS MVS Initialization and Tuning Reference*. For more information on using the Paging Activity report, see *z/OS RMF Report Analysis* and the whitepaper, WP100269 "z/OS Performance: Managing Processor Storage in a 64–bit environment".

Messages:

This check issues the following exception messages:

- IARH100E

See the IARH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:

SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RSM_REAL

Description:

The REAL setting in IEASYSxx, which controls the amount of central storage that can be allocated concurrently for ADDRSPC=REAL (V=R) jobs.

Reason for check:

IBM suggests that the REAL setting should be set to 0. However, this would not be valid if you have a need to run V=R jobs. Setting REAL=0 in IEASYSxx will improve performance.

z/OS releases the check applies to:

z/OS V1R4 and later, in both ESA and z/Architecture modes.

User override of IBM values:

The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    CHECK(IBMRSM,RSM_REAL),
    INTERVAL(ONETIME),
    SEVERITY(LOW),
    DATE('date_of_the_change')
```

Parameters accepted:

No.

Reference:
For more information on real storage, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IARH101E

See the IARH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RSM_RSU

Description:
The RSU setting in IEASYSxx, which controls the amount of central storage that can be reconfigured.

Reason for check:
IBM suggest that the RSU setting should be set to 0. However, this would not be valid if you have a need to reconfigure storage. Setting RSU=0 in IEASYSxx will improve performance.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMRSM,RSM_RSU),
INTERVAL(ONETIME),
SEVERITY(LOW),
DATE('date_of_the_change')
```

Parameters accepted:
No.

Reference:
For more information on reconfigurable storage, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IARH102E

See the IARH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RTM checks (IBMRTM)

RTM_IEAVTRML

Description:
Validate that no resource manager module names are specified in CSECT IEAVTRML of load module IGC0001C.

Reason for check:
Installations should use the RESMGR service instead of IEAVTRML to define End of Task (EOT) and End of Memory (EOM) resource managers to the system. If installations use IEAVTRML, RTM calls the resource manager for every EOT and EOM in the system. In most situations, this results in thousands of unnecessary invocations of the resource manager per day which impacts system performance.

z/OS releases the check applies to:
z/OS V1R11 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMRTM,RTM_IEAVTRML)
ACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
PARM('ALL')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
Yes. When VERBOSE(YES) is specified, all of the resource managers found in IEAVTRML will be listed in the message log whether or not an exception is raised.

Parameters accepted:

- PARM('ALL') specifies that exceptions should be issued for all module names specified in IEAVTRML. This is the system default.
- PARM('NEW(value)') specifies that the current contents of IEAVTRML are to be treated as correct and exceptions should be issued only for new module names added to IEAVTRML after this time. This specification persists across restarts of the Health Checker, including IPLs. Note that the system only recognizes changes to IEAVTRML via an IPL with CLPA.

Reference:
For more information about EOT and EOM resource managers, see Using Resource Managers in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

Messages:
This check issues the following exception messages:

- IEAVTRH03I

See the IEAVTRH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SDSF checks (IBMSDSF)

SDSF_CLASS_SDSF_ACTIVE

Description:
Checks that the SAF class SDSF is active.

Reason for check:
SAF based security is used to protect SDSF functions.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMSDSF,SDSF_CLASS_SDSF_ACTIVE),
INTERVAL(ONETIME),
SEVERITY(LOW),
DATE('date_of_the_change')
```

Debug support:
Yes, causes diagnostic message containing the return and reason codes from the RACROUTE service to be issued.

Verbose support:
No.

Parameters accepted:
None.

Reference:
For more information, see *z/OS SDSF Operation and Customization*.

Messages:
This check issues the following exception messages:

- ISFH1016E

See SDSF messages in *z/OS SDSF Operation and Customization*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SDSF_ISFPARMS_IN_USE

Set-up for check:
If you run the SDSF server, no set-up is required for this check - the server will automatically register SDSF health checks with IBM Health Checker for z/OS during server initialization.

If you do not run the SDSF server, you must do some set-up to define this check to IBM Health Checker for z/OS in the PROGxx parmlib member. To do

this, copy sample member ISFSPROG from ISF.SISFJCL into your PROGxx member. Then, issue the SET PROG=xx command to activate that PROGxx parmlib member.

If you want to define the check to the system dynamically, you can issue the SETPROG command as follows:

```
SETPROG EXIT,ADD,EXITNAME(HZSADDCHECK),MODNAME(ISFHCADC)
```

Then, issue the MODIFY command to add all new checks to the system:

```
F hzsproc,ADDNEW
```

Description:

- Checks that SDSF dynamic statements in ISFPRMxx are being used for configuration options to avoid reassembly of ISFPARMS.
- Checks that if ISFPARMS is being used, only default values have been specified.

Reason for check:

SDSF's internal parameters are used for specifying global configuration options, panel formats, and security for SDSF function. There are two alternatives for SDSF's internal parameters:

- Assembler macros that you define, assemble and link into the SDSF load library (ISFPARMS)
- Statements that reside in parmlib member ISFPRMxx.

IBM suggests that you use parmlib member ISFPRMxx rather than the assembler format ISFPARMS because some options are only available using the statement format. In addition, you must assemble the macros with every release, and the ISFPARMS from one release cannot be shared with the ISFPARMS from another release.

z/OS releases the check applies to:

z/OS V1R8 and later.

User override of IBM values:

The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
   CHECK(IBMSDSF,SDSF_ISFPARMS_IN_USE),
   INTERVAL(ONETIME),
   SEVERITY(LOW),
   PARM('SERVER(SDSF)'),
   DATE('date_of_the_change')
```

Debug support:

No

Verbose support:

Yes. Controls whether additional messages are issued describing ISFPARMS keywords that have been found to be customized.

Parameters accepted:

Yes; SDSF server name to be processed (keyword SDSF). The default value is SDSF.

Reference:

For more information, see *z/OS SDSF Operation and Customization*.

Messages:
This check issues the following exception messages:

- ISFH1005E

See SDSF messages in *z/OS SDSF Operation and Customization*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SDUMP checks (IBMSDUMP)

SDUMP_AVAILABLE

Description:
Ensures that SDUMP is enabled to collect SVC Dumps.

Reason for check:
When a system program experiences a condition requiring a snapshot of virtual storage, it can request an SVC dump.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMSDUMP,SDUMP_AVAILABLE)
SEVERITY(MEDIUM)
INTERVAL(OneTime)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Parameters accepted:
No.

Reference:
For more information on SDUMP, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Messages:
This check issues the following exception messages:

- IEAH701I
- IEAH703I

See the IEAH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SDUMP_AUTO_ALLOCATION

Description:
Checks to see whether automatic allocation of SVC dump data sets is enabled.

Reason for check:
Automatic allocation of SVC dump data sets.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMSDUMP,SDUMP_AUTO_ALLOCATION)
SEVERITY(MEDIUM)
INTERVAL(OneTime)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Parameters accepted:
No.

Reference:
For more information, see *z/OS MVS Diagnosis: Tools and Service Aids*.

Messages:
This check issues the following exception messages:

- IEAH701I
- IEAH703I

See See the IEAH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Serviceability checks (IBMSLIP)

SLIP_PER

Description:
Checks to see whether a SLIP PER trap has been continuously active for longer than the threshold.

Reason for check:
An active, but not needed, SLIP PER trap can cause degraded system performance

z/OS releases the check applies to:
z/OS V2R1 and higher.

Parameters accepted:
Yes, the following parameters are accepted:

`TIME(DAYS,n) or TIME(HOURS,n)`

TIME(DAYS,*n*) or TIME(HOURS,*n*)
Each of which identifies the length of time that a SLIP PER trap must be active before the exception is raised. "*n*" may range from 1-9999.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
    CHECK(IBMSLIP,SLIP_PER),
    INTERVAL(4:00),
    SEVERITY(LOW),
    PARM('TIME(DAYS,30)'),
    DATE('20130901'),
    Reason('Your reason for making the update.')
```

Debug support:
No.

Verbose support:
No.

Reference:
See the SLIP command in *z/OS MVS System Commands*.

Messages:
This check issues the following exception messages:

- IEAH101E

See IEAH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for MLS users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SMB checks (IBMSMB)

SMB_NO_ZFS_SYSPLEX_AWARE

Description:
Determines if the DFS/SMB File Server is running in a sysplex and if so, determines if any member of the sysplex is running zFS sysplex aware.

Reason for check:

In a sysplex environment, exportation of a zFS file system and subsequent sharing by the DFS/SMB server can only take place on the system that owns the file system and is not running zFS sysplex aware. Beginning with z/OS V1R11, the SMB server cannot export zFS read/write file systems when zFS is running sysplex-aware on either:

- The same system where the SMB server is running
- The system that owns the zFS file system

If you want to export zFS file systems using the SMB server, you must configure zFS as non-sysplex aware (by specifying sysplex=off in the zFS IOEFSPRM configuration file).

z/OS releases the check applies to:
z/OS V1R11 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMSMB,SMB_NO_ZFS_SYSPLEX_AWARE)
  SEVERITY(MEDIUM)
```

```
INTERVAL(ONETIME)
   DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For additional information see *z/OS Distributed File Service SMB Administration*.

Messages:
This check issues the following exception messages:

- IOEWH0011E

See *z/OS Distributed File Service Messages and Codes*.

ZOSMIGREC_SMB_RPC

Description:
Determines if the DFS/SMB File Server is running in conjunction with Distributed Computing Environment(DCE) or DCE DFS or both

Reason for check:

Beginning with z/OS V1R11, SMB can still run with DCE and DCE/DFS using the RPC protocol, but this environment might not be supported by IBM much longer. To prepare for this change, migrate data in DCE/DFSD Episode file systems to TFS, HFS, or zFS file systems.

z/OS releases the check applies to:
z/OS V1R11 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. You can copy and modify this statement to override the check defaults:

```
UPDATE
  CHECK(IBMSMB,ZOSMIGREC_SMB_RPC)
  SEVERITY(MEDIUM)
  INTERVAL(ONETIME)
  DATE(20090219)
  REASON('Checks whether or not SMB is running on conjunction with the
     Distributed Computing Environment (DCE) as well as the DCE/DFS File Server')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For additional information see *z/OS Distributed File Service SMB Administration*.

Messages:
This check issues the following exception messages:

- IOEWH0020E

See *z/OS Distributed File Service Messages and Codes*.

SMS checks (IBMSMS)

SMS_CDS_REUSE_OPTION

Description:
This check verifies that the active control data set (ACDS) and communications data set (COMMDS) are defined with the REUSE option.

Reason for check:
As a best practice, defining ACDS or COMMDS with the REUSE option helps to avoid running into space problems (SMS reason code 6068) as result of subsequent ACDS or COMMDS updates, or IMPORT/EXPORT functions.

z/OS releases the check applies to:
z/OS V1R12 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMSMS, SMS_CDS_REUSE_OPTION)
 SEVERITY(MED)
 DATE('date_of_the_change')
 REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For additional information see Allocating an ACDS and Allocating a COMMDS in *z/OS DFSMSdfp Storage Administration*.

Messages:
This check issues the following exception messages:

- IGDH1011E

See *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

SMS_CDS_SEPARATE_VOLUMES

Description:
This check verifies that the active control data set (ACDS) and communications data set (COMMDS) are not residing on same volume.

Reason for check:
To ease recovery in case of failure, the ACDS should reside on a different volume than the COMMDS. Also, you should allocate a spare ACDS on a different volume. The control data set (ACDS or COMMDS) must reside on a

volume that is not reserved by other systems for a long period of time because the control data set (ACDS or COMMDS) must be available to access for SMS processing to continue.

z/OS releases the check applies to:
z/OS V1R12 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMSMS, SMS_CDS_SEPARATE_VOLUMES)
 SEVERITY(MED)
 DATE('date_of_the_change')
 REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For additional information see Allocating an ACDS and Allocating a COMMDS in *z/OS DFSMSdfp Storage Administration*.

Messages:
This check issues the following exception messages:

- IGDH1001E

See *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

Supervisor checks (IBMSUP)

IEA_ASIDS

Description:
This check reports on available "normal" and "replacement" ASIDs

Reason for check:
ASIDs are a finite resource. It is important to know how many remain available. Running the system in exception has no consequence. The exception is intended to alert to the possibilities.

z/OS releases the check applies to:
z/OS V1R9 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMSUP,IEA_ASIDS),
      INTERVAL(01:00),
```

```
SEVERITY(LOW),
PARM('NORMAL(5%),REPLACEMENT(5%),DAYSUNTILIPL(1)'
    ),
DATE('20060424')
Reason('ASIDs are a finite resource. It is important to ',
       'know how many remain available.')
```

Debug support:
No

Verbose support:
Yes. When VERBOSE mode is in effect, information about individual connections to non-reusable ASIDs is provided

Parameters accepted:

- NORMAL(n) specifies an integer 1-ASVTMAXI or a percent 1-100 (which is applied to the value of ASVTMAXI, the number of total possible normal ASIDs). If the number of available normal ASIDs falls below the limit, an exception message is issued. The default is 5%.
- REPLACEMENT(n) specifies an integer 1-ASVTNONR or a percent 1-100 (which is applied to the value of ASVTNONR, the number of total possible replacement ASIDs). If the number of available replacement ASIDs falls below the limit, an exception message is issued. The default is 5%.
- DAYSUNTILIPL(n) specifies an integer 1-99999. If the system will run out of ASIDs in n days, given the rate of ASID depletion calculated from the currently available information, an exception message is issued. The default is 1.

Reference:

- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS Initialization and Tuning Guide*

Messages:
This check issues the following exception messages:

- IEAVEH020E,
- IEAVEH021E,
- IEAVEH060E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA).*

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

IEA_LXS

Description:
This check reports on available system and non-system LXs and extended LXs (ELXs)

Reason for check:
LXs are a finite resource. It is important to know how many remain available. Running the system in exception has no consequence. The exception is intended to alert to the possibilities.

z/OS releases the check applies to:
z/OS V1R9 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
      CHECK(IBMSUP,IEA_LXS),
      INTERVAL(01:00),
      SEVERITY(LOW),
      PARM('LX(15%),ELX(15%),SYSLX(15%),SYSELX(15%)'
          ),
      DATE('20060424')
      REASON('LXs are a finite resource. It is important to ',
             'know how many remain available.')
```

Debug support:
No

Verbose support:
Yes. When VERBOSE mode is in effect, information about each individual LX is provided

Parameters accepted:

- LX(n) specifies an integer 0-SvtxLXNSysDefined, or a percent 1-100 (which is applied to the value of SvtxLXNSysDefined, the number of defined non-system LXs). If the number of non-system LXs falls below the limit, an exception message is issued. The default is 15%.
- ELX(n) specifies an integer 0-SvtxBLXNSysDefined, or a percent 1-100 (which is applied to the value of SvtxBLXNSysDefined, the number of defined non-system extended LXs). If the number of non-system extended LXs falls below the limit, an exception message is issued. The default is 15%.
- SYSLX(n) specifies an integer 0-SvtxLXSysDefined, or a percent 1-100 (which is applied to the value of SvtxLXSysDefined, the number of defined system LXs). If the number of system LXs falls below the limit, an exception message is issued. The default is 15%.
- SYSELX(n) specifies an integer 0-SvtxBLXSysDefined, or a percent 1-100 (which is applied to the value of SvtxBLXSysDefined, the number of defined system extended LXs). If the number of system extended LXs falls below the limit, an exception message is issued. The default is 15%.

Reference:

- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS Initialization and Tuning Guide*

Messages:
This check issues the following exception messages:

- IEAVEH050E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SUP_HIPERDISPATCH

Description:
This check verifies whether the check's expected HiperDispatch state matches

the actual HiperDispatch state of the system. The following terms are used to describe the Hiperdispatch State (where terms appearing on the same line are used interchangeably):

- YES and enabled
- NO and disabled

The check's expected HiperDispatch state is determined by the parameters specified or defaulted for this check. See the parameters section for further details.

Note that the system will register this check only on machines that support HiperDispatch. IBM System z10® is the first machine that supports HiperDispatch.

Reason for check:

HiperDispatch provides a performance improvement by optimizing the use of system cache. The performance gain HiperDispatch provides typically increases with a newer hardware generation and can improve with newer releases of z/OS.

Before enabling HiperDispatch for the first time, review the "Planning Considerations for HiperDispatch Mode" White Paper located on IBM Techdocs at http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101229.

When a machine of a newer hardware generation is installed, for any z/OS partition(s) that are running with HiperDispatch disabled, the system programmer should reevaluate whether those z/OS partition(s) should be migrated to run with HiperDispatch enabled in the new environment.

When a new z/OS release contains a noteworthy performance improvement for HiperDispatch=YES, the date of the Health Check will be updated. The date is updated to alert the system programmer that partitions running with HiperDispatch disabled should be reevaluated to see if it is appropriate to migrate those partitions to HiperDispatch enabled.

On any z/OS release running on IBM System z10 hardware, HiperDispatch disabled is the default. On IBM System z10 systems, customers are encouraged to try running with HiperDispatch enabled.

Beginning with z/OS V1R13 on zEnterprise 196 hardware, HiperDispatch enabled is the default. With zEnterprise 196 hardware, z/OS partitions with share greater than two physical processors will typically experience improved processor efficiency with HiperDispatch enabled. z/OS partitions with share less than 2 physical processors typically do not receive a detectable performance improvement with HiperDispatch enabled, but IBM recommends running those partitions with HiperDispatch enabled when the performance improvement is greater than or equal to HiperDispatch disabled. Initially all z/OS partitions on non-IBM System z10 machines that run HiperDispatch disabled will result in this Health Check raising an exception. The system programmer can supply the machine type to this Health Check to indicate that a partition is intentionally running with HiperDispatch disabled on a particular machine type.

IBM suggests that all partitions that experience improved or equivalent processor efficiency with HiperDispatch enabled continue running with HiperDispatch enabled.

z/OS releases the check applies to:

z/OS V1R13 with apar OA36150 on a zEnterprise 196.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
   CHECK(IBMSUP, SUP_HIPERDISPATCH)
   ACTIVE
   SEVERITY(MED)
   INTERVAL(24:00)
   DATE('date_of_the_change')
   PARM('HIPERDISPATCH(YES),MachTypes(aaaa,bbbb,...)')
   REASON('Your reason for making the update.');
```

Debug support:
No

Verbose support:
No

Parameters accepted: Yes:

- If the HIPERDISPATCH keyword has a value of 'YES', the check expects HiperDispatch to be enabled on the machine. The default is 'YES'.
- If the HIPERDISPATCH keyword has a value of 'NO' and the machine is an IBM System z10, the check expects that HiperDispatch is disabled on the machine.
- If the HiperDispatch keyword has a value of 'NO' and the machine is not an IBM System z10, the check's expected HiperDispatch state depends on the MachTypes parameter.
- MachTypes is optional parameter that contains a list of up to 10 machine types (for example, the zEnterprise 196 machine type is 2817). When specified, the MachTypes parameter is always syntactically validated, but it has no effect on the check's expected HiperDispatch state when the system is a IBM System z10 or the HIPERDISPATCH(YES) parameter was specified or defaulted. The MachTypes parameter can only affect the check's expected HiperDispatch state on a non-IBM System z10 machine with the HiperDispatch(NO) parameter specified. In this case, the check's expected HiperDispatch state is determined as follows
  - When the current machine type is not in the MachTypes list or the MachTypes parameter is not specified, the check expects HiperDispatch to be enabled.
  - When the current machine type is in the MachTypes list, the check expects HiperDispatch to be disabled.

Reference:

- IEAOPTxx (OPT parameters) in*z/OS MVS Initialization and Tuning Reference*
- SET command in *z/OS MVS System Commands*
- White Paper titled "Planning Considerations for HiperDispatch Mode" located on IBM Techdocs: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101229

Messages:
This check issues the following exception messages:

- IEAVEH071E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SUP_HiperDispatchCPUConfig

Description:
This check monitors the number of CPUs installed and HiperDispatch state of the system. On systems where HiperDispatch is disabled, this check provides a warning when the highest CPU ID of any CPU configured to this system is close to the maximum allowed (X'3F'). On systems where HiperDispatch is enabled, this check is always successful.

The system runs this check whenever any of the following occur:

- IBM Health Checker for z/OS starts
- HiperDispatch mode switch
- A CPU is dynamically added to the system's configuration

Note that you can only add this check on z/OS releases that support CPU ids greater than X'3F' and hardware capable of supporting more than 64 CPUs.

Reason for check:
A system with HiperDispatch disabled can use CPU ID 0 through CPU ID X'3F'. For the system to use CPU IDs above X'3F', the system must have HiperDispatch enabled.

z/OS releases the check applies to:
z/OS V1R11 and above with APAR OA30476 installed running on z hardware capable of running with more than 64 CPUs.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMSUP, SUP_HiperDispatchCPUConfig)
ACTIVE
SEVERITY(LOW)
INTERVAL(ONETIME)
DATE('date_of_the_change')
PARM('CpusLeftB4NeedHd(8)')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, for a machine with HiperDispatch disabled.
PARM('CpusLeftB4NeedHd(*n*)') specifies the minimum number of remaining CPUs which must be able to be installed and used with HiperDispatch

disabled for the check to succeed. This value can be from 0 - 63. The default is 8. If you specify a value of 0 for this parameter, the check will never find an exception.

The system ignores this parameter when HiperDispatch is enabled.

Reference:

- IEAOPTxx (OPT parameters) in*z/OS MVS Initialization and Tuning Reference*
- Information on HiperDispatch mode in *z/OS MVS Planning: Workload Management*
- SET command in *z/OS MVS System Commands*.

Messages:

This check issues the following exception messages:

- IEAVEH081E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:

SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SUP_LCCA_ABOVE_16M

Description:

Checks to see whether the residency mode (RMODE), specified for the LCCA control block in the CBLOC parameter of the DIAGxx parmlib member, is the expected value. The default RMODE for the LCCA control block is RMODE 31. The check will look for RMODE 31 for the LCCA control block unless you specify an RMODE of 24 in the RMODE parameter for the check.

Reason for check:

The suggested RMODE for the LCCA control block is RMODE 31.

z/OS releases the check applies to:

z/OS V1R12 and higher.

Type of check (local or remote):

Local

User override of IBM values:

The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMRCF,SUP_LCCA_ABOVE_16M),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('CBLOC(31)'),
DATE('date_of_the_change')
Reason('Your reason for making the update.')
```

Debug support:

No

Verbose support:

No

Parameters accepted:

- CBLOC(31), which is the default, specifies that you want the check to generate an exception if it finds that IHALCCA had been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member.
- CBLOC(24) specifies that you want the check to generate an exception if it finds that IHALCCA either:
  - Had been specified within the CBLOC VIRTUAL31 parameter of the DIAGxx parmlib member.
  - Had not been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member (because CBLOC VIRTUAL31 is the default for the LCCA)..

Reference:
See the CBLOC parameter in the DIAGxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IEAVEH091E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SUP_SYSTEM_SYMBOL_TABLE_SIZE

Description:
Checks to see whether the size of the static system symbol table has exceeded the threshold. The check is initially run once and is also run when the SETLOAD xx,IEASYM command is successfully processed.

Reason for check:
Monitor the size of the system symbol table.

z/OS releases the check applies to:
z/OS V2R1 and higher.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMSUP,SUP_SYSTEM_SYMBOL_TABLE_SIZE),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('LIMIT(85%)'),
DATE('20100901'),
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
LIMIT({*n* | *p*%}) which defines a threshold value. The value of *n* may be a decimal number in the range 1 to 32512. The value of *p*% identifies a percentage *p* in the range 1 to 100, from which the system calculates the threshold value, based on the system maximum symbol table size of 32512 bytes.

Reference:
See the following in *z/OS MVS Initialization and Tuning Reference*:

- What are system symbols?
- IEASYMxx
- LOADxx
-

Messages:
This check issues the following exception messages:

- IEAVEH111E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA).*

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

Example of success message:

```
IEAVEH110I The system symbol table size is 264 bytes. This has not
exceeded the installation-specified threshold of 264 bytes.
The maximum size is 32512 bytes.
```

Example of exception message:

```
IEAVEH111E The system symbol table size is 268 bytes.
The installation-specified threshold of 264 bytes has been exceeded.
The maximum size is 32512 bytes.

  Explanation:  CHECK(IBMSUP,SUP_SYSTEM_SYMBOL_TABLE_SIZE) determined
    that the system symbol table size has exceeded the
    installation-specified threshold.

  System Action:  The system continues processing.

  Operator Response:  N/A

  System Programmer Response:  If you think you will need to add
    additional symbols in the future, see if you can consolidate or
    eliminate ones that already are defined.

Problem Determination:  N/A

Source:  Supervisor

Reference Documentation:  "What are system symbols", IEASYMxx, and
  LOADxx in z/OS MVS Initialization and Tuning Reference

Automation:  N/A

Check Reason:  Monitor the size of the system symbol table
```

ZOSMIGV1R12_SUP_LCCA_ABOVE_16M

Description:
Checks to see whether the residency mode (RMODE), specified for the LCCA

control block in the CBLOC parameter of the DIAGxx parmlib member, is the expected value. The default RMODE for the LCCA control block on z/OS systems at the pre-z/OS R12 level is RMODE 24. The check will look for RMODE 24 for the LCCA control block unless you specify an RMODE of 31 in the RMODE parameter for the check.

Reason for check:
The default RMODE for the LCCA control block is changing from RMODE 24 to RMODE 31 in z/OS V1R12.

z/OS releases the check applies to:
z/OS V1R10 and V1R11.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMSUP,ZOSMIGV1R12_SUP_LCCA_ABOVE_16M),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('CBLOC(31)'),
DATE('date_of_the_change')
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:

- CBLOC(31), which is the default, specifies that you want the check to generate an exception if it finds that IHAPCCA had been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member.
- CBLOC(24) specifies that you want the check to generate an exception if it finds that IHAPCCA either:
  - Had been specified within the CBLOC VIRTUAL31 parameter of the DIAGxx parmlib member.
  - Had not been specified within the CBLOC VIRTUAL24 parameter of the DIAGxx parmlib member (because CBLOC VIRTUAL24 is the default for the PCCA).

Reference:
See the CBLOC parameter in the DIAGxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IEAVEH091E

See the IEAVEH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

System logger checks (IBMIXGLOGR)

IXGLOGR_STAGINGDSFULL

Description:

Reports on log streams that have encountered staging data set full conditions. When a staging data set is full, new write operations to the log stream fail. Applications using log streams with a staging data set that fills might experience a slow down or an outage if the condition is not resolved in a timely fashion. SMF must be active for system logger to report on staging dataset full conditions when log streams remain connected.

Reason for check:

IBM recommends that tuning actions be taken to avoid future full conditions. For more details see message description for IXGH008E.

After conditions are sufficiently corrected before a certain time, set that as a TIME(mm/dd/yyyy hh:mm:ss) parameter in an UPDATE statement. This will suppress conditions older than that time. You can cut and paste the time from the report output.

After updating to TIME(mm/dd/yyyy hh:mm:ss), if you want to see counts for the log stream values, you'll either have to switch to the ALL parameter, or review the information from a SMF report.

We recommend that check output be saved for historical purposes, especially if you perform parameter updates. See the following information for using HZSPRINT to record check output to a data set or log stream:

- "Optionally set up the HZSPRINT utility" on page 12
- "Setting up security for the HZSPRINT utility" on page 16
- "Using the HZSPRINT utility" on page 37

Because records used to report on this condition are typically filled in at the time of the system logger SMF reporting interval, conditions that occur between the time of the last SMF interval and the time the check is run may be missed. When ALL is specified, these conditions will appear when the report is run after the next system logger SMF reporting interval. If you update to a TIME(mm/dd/yyyy hh:mm:ss), parameter after the most recent SMF reporting interval, some conditions may never be reported on. You can either accept this loss, set a TIME(mm/dd/yyyy hh:mm:ss) before the most recent SMF interval, or use the ALL parameter to view these conditions when they become available.

z/OS releases the check applies to:

z/OS V1R7 and later.

Parameters accepted:

Yes, the check accepts the following parameters to control the time this check reports conditions from:

PARM('ALL')

The default parameter, ALL specifies that the check display conditions that happened since system logger was initialized, up to the most recent 16 log streams that have conditions. For each condition, the report lists the log stream name, corresponding structure name, time of last occurrence, and count of occurrences since logger started for each log stream that has a condition.

PARM('TIME(mm/dd/yyyy hh:mm:ss:)')
This parameter specifies that the check display staging data set full occurrences that happened since the requested time. For each condition, the report lists the log stream name, corresponding structure name, and time of last occurrence. Counts will not be shown when this parameter is active. The input time, must be a valid GMT, in the requested format, and can not be a time in the future.

Message IXGH008E or IXGH005I indicate the time the check is reporting from. If the system detects parameters in an incorrect format, it issues message IXGH004I and the check stops.

User override of IBM values:
The following shows keywords you can use to override check values on either the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL) PARM('ALL')
       SEVERITY(LOW) INTERVAL(4:00) DATE('date_of_the_change')
       REASON('Your reason for making the update.')
```

Use the MODIFY command with the UPDATE parameter to change the IXGLOGR_STAGINGDSFULL check parameters, as follows:

```
F HZSPROC,UPDATE,CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL)
         ,PARM('ALL' )

F HZSPROC,UPDATE,CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL)
         ,PARM('TIME(10/10/2007 17:42:34)')
```

You can also use system symbols in an HZSPRMxx member to modify a check to the current time. The following example shows how to update the check to only report on current log streams conditions:

1. Update HZSPRMxx to put system symbols in the parameter for IXLOGR_STAGINGDSFULL., as the following example shows:

   ```
   UPDATE CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL)
            PARM('TIME(&MON/&DAY/&YR4 &HR:&MIN:&SEC)')
   ```

2. Using the following console command to add the updated HZSPRMxx parmlib members to the list of members that IBM Health Checker for z/OS is using:

   ```
   F HZSPROC,ADD,PARMLIB=xx
   ```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXGH008E

See the IXGH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

Output: The following shows output from IXGLOGR_STAGINGDSFULL:

- Output for the no exceptions case:

```
CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL)
START TIME: 10/10/2007 13:30:20.109067
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: ALL
```

```
IXGH005I This system has not encountered any log stream staging data set
full conditions since 10/10/2007 13:25:30 (GMT).

END TIME: 10/10/2007 13:30:20.109198  STATUS: SUCCESSFUL
```

- Output for the check running with PARM(ALL) and some exceptions found:

```
CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL)
START TIME: 10/10/2007 13:52:38.668631
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: ALL

* Low Severity Exception *

IXGH008E One of more log streams encountered a staging data set
full condition since 10/10/2007 13:25:30 (GMT).
...
                                                 Time of Last
Log Stream                Structure        Count Condition (GMT)
TESTLOG1.HCHECK1.D6       *DASDONLY*       12    10/10/2007 17:37:49
TESTLOG1.HCHECK1.S5       STRUCT5          49    10/10/2007 17:38:56
TESTLOG1.HCHECK1.D3       *DASDONLY*       26    10/10/2007 17:40:12
TESTLOG1.HCHECK1.D5       *DASDONLY*       66    10/10/2007 17:41:28
TESTLOG1.HCHECK1.S7       STRUCT7          47    10/10/2007 17:42:34
TESTLOG1.HCHECK1.D1       *DASDONLY*       16    10/10/2007 17:44:00
TESTLOG1.HCHECK1.D2       *DASDONLY*       29    10/10/2007 17:45:06
TESTLOG1.HCHECK1.D4       *DASDONLY*       89    10/10/2007 17:46:12

END TIME: 10/10/2007 13:52:38.670195  STATUS: EXCEPTION-LOW
```

- Output for the check after it is updated to PARM('TIME(10/10/2007 17:42:34)' and run again. The check now only shows exceptions occurring after the time specified:

```
CHECK(IBMIXGLOGR,IXGLOGR_STAGINGDSFULL)
START TIME: 10/10/2007 13:59:57.386351
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: TIME(10/10/2007 17:42:34)

* Low Severity Exception *

IXGH008E One of more log streams encountered a staging data set
full condition since 10/10/2007 17:42:34 (GMT).
...
                                                 Time of Last
Log Stream                Structure        Count Condition (GMT)
TESTLOG1.HCHECK1.S7       STRUCT7          47    10/10/2007 17:42:34
TESTLOG1.HCHECK1.D1       *DASDONLY*       16    10/10/2007 17:44:00
TESTLOG1.HCHECK1.D2       *DASDONLY*       29    10/10/2007 17:45:06
TESTLOG1.HCHECK1.D4       *DASDONLY*       89    10/10/2007 17:46:12

END TIME: 10/10/2007 13:59:57.388904  STATUS: EXCEPTION-LOW
```

IXGLOGR_ENTRYTHRESHOLD

Description:

Reports on log streams that have encountered structure entry threshold conditions. This means that the structure had 90% of the entries in use at one time. When the entries reach 100%, write operations to all of the log streams in the structure fail until the full condition is resolved. Applications using the affected log streams might experience a slow down or possibly an outage if the condition is not resolved in a timely fashion. SMF must be active for system logger to report on entry threshold reached conditions when log streams reamin connected.

Because this check may flag valid log stream configurations as exceptions, it has been set to inactive by default. In order to use this check you must activate it.

Reason for check:

IBM suggests that tuning actions be taken to avoid future entry full conditions. For more details see message description for IXGH009E.

After conditions are sufficiently corrected before a certain time, set that as a TIME(mm/dd/yyyy hh:mm:ss) parameter in an UPDATE statement. This will suppress conditions older than that time. You can cut and paste the time from the report output.

After updating to TIME(mm/dd/yyyy hh:mm:ss), if you want to see counts for the log stream values, you'll either have to switch to the ALL parameter, or review the information from a SMF report.

We recommend that check output be saved for historical purposes, especially if you perform parameter updates. See the following information for using HZSPRINT to record check output to a data set or log stream:

- "Optionally set up the HZSPRINT utility" on page 12
- "Setting up security for the HZSPRINT utility" on page 16
- "Using the HZSPRINT utility" on page 37

Note that because records used to report on this condition are typically filled in at the time of the system logger SMF reporting interval, conditions that occur between the time of the last SMF interval and the time the check is run may be missed. When ALL is specified, these conditions will appear when the report is run after the next system logger SMF reporting interval. If you update to a TIME(mm/dd/yyyy hh:mm:ss), parameter after the most recent SMF reporting interval, some conditions may never be reported on. You can either accept this loss, set a TIME(mm/dd/yyyy hh:mm:ss) before the most recent SMF interval, or use the ALL parameter to view these conditions when they become available.

z/OS releases the check applies to:

z/OS V1R7 and later.

Parameters accepted:

Yes, the check accepts the following parameters to control the time this check reports conditions from:

PARM('ALL')

The default parameter, ALL, specifies that the check display conditions that happened since system logger was initialized, up to the most recent 16 log streams that have conditions. For each condition, the report lists the log stream name, corresponding structure name, time of last occurrence, and count of occurrences since logger started for each log stream that has a condition.

PARM('TIME(mm/dd/yyyy hh:mm:ss)')

This parameter specified that the check display exceptions happened since the requested time. For each exception condition, the report lists the log stream name, corresponding structure name, and time of last occurrence. Counts will not be shown when this parameter is active. The input time, must be a valid GMT, in the requested format, and can not be a time in the future.

Message IXGH009E or IXGH006I indicates the time the check is reporting from. If the inputted parameters are in an incorrect format message IXGH004I will be shown and the check will be stopped.

User override of IBM values:

The following shows keywords you can use to override check values on either the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMIXGLOGR,IXGLOGR_ENTRYTHRESHOLD) PARM('ALL')
       SEVERITY(LOW) INTERVAL(4:00) DATE(20071106)
       REASON('Logger entry threshold reached conditions
       should be investigated to determine if applications
       performance is being impacted')
```

Use the MODIFY command with the UPDATE parameter to change the IXGLOGR_ENTRYTHRESHOLD check parameters, as follows:

```
F HZSPROC,UPDATE,CHECK(IBMIXGLOGR,IXGLOGR_ENTRYTHRESHOLD)
         ,PARM('ALL')
F HZSPROC,UPDATE,CHECK(IBMIXGLOGR,IXGLOGR_ENTRYTHRESHOLD)
         ,PARM('TIME(10/10/2007 19:45:00)')
```

You can also use system symbols in an HZSPRMxx member to modify a check to the current time. The following example shows how to update the check to only report on current log streams conditions:

1. Update HZSPRMxx to put system symbols in the parameter for IXLOGR_ENTRYTHRESHOLD., as the following example shows:

   ```
   UPDATE CHECK(IBMIXGLOGR,IXGLOGR_ENTRYTHRESHOLD)
          PARM('TIME(&MON/&DAY/&YR4 &HR:&MIN:&SEC)')
   ```

2. Using the following console command to add the updated HZSPRMxx parmlib members to the list of members that IBM Health Checker for z/OS is using:

   ```
   F HZSPROC,ADD,PARMLIB=xx
   ```

Reference:

For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:

This check issues the following exception messages:

- IXGH009E

See the IXGH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:

SYSLOW

Output: The following shows output from IXGLOGR_ENTRYTHRESHOLD:

- Output for the no exceptions case:

```
CHECK(IBMIXGLOGR,IXGLOGR_ ENTRYTHRESHOLD)
START TIME: 10/10/2007 13:29:58.588368
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: ALL

IXGH006I This system has not encountered any structure entry threshold
conditions since 10/10/2007 13:25:30 (GMT).

END TIME: 10/10/2007 13:29:58.648301 STATUS: SUCCESSFUL
```

- Output for the check running with PARM(ALL) and some exceptions found:

```
CHECK(IBMIXGLOGR,IXGLOGR_ENTRYTHRESHOLD)
START TIME: 10/10/2007 16:08:40.099953
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: ALL

* Low Severity Exception *

IXGH009E One of more log streams encountered a structure entry
threshold condition since 10/10/2007 13:25:30 (GMT).
...
                                                 Time of Last
Log Stream                Structure       Count Condition (GMT)
TESTLOG1.HCHECK1.S1       LIST01          1     10/10/2007 19:42:16
TESTLOG1.HCHECK1.S3       LIST02          1     10/10/2007 19:43:02
TESTLOG1.HCHECK1.S2       LIST01          2     10/10/2007 19:43:49
TESTLOG1.HCHECK1.S5       LIST02          1     10/10/2007 19:44:35
TESTLOG1.HCHECK1.S7       LIST01          1     10/10/2007 19:45:22
TESTLOG1.HCHECK1.S6       LIST03          2     10/10/2007 19:46:08
TESTLOG1.HCHECK1.S4       LIST03          1     10/10/2007 19:46:55
TESTLOG1.HCHECK1.S8       LIST03          1     10/10/2007 19:47:41

END TIME: 10/10/2007 16:08:40.102732  STATUS: EXCEPTION-LOW
```

- Output for the check after it is updated to PARM('TIME(10/10/2007 19:45:00)') and run again. The check now only shows exceptions occurring after the time specified:

```
CHECK(IBMIXGLOGR,IXGLOGR_ENTRYTHRESHOLD)
START TIME: 10/10/2007 16:13:05.884363
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: TIME(10/10/2007 19:45:00)

* Low Severity Exception *

IXGH009E One of more log streams encountered a structure entry
threshold condition since 10/10/2007 19:45:00 (GMT).
...
                                                 Time of Last
Log Stream                Structure       Count Condition (GMT)
TESTLOG1.HCHECK1.S7       LIST01          1     10/10/2007 19:45:22
TESTLOG1.HCHECK1.S6       LIST03          2     10/10/2007 19:46:08
TESTLOG1.HCHECK1.S4       LIST03          1     10/10/2007 19:46:55
TESTLOG1.HCHECK1.S8       LIST03          1     10/10/2007 19:47:41

END TIME: 10/10/2007 16:13:05.885974  STATUS: EXCEPTION-LOW
3.3.12.3 IXGLOGR_STRUCTUREFULL
```

IXGLOGR_STRUCTUREFULL

Description:

Reports on log streams that have encountered structure element full conditions. This means that the log stream used all of the elements in its portion of the structure. Additional log streams writes to the structure fail until the full condition is relieved. Applications using the affected log streams may experience a slow down or possibly an outage if the condition is not resolved in a timely fashion. SMF must be active for system logger to report on structure element full conditions when log streams remain connected.

Reason for check:

IBM suggests that tuning actions be taken to avoid structure element full conditions. For more details see message description for IXGH007E.

After conditions are sufficiently corrected before a certain time, set that as a TIME(mm/dd/yyyy hh:mm:ss) parameter in an UPDATE statement. This will suppress conditions older than that time. You can cut and paste the time from the report output.

After updating to TIME(mm/dd/yyyy hh:mm:ss), if you want to see counts for the log stream values, you'll either have to switch to the ALL parameter, or review the information from a SMF report.

We recommend that check output be saved for historical purposes, especially if you perform parameter updates. See the following information for using HZSPRINT to record check output to a data set or log stream:

- "Optionally set up the HZSPRINT utility" on page 12
- "Setting up security for the HZSPRINT utility" on page 16
- "Using the HZSPRINT utility" on page 37

Note that because records used to report on this condition are typically filled in at the time of the system logger SMF reporting interval, conditions that occur between the time of the last SMF interval and the time the check is run may be missed. When ALL is specified, these conditions will appear when the report is run after the next system logger SMF reporting interval. If you update to a TIME(mm/dd/yyyy hh:mm:ss), parameter after the most recent SMF reporting interval, some conditions may never be reported on. You can either accept this loss, set a TIME(mm/dd/yyyy hh:mm:ss) before the most recent SMF interval, or use the ALL parameter to view these conditions when they become available.

z/OS releases the check applies to:
z/OS V1R7 and later.

Parameters accepted:
Yes, this check accepts the following parameters to control the time this check reports exceptions from:

PARM('ALL')
The default parameter, ALL, specifies that the check display exceptions occuring since system logger was initialized, up to the most recent 16 log streams that have conditions. For each exception condition, the report lists the log stream name, corresponding structure name, time of last occurrence, and count of occurrences since logger started for each log stream that has a condition.

PARM('TIME(mm/dd/yyyy hh:mm:ss)')
This parameter specified that the check display exceptions that happened since the requested time. For each exception condition, the report lists the log stream name, corresponding structure name, and time of last occurrence. Counts will not be shown when this parameter is active. The input time, must be a valid GMT, in the requested format, and can not be a time in the future.

In the check output, message IXGH007E or IXGH004I indicates the time the check is reporting from. If the inputted parameters are in an incorrect format, the system issues message IXGH004I and stops the check.

User override of IBM values:
The following shows keywords you can use to override check values on either the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL PARM('ALL')
       SEVERITY(LOW) INTERVAL(4:00) DATE('date_of_the_change')
       REASON('Your reason for making the update.')
```

Use the MODIFY command with the UPDATE parameter to change the IXGLOGR_ENTRYTHRESHOLD check parameters, as follows:

```
F HZSPROC,UPDATE,CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL),PARM('ALL')

F HZSPROC,UPDATE,CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL),PARM('TIME(10/10/2007 19:45:00')
```

You can also use system symbols in an HZSPRMxx member to modify a check to the current time. The following example shows how to update the check to only report on current log streams conditions:

1. Update HZSPRMxx to put system symbols in the parameter for IXLOGR_ENTRYTHRESHOLD., as the following example shows:

   ```
   UPDATE CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL)
        PARM('TIME(&MON;/&DAY;/&YR4; &HR;:&MIN;:&SEC;)')
   ```

2. Using the following console command to add the updated HZSPRMxx parmlib members to the list of members that IBM Health Checker for z/OS is using:

   ```
   F HZSPROC,ADD,PARMLIB=xx
   ```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXGH007E

See the IXGH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

Output: The following shows output from IXGLOGR_STRUCTUREFULL:

- Output for the no exceptions case:

  ```
  CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL)
  START TIME: 10/10/2007 13:29:57.788349
  CHECK DATE: 20060615  CHECK SEVERITY: LOW
  CHECK PARM: ALL

  IXGH004I This system has not encountered any log stream structure
  element full conditions since 10/10/2007 13:25:30 (GMT).
  ```

- Output for the check running with PARM(ALL) and some exceptions found:

  ```
  CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL)
  START TIME: 10/10/2007 16:10:49.286937
  CHECK DATE: 20060615  CHECK SEVERITY: LOW
  CHECK PARM: ALL

  * Low Severity Exception *

  IXGH007E One or more log streams encountered a structure element full
  condition since 10/10/2007 13:25:30 (GMT).
  ...
                                                    Time of Last
  Log Stream                Structure       Count Condition (GMT)
  TESTLOG1.HCHECK1.S1       LIST01          1     10/10/2007 19:42:16
  TESTLOG1.HCHECK1.S3       LIST02          1     10/10/2007 19:43:02
  TESTLOG1.HCHECK1.S2       LIST01          2     10/10/2007 19:43:49
  TESTLOG1.HCHECK1.S5       LIST02          1     10/10/2007 19:44:35
  ```

```
TESTLOG1.HCHECK1.S7        LIST01           1     10/10/2007 19:45:22
TESTLOG1.HCHECK1.S6        LIST03           2     10/10/2007 19:46:08
TESTLOG1.HCHECK1.S4        LIST03           1     10/10/2007 19:46:55
TESTLOG1.HCHECK1.S8        LIST03           1     10/10/2007 19:47:41

END TIME: 10/10/2007 16:10:49.288364  STATUS: EXCEPTION-LOW
```

- Output for the check after it is updated to PARM('TIME(10/10/2007 19:45:00)') and run again. The check now only shows exceptions occurring after the time specified:

```
CHECK(IBMIXGLOGR,IXGLOGR_STRUCTUREFULL)
START TIME: 10/10/2007 16:12:49.140899
CHECK DATE: 20060615  CHECK SEVERITY: LOW
CHECK PARM: TIME(10/10/2007 19:45:00)

* Low Severity Exception *

IXGH007E One or more log streams encountered a structure element full
condition since 10/10/2007 19:45:00 (GMT).
...                                                  Time of Last
Log Stream                 Structure        Count Condition (GMT)
TESTLOG1.HCHECK1.S7        LIST01           1     10/10/2007 19:45:22
TESTLOG1.HCHECK1.S6        LIST03           2     10/10/2007 19:46:08
TESTLOG1.HCHECK1.S4        LIST03           1     10/10/2007 19:46:55
TESTLOG1.HCHECK1.S8        LIST03           1     10/10/2007 19:47:41

END TIME: 10/10/2007 16:12:49.142517  STATUS: EXCEPTION-LOW
```

System trace checks (IBMSYSTRACE)

SYSTRACE_BRANCH

Description:
Checks to see whether system trace is using the BR=ON parameter of the TRACE command and has been active for a longer time than the defined duration.

Reason for check:
A branch trace runs continuously so an active unneeded BRANCH=ON option can cause degraded system performance. Use branch tracing only for short periods of time to solve a specific problem and do not use branch tracing as the default for system tracing on your system. BR=ON is intended for use in system software problem determination and diagnosis situations only.

z/OS releases the check applies to:
z/OS V2R1 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(SYSTRACE_BRANCH) OWNER(IBMSYSTRACE)
ACTIVE
DEBUG(OFF)
INTERVAL(4:00),
SEVERITY(LOW),
```

```
PARM('TIME(DAYS,07)'),
DATE('20110601'),
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
Yes

Parameters accepted:
No.

Reference:
See the following information:

- TRACE command in *z/OS MVS System Commands*.
- Tracing branch instructions in *z/OS MVS Diagnosis: Tools and Service Aids*.

Messages:
This check issues the following exception messages:

- IEAH801E

See the IEAH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

SYSTRACE_MODE

Description:
Checks to see whether system trace is using the MODE=ON parameter of the TRACE command and has been active for a longer time than the defined duration.

Reason for check:
A mode trace runs continuously, so an active unneeded MODE=ON option can cause degraded system performance. Use mode tracing only for short periods of time to solve a specific problem and do not use mode tracing as the default for system tracing on your system. MODE=ON is intended for use in system software problem determination and diagnosis situations only.

z/OS releases the check applies to:
z/OS V2R1 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(SYSTRACE_MODE) OWNER(IBMSYSTRACE)
ACTIVE
DEBUG(OFF)
INTERVAL(4:00),
SEVERITY(LOW),
PARM('TIME(DAYS,07)'),
DATE('20110601'),
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
Yes

Parameters accepted:
No.

Reference:
See the following information:

- TRACE command in *z/OS MVS System Commands*.
- MODE and MOBR trace entries in *z/OS MVS Diagnosis: Tools and Service Aids*.

Messages:
This check issues the following exception messages:

- IEAH804E

See the IEAH messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Timer supervisor checks (IBMTIMER)

ZOSMIGREC_SUP_TIMER_INUSE

Description:
Verify that Server Time Protocol (STP) is in use, when appropriate.

Reason for check:
Server Time Protocol is recommended because the Sysplex Timer (9037-002) has been withdrawn from marketing and STP is planned to be its replacement.

z/OS releases the check applies to:
z/OS V1R11 and later.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMTIMER, ZOSMIGREC_SUP_TIMER_INUSE)
INACTIVE
SEVERITY(LOW) INTERVAL(ONETIME) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For more information on the migration action from the Sysplex Timer to STP, see *z/OS Migration*. To implement STP, see the STP Web site and the

publications and other resources that are listed there. The STP Web site is at http://www.ibm.com/systems/z/pso/stp.html.

Messages:
This check issues the following exception messages:

- IEATH005E

See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for MLS users:
SYSLOW

TSO/E (IBMTSOE)

TSOE_USERLOGS

Description:
This check will report on whether user logs are being used for the receipt of sent messages.

Reason for check:
You should use user logs be used for processing user's messages. If you do not use them, the result can be contention on the system brodcast data set and the possibility of one user tying up the system while the user is accessing user mail.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMTSOE,TSOE_USERLOGS)
  SEVERITY(LOW) INTERVAL(ONETIME) DATE(20070607)
  REASON('User logs should be in use')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See *z/OS TSO/E Customization*

Messages:
This check issues the following exception messages:

- IKJH0202E

See *z/OS TSO/E Messages*.

Messages:
See .

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

TSOE_PARMLIB_ERROR

Description:
This check will report on whether any of the groupings of TSO/E settings failed to be built at IPL due to an error processing the commands in IKJTSOxx. The groups of settings that were defaulted due to any errors will be listed by this check.

Reason for check:
For ease of maintenance and dynamic updates, IBM suggests that TSO/E system wide settings be managed via a PARMLIB member, IKJTSOxx. If there is a syntax error or other error in processing the definitions in the PARMLIB member, TSO/E will default to a set of definitions that may not be desirable for the system being IPLd.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMTSOE,TSOE_PARMLIB_ERROR)
  SEVERITY(LOW) INTERVAL(ONETIME) DATE(20070607)
  REASON('PARMLIB errors may have occurred during IPL')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See *z/OS TSO/E Customization*

Messages:
This check issues the following exception messages:

- IKJH0302E
- IKJH0303E
- IKJH0304E

See *z/OS TSO/E Messages*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

z/OS UNIX System Services checks (IBMUSS)

USS_AUTOMOUNT_DELAY

Description:
Automount delay configuration values in a sysplex are at least 10 minutes

Reason for check:
Each configuration should have a delay time of at least 10 minutes. Anything lower can cause the system to hang, continually trying to unmount file systems and failing. The message will show the automount configured directory, the configuration name and the delay value.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMUSS,USS_AUTOMOUNT_DELAY)
SEVERITY(MED)
INTERVAL(24:00)
PARM('DELAY=10')
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes. Specify either PARM('DELAY=*delay*') or PARM('*delay*'). The default is PARM('DELAY=10')

Reference:
See:

- *z/OS UNIX System Services Planning* for information on using the automount facility.
- *z/OS UNIX System Services Command Reference* for information on the automount command.

Messages:
This check issues the following exception messages:

- BPXH030E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

USS_CLIENT_MOUNTS

Description:
This check will generate an exception when a file system is found that is function shipping but could be mounted locally.

Reason for check:
File systems should not function ship if they can be mounted locally to avoid performance degradation.

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMUSS,USS_CLIENT_MOUNTS)
  SEVERITY(MED) INTERVAL(1:00) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See:

- *z/OS UNIX System Services Planning* .
- *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- BPXH065E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The following shows output from the check:

```
CHECK(IBMUSS,USS_CLIENTS_MOUNTS)
CHECK DATE: 20070809  CHECK SEVERITY: MEDIUM

BPXH003I z/OS UNIX System Services was initialized using OMVS=(DD,DN),
where each 2-character item is a BPXPRMxx suffix.

BPXH063I The following file systems are available through a remote owner
system:
------------------------------------------------------------
File System: MYHFS
Mount Mode:  READ
PFS Type:    HFS


* Medium Severity Exception *

BPXH065E One or more file systems that should be locally mounted are
available through a remote system.

  Explanation:  Check USS_CLIENTS_MOUNTS found one or more file systems
    that are should be locally mounted.  This condition occurs in a
```

```
    shared file system configuration.  The file system was intended to
    be mounted locally but either the local or the owning physical file
    system has become inactive.  The file system is made available
    through a remote mount on the owning system.

  System Action:  The file system is available through the remote system
    for processing.

  Operator Response:  N/A

  System Programmer Response:  The file system should be accessible
    through a local mount.  Determine why it is not and correct the
    situation.  The original mount of the file system may have failed
    because the file system is not accessible from the local system.
    The file system may have been correctly mounted and subsequently
    converted to a remote mount if the physical file system is no longer
    active.
    If the physical file system is TYPE(NFS), make sure that TCPIP is
    operational on this system.

    Otherwise, it may be necessary to unmount the file system and then
    mount it again.

  Problem Determination:  See BPXH063I in the message buffer.

  Source:  z/OS UNIX System Services

  Reference Documentation:
    For information on modifing BPXPRMxx see:
     Customizing z/OS UNIX in z/OS UNIX System Services Planning
     BPXPRMxx in z/OS MVS Initialization and Tuning Reference

    For information on using the DISPLAY OMVS,MF command see:
     DISPLAY in MVS System Command Reference in z/OS MVS System Commands

  Automation:  N/A

  Check Reason:  File systems should not function ship if they can be
    mounted locally.  Performance is not optimal in this situation.

END TIME: 11/14/2007 16:37:17.405073  STATUS: EXCEPTION-MED
```

USS_FILESYS_CONFIG

Description:

Evaluates the file system configuration, which includes:

- AUTOMOVE setup verification
- zFS for a multilevel security configuration.
- Mode (either RDWR/READ) of the root, system specific, and version HFSs.

Reason for check:

The system specific file system should be mode RDWR. The version file system should be mode READ.

Define your version and sysplex root file systems as AUTOMOVE and define your system-specific file systems as UNMOUNT. Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE. If you do, the file system defined as AUTOMOVE will not be available until the failing system is restarted. A sysplex file system that changes ownership as the result of a system failure, will only be accessible in the new environment if its mount point is also accessible. The Automove check verifies that your file systems are set up according to these rules. This check is only applicable for images that are part of a sysplex.

The AUTOMOVE | NOAUTOMOVE | UNMOUNT parameters on ROOT and MOUNT indicate what happens to the file system if the system that owns that file system goes down. The AUTOMOVE parameter specifies that ownership of the file system is automatically moved to another system. It is the default. The NOAUTOMOVE parameter specifies that the file system will not be moved if the owning system goes down and the file system is not accessible. –UNMOUNT specifies that the file system will be unmounted when the system leaves the sysplex.

z/OS releases the check applies to:
z/OS V1R4 and later. On a z/OS V1R4 system, the check does not evaluate zFS for an MLS configuration.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMUSS,USS_FILESYS_CONFIG)
SEVERITY(HI)
INTERVAL(24:00)
PARM('SYSPLEX')
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes:

- PARM('SYSPLEX') - Specifies that the check verify file system configuration. This is the default when SYSPLEX(YES) is specified in the BPXPRMxx parmlib member.
- PARM('NOPLEX') - Verifies the MLS configuration. This is the default when SYSPLEX(NO) is specified in the BPXPRMxx parmlib member.

Note: You may use PARM('NOPLEX') when SYSPLEX(YES) is specified in the BPXPRMxx parmlib member if you only want to verify the MLS configuration.

Reference:
For more information on file systems, see *z/OS UNIX System Services Planning*, and APAR II3129.

Messages:
This check issues the following exception messages:

- BPXH002E
- BPXH007E
- BPXH011E
- BPXH012E
- BPXH014E
- BPXH015E
- BPXH017E
- BPXH018E
- BPXH024E

- BPXH025E
- BPXH028E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

USS_HFS_DETECTED

Description:
The check verifies all file systems mounted and issues an exception message if any HFS file systems are found. The check only looks for HFS file systems that are owned on the system running the health check.

This check runs any time an HFS file system is successfully mounted, unless overridden by the RUN_ON_MOUNT=NO parameter. The check will also run any time `MODIFY BPXOINIT,FILESYS=REINIT` is issued.

The check does not run after the `MODIFY OMVS,NEWROOT=xxx` command is issued.

Reason for check:
HFS file systems are no longer the strategic file system. All HFS file systems should be migrated to zFS.

z/OS releases the check applies to:
z/OS V1R11 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMUSS,USS_HFS_DETECTED)
  SEVERITY(LOW)
  INTERVAL(24:00)
  PARM('RUN_ON_MOUNT=YES')
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes:

- RUN_ON_MOUNT=YES|NO - indicates whether or not the check should run after the successful mount of an HFS file system. RUN_ON_MOUNT=YES is the default.
- HFS_LIST=(*listoffile systems*) - A list of HFS file systems that you wish this check to ignore. The check will not issue exception messages for a file system mounted with that name. Each file system name can be from 1 to 44 characters long. File system names are separated by a comma. You can specify up to either 23 file systems or however many will fit within the 256 character limit for check parameters.

  An example of this parameter might be HFS_LIST=(USS.ROOT.HFS,MYFILE.HFS)

Reference:
For more information on file systems, see Managing the z/OS UNIX file system in *z/OS UNIX System Services Planning*.

Messages:
This check issues the following exception messages:

- BPXH068E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The following shows output from the check:

```
CHECK(IBMUSS,USS_HFS_DETECTED)
START TIME: 05/08/2009 16:06:02.974215
CHECK DATE: 20090427  CHECK SEVERITY: LOW
CHECK PARM: HFS_LIST=(ZOS112.LPP.HFS)


BPXH003I z/OS UNIX System Services was initialized using OMVS=(2W,DN),
where each 2-character item is a BPXPRMxx suffix.

BPXH069I The following HFS file systems were found:
------------------------------------------------------------
ZOS112.NLS.HFS
ZOS112.MAN.HFS

* Low Severity Exception *

BPXH068E One or more HFS file systems mounted.

  Explanation:  The USS_HFS_DETECTED check found one or more active HFS
    file systems on the current system.

  System Action:  The system continues processing.

  Operator Response:  Report this problem to the system programmer.

  System Programmer Response:  HFS file systems are no longer the
    strategic file system.  All HFS file systems should be migrated to
    zFS.

  Problem Determination:  See BPXH069I in the message buffer.

  Source:  z/OS UNIX System Services

  Reference Documentation:  For information on migrating the HFS file
    system to a zFS file system see the chapter on Managing the z/OS
    file system in z/OS UNIX System Services Planning

  Automation:  N/A

  Check Reason:  HFS file systems are no longer the strategic file file
    system.  All HFS file systems should be migrated to zFS.

END TIME: 05/08/2009 16:06:02.985862  STATUS: EXCEPTION-LOW
```

USS_KERNEL_PVTSTG_THRESHOLD

Description:
This check monitors the current usage of UNIX System Services kernel private storage below the bar against a suggested threshold.

Reason for check:
At kernel initialization 80% of the available region is designated for stack cell pool cells and the remaining 20% for kernel and system usage. Because the stack cell pool is only extended as needed all of the storage that is designated for stack cell might not be allocated. If the allocation of storage designated for kernel and system usage overflows into the storage designed for stack cells, the number of stack cells that can be allocated is reduced. Reducing the number of stack cells that can be allocated can reduce the system-wide UNIX workload capacity. This check notifies the installation of the potential reduced capacity.

z/OS releases the check applies to:
z/OS V2R1 and later.

User override of IBM values:
Use the keywords in the following statement to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement can be copied and modified to override the check defaults:

```
UPDATE,
        CHECK(IBMUSS,USS_KERNEL_PVTSTG_THRESHOLD),
        INTERVAL(00:02),
        SEVERITY(HIGH),
        PARM('PVTSTG(85%)'),
        DATE('20131005')
```

Debug support:
No

Verbose support:
No

Parameters accepted:

- PVTSTG(*n*%) where *n* is an integer, 0-100, indicating the threshold percent for usage of kernel private storage below the bar that was available after kernel initialization.
- Default: PVTSTG(85%)

Reference:
None.

Messages:
This check issues the following exception message:

- BPXH072E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information about using security labels.

USS_KERNEL_STACKS_THRESHOLD

Description:
This check monitors the current usage of z/OS UNIX kernel stacks cell pool cells against a suggested threshold.

Reason for check:
If the number of kernel stack cell pool cells in use reaches the system maximum, additional cells cannot be allocated and system calls that use the kernel address space are disallowed. By monitoring stack cell usage

installations might be able to prevent impacts to critical workloads by quiescing noncritical workloads before the supply of stack cells is exhausted.

z/OS releases the check applies to:
z/OS V2R1 and later.

User override of IBM values:
The following statement shows keywords that you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement can be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMUSS,USS_KERNEL_STACKS_THRESHOLD),
INTERVAL(00:02),
SEVERITY(HIGH),
PARM('STACKS(85%)'),
DATE('20131005')
```

Debug support:
No

Verbose support:
No

Parameters accepted:

- STACKS(*n*%) where *n* is an integer, 0-100, indicating the threshold percent for utilization of kernel stack cell pool cells.
- Default: STACKS(85%)

Reference:
See:

- *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- BPXH071E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information about using security labels.

USS_MAXSOCKETS_MAXFILEPROC

Description:
MAXSOCKETS (AF_INET) and MAXFILEPROC are set high enough

Reason for check:
This check will look at the values for MAXSOCKETS and MAXFILEPROC and give an exception message if either is too low. If set too low, you can run out of sockets or file descriptors that can be used. MAXSOCKETS and MAXFILEPROC values will each be compared to 64000 unless the value is overridden in HZSPRMxx.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)
SEVERITY(LOW)
INTERVAL(24:00)
PARM('MAXSOCKETS=64000,MAXFILEPROC=64000')
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes. PARM('MAXSOCKETS=*maxsockets*,MAXFILEPROC=*maxfileproc*')
MAXSOCKETS and MAXFILEPROC are required integer values to be compared with internal values.

- The valid range for MAXSOCKETS is 0 through 16777215.
- The valid range for MAXFILEPROC is 3 through 524287.

The default is PARM(' MAXFILESOCKETS=64000,MAXFILEPROC=64000').

You can also specify these parameters without keywords, as PARM('*maxsockets*,*maxfileproc*').

Reference:
See:

- *z/OS MVS System Commands* for information on the SETOMVS command.
- *z/OS UNIX System Services Planning* for information on how to change the MAXSOCKETS and MAXFILEPROC values using the SETOMVS command.

Messages:
This check issues the following exception messages:

- BPXH032E
- BPXH033E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

USS_PARMLIB

Description:
This check will compare z/OS UNIX System Services current system settings with those specified in the BPXPRMxx parmlib members used during initialization and issue an exception message if a difference is found. For PARM values on the MOUNT statement, the check is case sensitive, detecting and raising an exception for otherwise identical values that are expressed in different cases.

Note that if a dynamic symbolic is contained in the value specification for any BPXPRMxx statement, that statement may be flagged as changed because the current dynamic value is different from the value that was used when the BPXPRMXX statement was originally processed.

If the system finds syntax errors in one or more of the BPXPRMxx parmlib members the check issues message BPXH046E. If syntax errors are found, the check does not make a comparison between the system and the parmlib settings. You must correct all syntax errors before the check will compare the system and the parmlib settings. See the explanation for message BPXH046E in *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for more information.

The z/OS UNIX dynamic settings that will be checked are:

z/OS UNIX dynamic settings checked

AUTHPGMLIST
AUTOCVT
FILESYSTYPE
FORKCOPY
IPCSEMNIDS
IPCSEMNOPS
IPCSEMNSEMS
IPCMSGNIDS
IPCMSGQBYTES
IPCMSGQMNUM
IPCSHMMPAGES
IPCSHMNIDS
IPCSHMNSEGS
IPCSHMSPAGES
LIMMSG
LOSTMSG
MAXASSIZE
MAXCORESIZE
MAXCPUTIME
MAXFILEPROC
MAXFILESIZE
MAXMMAPAREA
MAXPIPEUSER
MAXPROCSYS
MAXPROCUSER
MAXPTYS
MAXQUEUEDSIGS
MAXSHAREPAGES
MAXUIDS
MAXUSERMOUNTSYS
MAXUSERMOUNTUSER
MOUNT FILESYSTEM
NETWORK
NONEMPTYMOUNTPT
PRIORITYGOAL
PRIORITYPG
ROOT FILESYSTEM
SHRLIBMAXPAGES
SHRLIBRGNSIZE
STEPLIBLIST
SUPERUSER
SYSCALL_COUNTS
TTYGROUP
USERIDALIASTABLE
VERSION

- For the FILESYSTYPE statement, the types specified in the BPXPRMxx parmlib members will be compared to what Physical File Systems are currently running.
- For the ROOT/MOUNT FILESYSTEM statements, the following will be checked:
  - Mount point
  - Mode, RDWR for example.
  - Automove setting. Note that AUTOMOVE and system list are both treated as AUTOMOVE.
  - PARM subparameter
- For the NETWORK statement, only the MAXSOCKETS value will be checked for AF_INET and AF_INET6.

Reason for check:
When dynamic changes are made to z/OS UNIX, the BPXPRMxx parmlib members should be updated with the changes so that they will be available the next time z/OS UNIX is initialized.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMUSS,USS_PARMLIB)
  SEVERITY(LOW) INTERVAL(01:00) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See:

- *z/OS UNIX System Services Planning*
- *z/OS MVS Initialization and Tuning Reference*

Messages:
This check issues the following exception messages:

- BPXH040E
- BPXH046E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output:

```
CHECK(IBMUSS,USS_PARMLIB)
START TIME: 03/29/2006 16:09:05.512021
CHECK DATE: 20060112  CHECK SEVERITY: LOW

BPXH003I z/OS UNIX System Services was initialized using OMVS=(DD,DN),
where each 2-character item is a BPXPRMxx suffix.

BPXH041I The following differences were found between the system
settings and the BPXPRMxx parmlib concatenation:


Option               BPXPRMxx Value            System Value
------------------------------------------------------------
MAXFILEPROC                     256                     111
MAXPTYS                         256                     255
MAXCPUTIME                     1000                     999

Physical File Systems not in parmlib
------------------------------------------------------------
AUTOMNT

Changed File Systems
------------------------------------------------------------
File System: ZOS18.ETC.HFS
BPXPRMxx Value:
 Path: etc
 Automove: AUTOMOVE
 Access: RDWR
 Parm: NONE
```

```
System Value:
 Path: etc
 Automove: AUTOMOVE
 Access: READ
 Parm: NONE

File System: MYHFS
BPXPRMxx Value:
 Path: /jkad
 Automove: AUTOMOVE
 Access: RDWR
 Parm: NONE

System Value:
 This file system is currently not mounted.


* Low Severity Exception *

BPXH040E One or more differences were found between the system settings
and the settings in the current BPXPRMxx parmlib concatenation.

  Explanation:  Check USS_PARMLIB detected changes made to either the
    system settings or to the BPXPRMxx parmlib members.

  System Action:  The system continues processing.

  Operator Response:  Report this problem to the system programmer.

  System Programmer Response:  View the message buffer for information
    on what values have changed. Use the DISPLAY OMVS,OPTIONS command
    to view what the current system settings are. The system values can
    be changed dynamically by using the SETOMVS command. If the current
    system values are desired, then a permanent definition should be
    created so the values will be available the next time z/OS UNIX
    System Services is initialized. A permanent definition can be
    created by editing the BPXPRMxx parmlib members to include the
    desired values.

  Problem Determination:  See BPXH041I in the message buffer.

  Source:  z/OS UNIX System Services

  Reference Documentation:  See MVS System Command Reference in z/OS
    MVS System Commands for information on using the DISPLAY
    OMVS,OPTIONS command. See MVS System Command Reference in z/OS MVS
    System Commands and Managing operations, section: Dynamically
    changing the BPXPRMxx parameter values in z/OS UNIX System Services
    Planning for information on using the SETOMVS command. See
    Customizing z/OS UNIX in z/OS UNIX System Services Planning and
    BPXPRMxx in z/OS MVS Initialization and Tuning Reference for
    information on editing the BPXPRMxx parmlib members.

  Automation:  N/A

  Check Reason:  Reconfiguration settings should be kept in a permanent
    location so they are available the next time z/OS UNIX is
    initialized.
```

USS_PARMLIB_MOUNTS

Description:

This check will generate an exception when a file system in ROOT or MOUNT statement specified in the BPXPRMxx parmlib members used during

initialization fails to mount. Mount failures due to duplicate MOUNT statements with different attributes, such as mount point or mode, will not be flagged by this check.

Note that symbolic links in the path will not be resolved in the output for check exception in system message BPXH059I.

The check is rerun automatically after the failing file system is successfully mounted and after issuing the `F BPXOINIT,FILESYS=REINIT` system command. However, USS_PARMLIB_MOUNTS only checks the values that were specified in BPXPRMxx at initialization time. If BPXPRMxx has been updated since then with different values, those values will not be included in any subsequent checks.

Reason for check:
BPXPRMxx parmlib mount failures should be corrected in a timely manner to avoid outages.

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMUSS,USS_PARMLIB_MOUNTS)
  SEVERITY(HIGH) INTERVAL(ONCE) DATE('date_of_the_change')
   REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See:

- *z/OS UNIX System Services Planning* .
- *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- BPXH061E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The following shows output from the check - note that symbolic links in the path are not resolved in the output:

```
CHECK(IBMUSS,USS_PARMLIB_MOUNTS)
START TIME: 11/30/2007 13:23:50.671024
CHECK DATE: 20070809  CHECK SEVERITY: HIGH

BPXH003I z/OS UNIX System Services was initialized using OMVS=(D3,DN),
```

```
where each 2-character item is a BPXPRMxx suffix.

BPXH059I The following file systems are not active:
------------------------------------------------------------
File System: DAN.HFS
 Parmlib Member: BPXPRMDN
 Path: /asd
 Return Code: 00000081
 Reason Code: 1288005C

* High Severity Exception *

BPXH061E One or more file systems specified in the BPXPRMxx parmlib
members are not mounted.

  Explanation:  During the USS_PARMLIB_MOUNTS check, one or more
    file systems that were specified in the BPXPRMxx parmlib members
    used for initialization were found not to be active.

  System Action:  The system continues processing.

  Operator Response:  Report this problem to the system programmer.

  System Programmer Response:  Review the return code and reason code in
    the summary message and determine why the file systems are not
    active. Correct the problem using documented procedures. After the
    problem has been corrected, mount each file system using one of the
    following procedures:
      Ask a superuser to enter the corrected information using the TSO/E
      MOUNT command or the mount shell command. If the statement in
      error was the ROOT statement, specify '/' as the mount point.

      Alternatively, the SET OMVS=(xx) system command can be issued,
      where "xx" is the last two characters of a BPXPRMxx parmlib member
      that contains the MOUNT statement(s) to re-process.

  Problem Determination:  See BPXH059I in the message buffer.

  Source:  z/OS UNIX System Services

  Reference Documentation:
    For information on modifing BPXPRMxx see:
     Customizing z/OS UNIX in z/OS UNIX System Services Planning
     BPXPRMxx in z/OS MVS Initialization and Tuning Reference

    For information on using the DISPLAY OMVS,MF command see:
     DISPLAY in MVS System Command Reference in z/OS MVS System Commands

  Automation:  N/A

  Check Reason:  BPXPRMxx parmlib mount failures can cause outages if
    not handled in a timely manner.

END TIME: 11/30/2007 13:23:50.709436  STATUS: EXCEPTION-HIGH
```

ZOSMIGREC_ROOT_FS_SIZE

Description:

Because of release enhancements and service, the size of the z/OS root file system (or “version root file system”) continues to grow from release to release. As of z/OS V1R10, the size of the z/OS root file system, whether HFS or zFS, was approximately 3100 cylinders on a 3390 Direct Access Storage Device. This is close to the limit of 3339 cylinders on a 3390-3 device, and if not accommodated can halt the installation.

This check examines the volume that the version root file system resides on, and if that volume has an acceptable amount of available cylinders. This check passes if the volume that the version root file system resides on has an amount of available cylinders greater than the minimum required (Defaulted at 500 cylinders). This check fails if the volume that the version root file system resides on has an amount of available cylinders less than the minimum required (Defaulted at 500 cylinders). This check will also fail if the Health Check userid does not have READ access on the OPERCMDS MVS.DISPLAY.OMVS resource.If the version root file system data set is SMS-managed, this check is not applicable.

Reason for check:
Verify size accommodation for the z/OS root file system to prevent halt on installation.

z/OS releases the check applies to:
z/OS V1R9 and later.

Parameters accepted:

MIN_CYLINDERS
The amount of cylinders available. You can specify a value in the range 200-1000000. The default is 500.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMUSS, ZOSMIGREC_ROOT_FS_SIZE)
VERBOSE(NO)
SEVERITY(LOW)
INTERVAL(ONETIME)
INACTIVE
DATE('date_of_the_change')
PARM('MIN_CYLINDERS=500')
REASON('Your reason for making the update.')
```

Debug support:
Yes

Verbose support:
No

Reference:
For more information on the migration action for keeping your z/OS root file system accommodated, see Accommodate changes to support read-only z/OS root for the cron, mail, and uucp utilities in *z/OS Migration*.

Messages:
This check issues the following exception messages:

- BPXH903E
- BPXH904E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*

SECLABEL recommended for MLS users:
SYSLOW

ZOSMIGV1R13_RO_SYMLINKS

Description:
Before z/OS V1R13, certain post-installation activities had to be done for each new release for the cron, mail, and uucp utilities in order for the root file system to be mounted read-only. Starting in z/OS V1R13, the /usr/lib/cron, /usr/mail, and /usr/spool directories are provided as symbolic links. On z/OS V1R11 and z/OS V1R12 this check will indicate whether you will be affected by this migration action in z/OS V1R13.

Reason for check:
Verify whether the z/OS V1R13 changes for cron, mail, and uucp to support a read-only root will affect a system.

z/OS releases the check applies to:
z/OS V1R11 and V1R12.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMUSS, ZOSMIGV1R13_RO_SYMLINK)
VERBOSE(NO)
DEBUG(NO)
SEVERITY(LOW)
INTERVAL(ONETIME)
INACTIVE
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
Yes

Verbose support:
No

Reference:
For more information on the migration action for keeping your z/OS root file system accommodated, see Accommodate changes to support read-only z/OS root for the cron, mail, and uucp utilities in *z/OS Migration*.

Messages:
This check issues the following exception messages:

- BPXH915E

See the BPXH messages in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*

SECLABEL recommended for MLS users:
SYSLOW

VLF checks (IBMVLF)

VLF_MAXVIRT

Description:
Checks to see whether the virtual lookaside facility (VLF) is trimming recently

added objects to make room for new objects. If so, the MAXVIRT setting for at least one VLF class may be too small for VLF to provide a good performance benefit. This check runs once an hour.

Reason for check:
Optimize VLF data space usage.

z/OS releases the check applies to:
z/OS V2R1 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVLF,VLF_MAXVIRT),
INTERVAL(1:00),
SEVERITY(LOW),
PARM('ALERTAGE(*,60)'),
DATE('20110802')
Reason('Your reason for making the update.')
```

Debug support:
No

Verbose support:
Yes

Parameters accepted:
ALERTAGE(*class_name1*,*alert_age1*,*class_name2*,*alert_age2*,...*class_namex*,*alert_agex*)

The ALERTAGE check parameter lets you override the AlertAge values specified in the COFVLFxx parmlib member for their respective VLF classes. The alert age is a value, in seconds, that the check uses to determine whether objects are being trimmed too rapidly to meet the installation's VLF usage goals. If the check determines that objects in the specified classes are being trimmed sooner than the specified alert age value, the check issues exception message COFVLH02E.

The variable for the parameters is as follows:

class_namex
Class names may be one to seven alphanumeric characters including @, #, and $, and may use the standard MVS wildcard characters (* and ?). If multiple class_namex specifications match for the same class, the last matching one in the list will be honored for that class.

If you specify a class name in this parameter that does not match any class name defined in the current COFVLFxx parmlib member, the check ignores this parameter.

alert_agex
alert_agex is a decimal value in the range 0 to 99999999 that indicates an alert age in seconds. If an object younger than this alert age value is trimmed, the VLF_MAXVIRT check issues exception message COFVLH02E.

Default: None - you must specify an alert age to use this check parameter.

A value of 0 indicates that no trimming alert should be issued for this class. Note that higher values make it more likely that an alert will be issued.

The class alert ages specified in the check parameters override the ones specified in the class definitions in the current COFVLFxx parmlib member, and will persist through a VLF stop and restart. If no alert age is specified in either the COFVLFxx parmlib member or check parameters, the check uses the default alert age of 60.

Reference:
For more information see the ALERTAGE parameter in the COFVLFxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- COFVLH02E

See the COFVLH messages in *z/OS MVS System Messages, Vol 4 (CBD-DMO)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSAM checks (IBMVSAM)

VSAM_INDEX_TRAP

Description:
Checks to see if the VSAM index trap is enabled or not. The index trap validates each index record before the system writes it, looking for any corruption in the records.

Reason for check:
IBM recommends running with the index trap enabled because it validates index records before they are written to DASD. If the system detects an error, it bypasses the write, preventing permanent damage to the data set structure. In addition, the index trap captures diagnostic data.

z/OS releases the check applies to:
z/OS V1R4 and later.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSAM,VSAM_INDEX_TRAP)
INACTIVE
SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No.

Reference:
For more information on the VSAM index trap, see:

- *z/OS DFSMSdfp Diagnosis*
- *z/OS DFSMS Using Data Sets*

Messages:
This check issues the following exception messages:

- IDAHC102E

See the IDAHC messages in *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSAM RLS checks (IBMVSAMRLS)

VSAMRLS_CFCACHE_MINIMUM_SIZE

Description:
This check makes sure that all CF caches are above the IBM recommended minimum size. This check identifies CF caches that are below the IBM recommended minimum value for the sysplex.

Reason for check:
The minimum CF cache size should be at least 10% of CF cache optimal size. A CF cache structure must be at least large enough to hold all of the MVS information required to describe a structure of maximum size. CF cache structures must be defined to MVS and also in the SMS base configuration. CF cache structures provide a level of storage hierarchy between local memory and DASD cache. They are also used as a system buffer pool for VSAM RLS data when that data is modified on other systems. Each CF cache structure is contained in a single CF. You might have multiple CFs and multiple CF cache structures. Performance should improve when the CF cache is larger than the sum of the local VSAM LRS buffer pool sizes. When the CF cache is smaller, performance depends upon the dynamics of the data references among the systems involved.

z/OS releases the check applies to:
z/OS V1R9 and up.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSAMRLS,VSAMRLS_CFCACHE_MINIMUM_SIZE)
SEVERITY(MED) INTERVAL(24:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For more information, see the section on "Defining CF Cache Structures" in *z/OS DFSMSdfp Storage Administration*.

Messages:
This check issues the following exception messages:

- None

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

VSAMRLS_CFLS_FALSE_CONTENTION

Description:
The check looks for false contention on your system caused by a lock structure that is not big enough.

Reason for check:
If the lock structure size is too small, the system could experience excessive false contention, resulting in performance degradation. The acceptable false contention threshold is 5%. If the system experiences performance degradation, the lock structure size should be increased.

z/OS releases the check applies to:
z/OS V1R9 and up.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSAMRLS,VSAMRLS_CFLS_FALSE_CONTENTION)
SEVERITY(MED) INTERVAL(01:00) DATE('date_of_the_change')
PARMS('THRESHOLD(5000)')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes:

PARMS('THRESHOLD(*threshold*)')

Specify an integer for the acceptable false contention threshold you want the check to use. This THRESHOLD value is specified in thousandths of a percent. The default value is PARM('THRESHOLD(5000)') which sets the false contention rate at which the health check issues an error to 5%.

.

Reference:
For more information, see the section on "Defining the CF Lock Structure" in *z/OS DFSMSdfp Storage Administration*.

Messages:
This check issues the following exception messages:

- IGWRH0131E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

VSAMRLS_DIAG_CONTENTION

Description:
Checks for VSAM RLS contention by looking at registered resources. Check displays a contention table if detected.

Reason for check:
IBM recommends monitoring VSAM RLS contention.

z/OS releases the check applies to:
z/OS V1R8 and up.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMVSAMRLS,VSAMRLS_DIAG_CONTENTION)
       SEVERITY(HI)
       INTERVAL(00:05)
       DATE('date_of_the_change')
       PARMS('ROWS(20),FILTER(0)')
       REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes:

PARM(ROWS)

Specify an integer, minimum 1, indicating the maximum number of rows to be displayed in the contention table.

PARM(FILTER)
Specify an integer from 0 to 3600 to filter out rows in the contention table with an elapsed time of 0-3600 seconds, leaving only the rows which have a TIME value (in seconds) greater than or equal to the FILTER value specified.

Note that the system looks at the ROWS parameter before the FILTER parameter, so the FILTER value will only be in effect on the number of rows specified in the ROWS parameter. So, if there were 500 rows and the

ROWS value is 200, it will show at most 200 rows in the contention table.A value of 0 disables the filter, and a value of 1 filters out all rows with an elapsed time of 0 seconds.

Reference:
For additional information see VSAM RLS Latch Contention in *z/OS DFSMSdfp Diagnosis*.

Messages:
This check issues the following exception messages:

- IGWRH0102E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSAMRLS_QUIESCE_STATUS

Description:
The check looks for unresponsive CICS® regions for QUIESCE and UNQUIESCE, which could indicate a problem.

Reason for check:
IBM recommends monitoring QUIESCE and UNQUIESCE events for unresponsive CICS regions.

z/OS releases the check applies to:
z/OS V1R9 and up.

Parameters accepted:
Yes.

PARM(THRESHOLD)
Specify an integer from 0 to 2147483647 to filter out rows in the quiesce status table with an elapsed time less than that specified for the value of THRESHOLD (in seconds).

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMVSAMRLS,VSAMRLS_QUIESCE_STATUS)
SEVERITY(MED)
INTERVAL(00:05)
DATE('date_of_the_change')
PARMS('THRESHOLD(300)')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For additional information, see the following commands in *z/OS MVS System Commands*:

- VARY SMS,SMSVSAM,SPHERE(*sphere*)
- DISPLAY SMS,SMSVSAM,QUIESCE

Messages:
This check issues the following exception messages:

- IGWRH0402E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

VSAMRLS_SHCDS_CONSISTENCY

Description:
The check is looking to see if all SHCDSs are allocated with consistent allocation amounts.

Reason for check:

Allocate SHCDSs with consistent allocation amounts and use consistent values for primary allocation and secondary allocation for each SHCDS. Consistent allocation optimizes space utilization, while inconsistent allocation amounts waste space.

When the SHCDS with the smallest allocation starts to runs out of space, all SHCDSs are extended with their secondary quantity. SHCDS with large secondary quantities may extend unnecessarily. VSAM RLS expects identical allocation amount for all SHCDSs.

z/OS releases the check applies to:
z/OS V1R9 and up.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSAMRLS,VSAMRLS_SHCDS_CONSISTENCY)
SEVERITY(MED) INTERVAL(01:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For more information, see the section on "Defining Sharing Control Data Sets" in *z/OS DFSMSdfp Storage Administration*.

Messages:
This check issues the following exception messages:

- IGWRH0141E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

VSAMRLS_SHCDS_MINIMUM_SIZE

Description:
The check detects SHCDS that are not big enough on your system.

Reason for check:
If the SHCDS is too small, the system may potentially experience problems, such as performance degradation. The initial size of the SHCDS needs to be at least 13 MB. A larger size should be used if there are more than 6 systems in the sysplex.

z/OS releases the check applies to:
z/OS V1R9 and up.

Parameters accepted:
Yes

PARM(NUMOFRDS)
Specify an integer from 1 to 3000 inclusive to indicate the average number of recoverable data sets open across the sysplex.

PARM(NUMOFRSS)
Specify an integer from 1 to 100 inclusive to indicate the average number of recoverable subsystems running across the sysplex.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSAMRLS,VSAMRLS_SHCDS_MINIMUM_SIZE)
SEVERITY(LOW) INTERVAL(24:00) DATE('date_of_the_change')
PARMS('NUMOFRDS(100),NUMOFRSS(10)')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For more information, see the section on "Defining Sharing Control Data Sets" in *z/OS DFSMSdfp Storage Administration*.

Messages:
This check issues the following exception messages:

- IGWRH0151E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

VSAMRLS_SINGLE_POINT_FAILURE

Description:
Verifies that the SHCDSs are on unique volumes.

Reason for check:
To avoid single points of failure, IBM suggests that you allocate SHCDS for a system on unique volumes.

z/OS releases the check applies to:
z/OS V1R8 and up.

Type of check (local or remote):
Local

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMVSAMRLS,VSAMRLS_SINGLE_POINT_FAILURE)
SEVERITY(HI)
INTERVAL(24:00)
DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No.

Reference:
For additional information see Defining Sharing Control Data Sets in *z/OS DFSMSdfp Storage Administration*

Messages:
This check issues the following exception messages:

- IGWRH202E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSAMRLS_TVS_ENABLED

Description:
Verify that DFSMStvs is enabled.

Reason for check:
Enable DFSMStvs if it is installed but not enabled.

z/OS releases the check applies to:
z/OS V1R9 and up.

Parameters accepted:
No

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSAMRLS, VSAMRLS_TVS_ENABLED)
SEVERITY(LOW) INTERVAL(04:00) DATE('date_of_the_change')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For more information, see the section on "Implementing DFSMStvs" in *z/OS DFSMStvs Planning and Operating Guide*.

Messages:
This check issues the following exception messages:

- IGWRH301E
- IGWRH302E

See the IGWRH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

VSM checks (IBMVSM)

The storage configuration is established during system initialization, based on system parameters, the size of the modules in LPA, and the nucleus. Your storage configuration can change when the system is IPLed even if system parameters have not changed.

IBM Health Checker for z/OS provides several checks and diagnostic reports to detect when the storage configuration has changed or may need to be changed. Information regarding the storage configuration is saved for comparison with prior IPLs.

The VSM checks include different storage reports, including reports showing IPL parameters, size and location of CSA, SQA, LPA and the nucleus, current common storage allocation, and the five highest users of common storage (available when storage tracking is active). These reports are generated along with the check output, as appropriate.

VSM_ALLOWUSERKEYCSA

Description:
This check examines the setting of the ALLOWUSERKEYCSA(YES|NO) DIAGxx option and compares it to the IBM recommended setting of ALLOWUSERKEYCSA(NO). A warning is issued if the setting is YES.

Reason for check:
Allowing programs to obtain user key CSA creates a security risk because CSA storage can then be modified by any unauthorized program. IBM recommends that ALLOWERUSERKEYCSA(NO) be coded in the active DIAGxx parmlib member.

Note: Coding ALLOWUSERKEYCSA(NO) for this option will cause user key programs attempting to obtain CSA storage to ABEND with abend code B78, reason code xxxxxx5C. (The first three bytes of the reason code provide internal failure details.) The default setting for this option is ALLOWUSERKEYCSA(NO).

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
No.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_ALLOWUSERKEYCSA),
ACTIVE,
INTERVAL(ONETIME),
SEVERITY(LOW),
DATE('20060201'),
```

Reference:
No

Messages:
This check issues the following exception messages:

- IGVH110E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_CSA_LARGEST_FREE

Description:
Monitor the current size of the largest contiguous free block of CSA/ECSA against the installation-specified or default minimum value.

Reason for check:
If the system is unable to satisfy storage-obtain requests for large blocks of common storage due to issues like storage fragmentation, a system outage may occur. This check provides an advanced warning about such common storage problem so the system programmers can take appropriate action.

z/OS releases the check applies to:
z/OS V2R1

Parameters accepted:
Yes, the following parameters are accepted:

- When not using dynamic severity: CSA(csabytes | csan%),ECSA(ecsabytes | ecsan%)
- When using dynamic severity:
  - CSA_HIGH(csabytes | csan%), CSA_MED(csabytes | csan%), CSA_LOW(csabytes | csan%), CSA_NONE(csabytes | csan%)
  - ECSA_HIGH(ecsabytes | ecsan%), ECSA_MED(ecsabytes | ecsan%), ECSA_LOW(ecsabytes | ecsan%), ECSA_NONE(ecsabytes | ecsan%)

Note: When using dynamic severity, you may specify thresholds for 1 or more of the parameters to identify different thresholds by severity level. Note that you do not need to specify thresholds for all of the parameters.

csan%
An integer, 0-100, followed by a percent sign '%', indicating the minimum size of the largest contiguous free block of CSA required on the system.

csabytes
A size in bytes, with an optional suffix (K,M), indicating the minimum size of the largest contiguous free block of CSA required on the system.

ecsan%
An integer, 0-100, followed by a percent sign '%', indicating the minimum size of the largest contiguous free block of ECSA required on the system.

ecsabytes
A size in bytes, with an optional suffix (K,M), indicating the minimum size of the largest contiguous free block of ECSA required on the system.

Default:. CSA(5%),ECSA(5%)

You can use synonyms for these parameters, as follows:

Table 53. Synonyms of Parameters

Parameter	Synonyms
CSA_HIGH	CSA_HI CSA_H
CSA_MED	CSA_M
CSA_LOW	CSA_L
CSA_NONE	CSA_NO CSA_N
ECSA_HIGH	ECSA_HI ECSA_H
ECSA_MED	ECSA_M
ECSA_LOW	ECSA_L
ECSA_NONE	ECSA_NO ECSA_N

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMVSM,VSM_CSA_LARGEST_FREE)
INTERVAL(00&colon.15)
SEVERITY(HIGH)
PARM('CSA(5%),ECSA(5%)')
DATE('20120101')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGVH111I
- IGVH112E
- IGVH504I

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_CSA_LIMIT

Description:
The current size of CSA against a minimum suggested value.

Reason for check:
The size of CSA should be adequate to meet the needs of the applications that run on your system. It can be established explicitly by the operator during system initialization. It can also be specified in the system parameter list (IEASYSxx), specified by the operator response SYSP=xx, or the default IEASYS00. The size of CSA can be greater than or less than the requested size because it is affected by other system areas that change when a new IPL occurs. For example, an increase in the size of SQA or LPA modules that must be loaded in storage below 16 megabytes can reduce the size of CSA or cause CSA to be allocated at a lower address. When the allocation of CSA crosses a 1-megabyte segment, the size of private storage is also changed. The size of LPA can cause less storage to be available in private and CSA. This should be considered when moving modules to LPA. System performance improves when the search order for important applications is appropriate and adequate storage is available. Whenever possible and when you would not compromise available virtual storage, you should use dynamic LPA to place frequently used modules in LPA. Make sure you do not inadvertently duplicate modules, module names, or aliases that already exist in LPA. Fixed LPA and fixed storage should be reserved for modules that should always be paged in because this reduces the available central storage on the system.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

Parameters accepted:
Yes:

- Number of bytes with optional suffix (K,M) indicating the minimum amount of below the line CSA required on the system (keyword: CSA)
- Number of bytes with optional suffix (K,M) indicating the minimum amount of above the line CSA required on the system (keyword: ECSA)
- Default: CSA(512K),ECSA(512K)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_CSA_LIMIT),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('CSA(512K),ECSA(512K)'),
DATE('date_of_the_change')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGVH101E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_SQA_LIMIT

Description:
The current size of SQA against a minimum suggested value.

Reason for check:
The total amount of virtual storage and number of private virtual storage address spaces affect the system's use of SQA. Like CSA, SQA size is determined by the operator or the system parameter list. When SQA falls below a threshold, a critical storage message is issued, new jobs cannot be created, and address spaces cannot be swapped in until the shortage is alleviated. If the size allocated for extended SQA is too small or is used up very quickly, the system attempts to use extended CSA. When both extended SQA and extended CSA are used up, the system allocates space from SQA and CSA below 16 megabytes. The allocation of this storage could eventually lead to a system failure.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

Parameters accepted:

- Number of bytes with optional suffix (K,M) indicating the minimum amount of below the line SQA required on the system (keyword: SQA)
- Number of bytes with optional suffix (K,M) indicating the minimum amount of above the line SQA required on the system (keyword: ESQA)
- Default: SQA(512K),ESQA(8M)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_SQA_LIMIT),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('SQA(512K),ESQA(8M)'),
DATE('date_of_the_change')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGVH101E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_PVT_LIMIT

Description:
Whether the size of private storage is adequate to meet the needs of applications that run on your system.

Reason for check:
The total amount of private virtual storage available to applications on the system is a direct result of the size of CSA and SQA, as well as LPA, MLPA, and FLPA. Changes to the size of any of these areas may impact the amount of virtual storage remaining to satisfy private storage requests.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

Parameters accepted:

- Number of bytes with optional suffix (K,M) indicating the minimum amount of below the line PVT required on the system (keyword: PVT)
- Number of bytes with optional suffix (K,M) indicating the minimum amount of above the line PVT required on the system (keyword: EPVT)
- Default:PVT(1M),EPVT(512M)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_PVT_LIMIT),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('PVT(1M),EPVT(512M)'),
DATE('date_of_the_change')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGVH101E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_CSA_THRESHOLD

Description:
The current allocation of CSA storage.

Reason for check:
When the current allocation has reached the user-specified or IBM-specified threshold value, an exception message and storage reports are created. The threshold report includes a comparison to high-water marks on the last IPL as well as the amount of the current allocation. The high-water mark is the highest amount of storage allocated since the system was IPLed.

If the threshold is specified as a percentage value (the default), an exception will be issued when the allocation of storage exceeds that threshold. If the

threshold is given as a size in bytes, an exception will be issued when the amount of storage remaining is less than the specified size.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

Verbose support:
Yes. At z/OS V1R11 and up, system owned storage is evaluated for possible inclusion in the "Five High Users Report" when the check is run in verbose mode. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the SDSF CK command display.

Parameters accepted:
Yes, as follows:

- When not using dynamic severity: CSA(*csabytes* | *csa*%), ECSA(*ecsabytes* | *ecsan*%)
- When using dynamic severity:
  - CSA_HIGH(*csabytes* | *csan*%), CSA_MED(*csabytes* | *csan*%), CSA_LOW(*csabytes* | *csan*%), CSA_NONE(*csabytes* | *csann*%)
  - ECSA_HIGH(*ecsabytes* | *ecsan*%), ECSA_MED(*ecsabytes* | *ecsan*%), ECSA_LOW(*ecsabytes* | *ecsan*%), ECSA_NONE(ecsabytes | ecsan%)

  Note that when specifying percentages for parameter values, the values should increase as the severity increases, since the percentage specifies a percent full amount (a larger value indicates a more severe condition), as shown in the following example:

  ```
  PARM('CSA_HIGH(95%),CSA_MED(80%),CSA_LOW(60%), ECSA_HIGH(90%),ECSA_MED(70%)')
  ```

  However, if you specify bytes for the parameter values, the values should decrease as the severity increases, since the values specify a bytes-remaining amount (a smaller value indicates a more severe condition), as shown in the following example:

  ```
  PARM('CSA_HIGH(64K),CSA_MED(256K),ECSA_HIGH(128K), ECSA_LOW(1M)')
  ```

csan%
: An integer, 0-10000 followed by %, indicating the threshold percent for utilization of CSA below the line.

csabytes
: A size in bytes, with an optional suffix (K,M) indicating the minimum amount of unallocated CSA.

ecsan%
: An integer, 0-10000 followed by %, indicating the threshold percent for utilization of ECSA above the line.

ecsabytes
: A size in bytes, with an optional suffix (K,M) indicating the minimum amount of unallocated ECSA.

Default: CSA(80%),ECSA(80%)

You can use synonyms for these parameters, as follows:

Parameter	Synonyms
CSA_HIGH	• CSA_HI • CSA_H
CSA_MED	• CSA_M
CSA_LOW	• CSA_L
CSA_NONE	• CSA_NO • CSA_N
ECSA_HIGH	• ECSA_HI • ECSA_H
ECSA_MED	• ECSA_M
ECSA_LOW	• ECSA_L
ECSA_NONE	• ECSA_NO • ECSA_N

Note that when specifying percentages for parameter values, your number for the HIGH cases is higher than the MED cases, as shown in the following example:

```
PARM('CSA_HIGH(95%),CSA_MED(80%),CSA_LOW(60%),ECSA_HIGH(90%),ECSA_MED(70%)')
```

On the other hand, if you specify bytes for the parameter values the values for HIGH cases are lower than those for MED cases, as shown below:

```
PARM('CSA_HIGH(64K),CSA_MED(256K),ECSA_HIGH(128K),ECSA_LOW(1M)')
```

User override of IBM values:

The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_CSA_THRESHOLD),
INTERVAL(00:05),
SEVERITY(HIGH),
PARM('CSA(80%),ECSA(80%)'),
DATE('date_of_the_change')
```

Reference:

For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:

This check issues the following exception messages:

- IGVH100E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:

SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Examples of message output for the VSM_CSA_THRESHOLD check when (E)SQA has overflowed into (E)CSA:

```
IGVH100E ECSA has exceeded the threshold percentage of 80%
Current allocation is 88% of 41252K.
Unallocated amount is 4684K bytes.
```

```
Explanation:  The current allocation of ECSA storage is 88% of
41252K.
This allocation exceeds the installation threshold.
4684K bytes or 11% is still available.

5001 CSA/ECSA pages were converted to SQA/ESQA.

The highest allocation during this IPL is unknown.
...
...
IGVH100I The current allocation of ECSA storage is 16565K of the
total size of 41252K. (The additional allocation of 5001 pages
of ECSA storage to (E)SQA, puts current allocation at 88%)  .
Ensuring an appropriate amount of storage is available is
critical to the long term operation of the system.  An exception
will be issued when the allocated size of ECSA is greater than
the installation specified threshold of 90%.
```

VSM_SQA_THRESHOLD

Description:

The current allocation of SQA storage.

Reason for check:

When the current allocation has reached the user-specified or IBM-specified threshold value, an exception message and storage reports are created. The threshold report includes a comparison to high-water marks on the last IPL as well as the amount of the current allocation. The high-water mark is the highest amount of storage allocated since the system was IPLed.

If the threshold is specified as a percentage value (the default), an exception will be issued when the allocation of storage exceeds that threshold. If the threshold is given as a size in bytes, an exception will be issued when the amount of storage remaining is less than the specified size.

z/OS releases the check applies to:

z/OS V1R4 and later, in both ESA and z/Architecture modes.

Parameters accepted:

Yes, as follows:

- When not using dynamic severity: SQA(*sqabytes* | *sqa*%), ESQA(*esqabytes* | *esqa*%)
- When using dynamic severity:
  - SQA_HIGH(*sqabytes* | *sqan*%), SQA_MED(*sqabytes* | *sqan*%), SQA_LOW(*sqabytes* | *sqan*%), SQA_NONE(*sqabytes* | *sqann*%)
  - ESQA_HIGH(*esqabytes* | *esqa*%), ESQA_MED(*esqabytes* | *esqa*%), ESQA_LOW(*esqabytes* | *esqa*%), ESQA_NONE(*esqabytes* | *esqa*%)

Note that when specifying percentages for parameter values, the values should increase as the severity increases since the percentage specifies a percent full amount (a larger value indicates a more severe condition), as shown in the following example:

```
PARM('SQA_HIGH(95%),SQA_MED(80%),SQA_LOW(60%), ESQA_HIGH(90%),ESQA_MED(70%)')
```

However, if you specify bytes for the parameter values, the values should decrease as the severity increases, since the values specify a bytes-remaining amount (a smaller value indicates a more severe condition), as shown in the following example:

```
PARM('SQA_HIGH(64K),SQA_MED(256K),ESQA_HIGH(128K), ESQA_LOW(1M)')
```

sqan%
An integer, 0-10000 followed by %, indicating the threshold percent for utilization of SQA below the line.

sqabytes
A size in bytes, with an optional suffix (K,M) indicating the minimum amount of unallocated SQA.

esqan%
An integer, 0-10000 followed by %, indicating the threshold percent for utilization of ESQA above the line.

esqabytes
A size in bytes, with optional suffix (K,M) indicating the minimum amount of unallocated ESQA.

Default: SQA(80%),ESQA(80%)

You can use synonyms for these parameters, as follows:

Parameter	Synonyms
SQA_HIGH	• SQA_HI • SQA_H
SQA_MED	• SQA_M
SQA_LOW	• SQA_L
SQA_NONE	• SQA_NO • SQA_N
ESQA_HIGH	• ESQA_HI • ESQA_H
ESQA_MED	• ESQA_M
ESQA_LOW	• ESQA_L
ESQA_NONE	• ESQA_NO • ESQA_N

Note that when specifying percentages for parameter values, your number for the HIGH cases is higher than the MED cases, as shown in the following example:

```
PARM('SQA_HIGH(95%),SQA_MED(80%),SQA_LOW(60%),ECSA_HIGH(90%),ECSA_MED(70%)')
```

On the other hand, if you specify bytes for the parameter values the values for HIGH cases are lower than those for MED cases, as shown below:

```
PARM('SQA_HIGH(64K),SQA_MED(256K),ECSA_HIGH(128K),ECSA_LOW(1M)')
```

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_SQA_THRESHOLD),
INTERVAL(00:15),
SEVERITY(MED),
PARM('SQA(80%),ESQA(80%)'),
DATE('date_of_the_change')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGVH100E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_CSA_CHANGE

Description:
Changes in the size of CSA or private (including the extended areas) since the last IPL.

Reason for check:
The values provided by IBM are 1 megabyte and 10 megabytes for storage below the line and storage above the line, respectively. These value are helpful in determining when the module growth in LPA or the nucleus could reduce the size of the private area.

z/OS releases the check applies to:
z/OS V1R4 and later, in both ESA and z/Architecture modes.

Parameters accepted:

- Number of bytes with optional suffix (K,M) indicating the threshold at which changes in CSA or private below the line will result in an exception being issued. (keyword: BELOW)
- Number of bytes with optional suffix (K,M) indicating the threshold at which changes in CSA or private above the line will result in an exception being issued. (keyword: ABOVE)
- Default: BELOW(1M),ABOVE(10M)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE,
CHECK(IBMVSM,VSM_CSA_CHANGE),
INTERVAL(ONETIME),
SEVERITY(HIGH),
PARM('BELOW(1M),ABOVE(10M)'),
DATE('date_of_the_change')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
This check issues the following exception messages:

- IGVH102E

See the IGVH messages in *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

XCF checks (IBMXCF)

XCF_CDS_MAXSYSTEM

Description:
Provide a warning when a function CDS (any CDS other than the sysplex CDS) is formatted with a MAXSYSTEM value that is less than the MAXSYSTEM value associated with the primary sysplex CDS.

This is a sysplex-wide check (GLOBAL) , which means that it runs on one system but reports on sysplex-wide values and practices. A global check shows up as disabled for all systems in the sysplex, except for the one where it is actually running.

Reason for check:
It is recommended that each couple data set defined to the sysplex be formatted with a MAXSYSTEM value that is at least equal to the value defined in the primary sysplex CDS.

If a function CDS has a smaller MAXSYSTEM value, then a system joining the sysplex with a higher slot number will not be able to use the function provided by that function CDS.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CDS_MAXSYSTEM)
               SEVERITY(MED) INTERVAL(ONETIME) DATE (20090707)
               REASON('CDS MAXSYSTEM value across all CDS types should be at least equal to
                       the value in the primary sysplex CDS')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0401E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CDS_SEPARATION

Description:
Checks that Sysplex couple data set and function couple data sets are properly isolated.

Reason for check:
This check verifies that the following best-practice recommendations are implemented:

- The Sysplex and CFRM primary couple data sets reside on different volumes.
- The LOGR primary CDS resides on a volume separate from the Sysplex and CFRM primaries, if in the view of the installation the rate of I/O activity to the LOGR CDS warrants it.
- Each primary couple data set resides on a different volume than its corresponding alternate couple data set.

z/OS releases the check applies to:
z/OS V1R4 and later.

Verbose support:
Yes. At z/OS V1R10 and up, a CDS configuration report is included when the check is run in verbose mode. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the CK panel in SDSF.

Parameters accepted:
At z/OS V1R10 and up, or V1R8 and up with OA22931 installed:

LOGR(NO | YES)
Indicates whether the system logger (LOGR) couple data set (CDS) is to be checked for separation from other performance-sensitive CDS types.

NO The check will not test whether the primary LOGR CDS is separated from other performance-sensitive CDS types.

YES
The check should verify that the primary LOGR CDS resides on a volume separate from other performance-sensitive CDS types.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CDS_SEPARATION)
       SEVERITY(HI) INTERVAL(001:00) DATE(20080104)
       PARM('LOGR(NO)')
       REASON('Ensure that CDS separation has been maintained.')
       VERBOSE(NO)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0240E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CDS_SPOF

Description:
Check that couple data sets are configured without single points of failure.

Reason for check:
It is recommended that each primary couple data set reside on a different volume from its corresponding alternate couple data set. The I/O configuration should not create any *hidden* single points of failure, for example, placing the volumes containing both primary and alternate of a given type in a single physical device or behind the same switch.

z/OS releases the check applies to:
z/OS V1R10 and later.

Verbose support:
Yes. A CDS configuration report is included when the check is run in verbose mode. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the CK panel in SDSF.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CDS_SPOF)
       SEVERITY(HIGH) INTERVAL(001:00) DATE(20070730)
       REASON('Ensure that couple data sets are configured'
              'without single points of failure.')
       VERBOSE(NO)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0242E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_ALLOCATION_PERMITTED

Description:
Check that, for each coupling facility in the CFRM active policy, structure allocation is permitted.

Reason for check:
If coupling facilities are not eligible for structure allocation, they should not remain in maintenance mode or any other state for any extended period of time. If the check shows a warning during maintenance or upgrade windows, the warning serves as a reminder to turn off maintenance mode when the service action is completed.

Parameters accepted:
None.

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_ALLOCATION_PERMITTED)
         SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0212E
- IXCH0215E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_CONNECTIVITY

Description:
Checks that the system has connectivity to each coupling facility, that multiple links (a/k/a CHPIDs or CFLINKs) to each coupling facility are both ONLINE and OPERATING, and identify single points of failure.

Reason for check:
To avoid single points of failure it is recommended that a system have connectivity to each coupling facility and that there are multiple links that are both ONLINE and OPERATIONAL.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_CONNECTIVITY)
         SEVERITY(MED) INTERVAL(001:00) DATE('date_of_the_change')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0446E
- IXCH0448E
- IXCH0450E

- IXCH0453E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_MEMORY_UTILIZATION

Description:
The check raises an exception when a coupling facility reaches the memory utilization percentage threshold defined for the check. Efficient and planned coupling facility memory utilization prevents a CF from becoming over-full and thereby allows the CF to allocate new structures, expand structures, sustain a viable failover environment and participate in structure rebuild and reallocation processing when needed. The memory utilization threshold percentage will be accepted as a parameter to the check. Note that XCF, independent from this check, will automatically contract structures to relieve coupling facility resource constraints when a coupling facility as a whole is at or above 90% memory utilization and coupling facility resources are not being used productively by the structures.

This is a local check.

Reason for check:
The percentage of memory use in a coupling facility should not approach an amount high enough to keep it from allocating of new structures or expanding existing structures.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
MAXUTILIZATION is a required parameter indicating the threshold percentage that the Coupling Facility memory utilization should not exceed. The number must be an integer in the range of 1 to 99. Specifying a percent (%) sign is optional (e.g. MAXUTILIZATION(60%)).

It is possible that system-initiated alter processing for a structure may start and increase the memory utilization percentage for a coupling facility before the check executes or raises an exception. This check may issue an exception during reconfiguration actions or during maintenance windows when the percentage of memory use exceeds the specified check parameter.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_MEMORY_UTILIZATION)
       SEVERITY(MED) INTERVAL(001:00) DATE (20090707)
       PARM('MAXUTILIZATION(60)')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0456E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_PROCESSORS

Description:
This check provides a warning when a coupling facility processor configuration is not consistent with IBM recommendations and may result in degraded response time and throughput possible for coupling facility requests as compared to coupling facilities configured for the best performance and throughput based on the coupling facility architected function level (CFLEVEL).

This check provides a parameter to enable the installation to specify names of CFs whose processor configurations should be excluded from the determination of the overall status of the check. The installation may wish to exclude non-production CFs or CFs that are running in 'test' mode with shared processors.

This check does not raise an EXCEPTION for a coupling facility that is enabled to use coupling thin interrupts. Coupling facilites at CFLEVEL 19 or above are capable of exploiting coupling thin interrupts.

Because CF processors can not be configured as dedicated processors in a VM environment, this check is **not** applicable when a z/OS image is running as a guest under VM and is disabled from running.

This is a local check.

Reason for check:
CF performance (as measured in service time) is degraded with shared CF central processors.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
EXCLUDE is a required parameter where you can specify a list of CFNames that the check should not consider in its verification processing. A CF named in the EXCLUDE list indicates that the check should not include the processor configuration for that CF in determining the overall check status. Processor configurations for excluded CFs will be reported on in message IXCH0912I and the report will indicate that the check results for the excluded CFNAME were not factored into the overall check status.

Specifiying EXCLUDE() will exclude no coupling facilities from the check verification process.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_PROCESSORS)
       SEVERITY(MED) INTERVAL(004:00) DATE (20090707)
       PARM('EXCLUDE(CFname,CFname)')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0444E
- IXCH0912I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_SCM_UTILIZATION

Description:
Provide a dynamic severity check exception when a CF exceeds one of the specified storage-class memory (SCM) utilization thresholds. The check informs an installation when CF SCM has reached certain usage thresholds and is thereby reaching a point when it might be unable to provide additional coupling facility capacity when needed during peak processing periods as well as provide relief when CF real storage capacity becomes constrained.

Reason for check:
The percentage of SCM utilization in a coupling facility should not approach an amount so high as to prevent the capability to provide relief for temporary CF real storage capacity constraints or additional structure capacity when needed during peak processing.

z/OS releases the check applies to:
z/OS V2R1 and later, or z/OS V1R13 with PTF for APAR OA40747 applied.

Parameters accepted:

```
PARM('[SCM_NONE(scm_none)]
     [,SCM_LOW(scm_low)]
     [,SCM_MED(scm_med)]
     [,SCM_HIGH(scm_high)]')
```

At least one threshold parameter is required to indicate a threshold percentage that the coupling facility storage-class memory utilization should not exceed. The number must be between 0 and 100 inclusive. This check supports dynamic severity setting and threshold keywords to correspond with the severity levels. The severity of the exception is based on the provided corresponding thresholds.

Note:

1. When using dynamic severity, you may specify thresholds for one or more of the parameters to identify different thresholds by severity level.
2. You do not need to specify thresholds for all of the parameters.

scm_none
The percentage of SCM in use by all structures in the CF relative to the CF's total configured SCM that will trigger severity-none processing. The SCM_NONE keyword can be abbreviated SCM_N.

scm_low
The percentage of SCM in use by all structures in the CF relative to the CF's total configured SCM that will trigger a low-severity exception. The SCM_LOW keyword can be abbreviated SCM_L.

scm_med
The percentage of SCM in use by all structures in the CF relative to the

CF's total configured SCM that will trigger a medium-severity exception. The SCM_MED keyword can be abbreviated SCM_M.

scm_high
The percentage of SCM in use by all structures in the CF relative to the CF's total configured SCM that will trigger a high-severity exception. The SCM_HIGH keyword can be abbreviated SCM_H.

Default: SCM_MED(80%)

Note: When specifying percentages for parameter values, your number for the higher severity case is larger than the lower case, as shown in the following example:

```
PARM('SCM_HIGH(90%),SCM_MED(50%),SCM_LOW(20%)')
```

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_SCM_UTILIZATION)
SEVERITY(MED) INTERVAL(001:00) DATE(20120322)
PARM('SCM_MED(80)')
REASON('Coupling facility storage-class memory should be'
       'available to provide relief for coupling facility'
       'real storage constraints.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0457I
- IXCH0458E
- IXCH0924I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_AVAILABILITY

Description:
Check that, for each structure defined in the CFRM active policy, the preference list has at least two coupling facilities located in different CECs and usable for structure allocation. To be usable for structure allocation, the coupling facility must have at least one system connected and have allocation permitted. When the structure has a policy change pending, the preference list from the pending policy is used for making this check.

Reason for check:
The preference list for each structure defined in the CFRM active policy should have at least two coupling facilities located in different CECs that support structure allocation. To support structure allocation, the coupling facility should have at least one system in the sysplex connected and be in a state permitting structure allocation.

Parameters accepted:
None.

z/OS releases the check applies to:
z/OS V1R10 and later.

Verbose support:
Yes. All structures defined in the CFRM active policy are included in the availability report when the check is run in verbose mode. Structures with availability problems are listed ahead of structures without availability problems. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the CK panel in SDSF.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_AVAILABILITY)
  SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
  REASON('Your reason for making the update.')
       VERBOSE(NO)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0212E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_DUPLEX

Description:
Check that, for each structure in the CFRM active policy that is currently allocated, the structure is actually duplexed when DUPLEX(ALLOWED) or DUPLEX(ENABLED) is specified.

Reason for check:
Structures should be duplexed whenever the CFRM active policy says they can be. If a particular structure is not duplexed, it might be an oversight that leaves the structure with less redundancy or recoverability than what was intended.

Parameters accepted:
None.

z/OS releases the check applies to:
z/OS V1R10 and later.

Verbose support:
Yes. All allocated structures with DUPLEX(ALLOWED) or DUPLEX(ENABLED) specified are included in the duplex report when the check is run in verbose mode. Structures that are not duplexed are listed ahead of duplexed structures. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the SDSF CK command display.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_DUPLEX)
        SEVERITY(MED) INTERVAL(001:00) DATE(20070707)
        REASON('Allocated structures with DUPLEX(ALLOWED) or'
               'DUPLEX(ENABLED) specified in the CFRM active'
               'policy should be duplexed.' )
        VERBOSE(NO)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0210E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_EXCLLIST

Description:
Check that each structure is excluded from all structures coded in its exclusion list.

Reason for check:
It is recommended that structure placement is in accordance with its exclusion list.

Parameters accepted:
None.

z/OS releases the check applies to:
z/OS V1R4 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_EXCLLIST)
        SEVERITY(MED) INTERVAL(008:00) DATE('date_of_the_change')
        REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0207E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_MAXSCM

Description:
Indicate a low severity check exception when the sum of the assigned storage-class memory (SCM) for structures allocated in a coupling facility (CF) exceeds the total SCM configured to the CF. This could be an indicator of a possible over- commitment of SCM by the coupling facility.

Reason for check:
It is recommended that the sum of the storage-class memory eligible to be assigned to coupling facility structures should not exceed the total SCM configuration for the coupling facility.

z/OS releases the check applies to:
z/OS V2R1 and later, or z/OS V1R13 with PTF for APAR OA40747 applied.

Parameters accepted:
None.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_MAXSCM)
SEVERITY(LOW) INTERVAL(008:00) DATE(20120322)
REASON('Storage-Class Memory resources eligible for'
       'use by structures should not exceed the'
       'availability of the resources to the CF.')
VERBOSE(YES)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0259E
- IXCH0260I
- IXCH0927I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_MAXSPACE

Description:
Indicate a low severity check exception when the sum of the maximum structure sizes and estimated augmented space values plus the total dump space exceeds the total space (TS) for the coupling facility (CF). This is an indicator of an "over commitment" of real storage by the CF.

Reason for check:
It is recommended that the sum of the storage resources allocated from CF real storage frames and eligible to be assigned to structures should not exceed the actual availability of the resources to the coupling facility at any time.

z/OS releases the check applies to:
z/OS V2R1 and later, or z/OS V1R13 with PTF for APAR OA40747 applied.

Parameters accepted:
None.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_MAXSPACE)
       SEVERITY(LOW) INTERVAL(008:00) DATE(20120322)
       Constant('CF real storage eligible for use by
                 structures should not exceed the
                 total space available to the CF')
       VERBOSE(YES)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0261E
- IXCH0262I
- IXCH0928I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_NONVOLATILE

Description:
Check active allocated structures in the CFRM active policy to determine whether a connector request for non-volatility and failure isolation from connectors is satisfied.

Reason for check:
IBM suggests that coupling facility structure non-volatility and failure isolation from connectors be provided when requested. Non-volatility and failure isolation from connectors may be provided by a coupling facility that is both non-volatile and failure isolated from connectors, or structure duplexing may be used to provide the same result.

Parameters accepted:
None.

z/OS releases the check applies to:
z/OS V1R10 and later.

Verbose support:
Yes. All connected structures are included in the nonvolatility report when the check is run in verbose mode. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the SDSF CK command display.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_NONVOLATILE)
       SEVERITY(MED) INTERVAL(008:00) DATE(20070707)
       REASON('Coupling facility structure non-volatility and'
              'failure isolation from connectors should be'
              'provided when requested.')
       VERBOSE(NO)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0222E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_POLICYSIZE

Description:
Check that structures in the CFRM active policy do not have too large a difference between the value specified for INITSIZE and the value specified for SIZE. All specifications of INITSIZE in the active or pending CFRM policy should indicate an initial structure size of at least half the maximum structure size (as determined by the SIZE specification). The policy should not specify an initial structure size less than the maximum structure size when altering of the structure size is not supported (as determined by this check).

This is a sysplex-wide check (GLOBAL) , which means that it runs on one system but reports on sysplex-wide values and practices. A global check shows up as disabled for all systems in the sysplex, except for the one where it is actually running.

Reason for check:
Specifying different INITSIZE and SIZE values provides flexibility to dynamically expand the size of a structure for workload changes, but too large a difference between INITSIZE and SIZE may waste coupling facility space or prevent structure allocation.

When allocating the structure initially, whether INITSIZE is specified or not, the system attempts to build all control structures that will be required to support the maximum size of the structure. These control structures are built in the control storage allocation of the structure. For structures whose users do not allow structure alter, the control storage allocated to accommodate larger sizes is wasted. An INITSIZE value substantially smaller than the SIZE value might cause the following:

- It might be impossible to allocate a structure at a size of INITSIZE, because the amount of control storage required to support the SIZE value might actually be larger than INITSIZE.
- If the allocation succeeds, it might result in a structure with a proportionally large amount of its storage allotted to structure controls, leaving too few structure objects to be exploited usefully by the associated application.

IBM suggests that the INITSIZE and SIZE specification for structures be determined by the CfSizer (Coupling Facility Structure Sizer) tool:

```
http://www.ibm.com/systems/support/z/cfsizer/
```

Any INITSIZE specification for a structure should be at least half of the SIZE specification. If structure alter is not allowed by users of a structure, INITSIZE should not be specified for that structure.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_POLICYSIZE)
       SEVERITY(MED) INTERVAL(004:00) DATE (20090707)
       REASON('Too large a difference between INITSIZE and SIZE may
               waste coupling facility space or prevent structure
               allocation.')
       VERBOSE(NO)
```

Verbose support:
Yes. All structures in the CFRM active policy are included in an IXCH0923I report when the check is run in verbose mode. A check can be put into verbose mode using the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the SDSF CK command display.

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0255E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_PREFLIST

Description:
Check that each structure is allocated according to the preference list in the CFRM active policy.

Reason for check:
It is recommended that structure placement is in accordance with the preference list.

z/OS releases the check applies to:
z/OS V1R7 and later.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_PREFLIST)
         SEVERITY(MED) INTERVAL(008:00) DATE('date_of_the_change')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_SCM_AUGMENTED

Description:
Check for CF structures that have residual in-use augmented space after all structure objects are removed from storage-class memory (SCM).

Reason for check:
Residual augmented space prevents alter processing from dynamically adjusting CF structure storage usage.

z/OS releases the check applies to:
z/OS V2R1 and later, or z/OS V1R13 with PTF for APAR OA40747 applied.

Parameters accepted:
None.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_SCM_AUGMENTED)
     SEVERITY(LOW) INTERVAL(004:00) DATE(20130211)
     PARM('')
     VERBOSE(NO)
     REASON('Residual augmented space prevents alter processing from '
     'dynamically adjusting CF structure storage usage.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0263E
- IXCH0264I
- IXCH0929I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_SCMMAXSIZE

Description:
Indicate a medium severity check exception when a CFRM policy structure definition specifies SCMMAXSIZE (requests that the structure be eligible to use storage-class memory), but the actual amount of storage-class memory (SCM) assigned to the structure by the CF is less than the specified SCMMAXSIZE.

Reason for check:
It is expected that the actual amount of SCM available to an allocated structure be equal to the CFRM policy SCMMAXSIZE value. Environmental conditions such as the total SCM configured to a CF may limit the amount of SCM that a structure has available to use.

z/OS releases the check applies to:
z/OS V2R1 and later, or z/OS V1R13 with PTF for APAR OA40747 applied.

Parameters accepted:
None.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_SCMMAXSIZE)
SEVERITY(MED) INTERVAL(008:00) DATE(20120322)
REASON('Storage-Class Memory (SCM) available to'
       'an allocated structure should be equal to'
       'the structure defined SCMMAXSIZE.')
VERBOSE(YES)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0258I
- IXCH0257E
- IXCH0926I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_STR_SCM_MINCOUNTS

Description:
Check that the number of structure objects allocated to a structure meets the required minimum for structures that can be duplexed according to the CFRM active policy.

A minimum number of structure objects is required to allocate a structure with a non-zero SCMMAXSIZE. But once allocated, the coupling facility may initiate reapportionment in order to perform migration from SCM (storage-class memory). Such reapportionment may cause a structure object count to go below the minimum.

Reason for check:
When allocating a new structure instance to establish duplexing for a structure that does not currently meet the minimum for a non-zero SCMMAXSIZE, the

new structure instance will be allocated with a zero SCMMAXSIZE - a value that is inconsistent with the CFRM policy SCMMAXSIZE value.

z/OS releases the check applies to:
z/OS V1R13 and later with APAR OA40747.

Parameters accepted:
None.

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_SCM_MINCOUNTS)
     SEVERITY(LOW) INTERVAL(004:00) DATE(20131007)
     PARM('')
     VERBOSE(NO)
     REASON(
     'Less objects than the minimum required causes secondary '
     'structure allocation for duplexing to be inconsistent '
     'with the CFRM active policy SCMMAXSIZE.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0265E
- IXCH0266I
- IXCH0930I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CR_STR_SCM_UTILIZATION

Description:
Provide a dynamic severity check exception when a structure exceeds one of the specified storage-class memory (SCM) utilization thresholds. The check informs an installation when structure assigned SCM has reached certain usage thresholds. The assigned SCM may become exhausted thereby becoming unable to provide continuous availability, additional structure capacity when needed during peak processing periods and provide relief when CF real storage allocated to the structure becomes constrained. The threshold percentages will be accepted as a check PARM.

Reason for check:
The percentage of SCM utilization for a structure should not approach an amount so high as to prevent the capability to provide relief for temporary CF real storage capacity or additional structure capacity when needed during peak processing.

z/OS releases the check applies to:
z/OS V2R1 and later, or z/OS V1R13 with PTF for APAR OA40747 applied.

Parameters accepted:

```
PARM('[SCM_NONE(scm_none)]
     [,SCM_LOW(scm_low)]
     [,SCM_MED(scm_med)]
     [,SCM_HIGH(scm_high)]')
```

At least one threshold parameter is required to indicate a threshold percentage that the structure storage-class memory utilization should not exceed. The number must be between 0 and 100 inclusive. This check supports dynamic severity setting and threshold keywords to correspond with the severity levels. The severity of the exception is based on the provided corresponding thresholds.

Note:

1. When using dynamic severity, you may specify thresholds for one or more of the parameters to identify different thresholds by severity level.
2. You do not need to specify thresholds for all of the parameters.

scm_none
The percentage of in-use SCM, relative to the amount of SCM eligible for use by the structure, that will trigger severity-none processing. The SCM_NONE keyword can be abbreviated SCM_N.

scm_low
The percentage of in-use SCM, relative to the amount of SCM eligible for use by the structure, that will trigger a low severity exceptions. The SCM_LOW keyword can be abbreviated SCM_L.

scm_med
The percentage of in-use SCM, relative to the amount of SCM eligible for use by the structure, that will trigger a medium-severity exception. The SCM_MED keyword can be abbreviated SCM_M.

scm_high
The percentage of in-use SCM, relative to the amount of SCM eligible for use by the structure, that will trigger a high-severity exception. The SCM_HIGH keyword can be abbreviated SCM_H.

Default: SCM_LOW(1%),SCM_MED(80%)

Note: When specifying percentages for parameter values, your number for the higher severity case is larger than the lower case, as shown in the following example:

```
PARM('SCM_HIGH(90%),SCM_MED(50%),SCM_LOW(20%)')
```

User override of IBM values:
The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_STR_SCM_UTILIZATION)
SEVERITY(MED) INTERVAL(000:15) DATE(20120322)
PARM('SCM_LOW(1),SCM_MED(80)')
REASON('Structure assigned storage-class memory should be'
       'available for additional structure capacity'
       'when needed during peak processing.')
VERBOSE(YES)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0224I
- IXCH0225E
- IXCH0925I

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CF_SYSPLEX_CONNECTIVITY

Description:
Check that the required number of coupling facilities are defined in the CFRM policy and connected to all active systems in the sysplex.

Reason for check:
When running in a Parallel Sysplex environment, IBM recommends that hardware redundancy be provided for coupling facilities (that is, at least two usable coupling facilities in the configuration) and that those coupling facilities be connected to all active systems in the sysplex.

z/OS releases the check applies to:
z/OS V1R10 and later.

Verbose support:
Yes. All coupling facilities defined in the CFRM active policy are included in the sysplex connectivity report when the check is run in verbose mode. You can remove a check from verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=NO parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the YES value with the NO value in the VERBOSE column of the SDSF CK command display.

Parameters accepted:
MINCFS(mincfs) is a required parameter indicating the number (in decimal) of coupling facilities that must be connected to all active systems in the sysplex. The number must be between 1 and 16 (inclusive).

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CF_SYSPLEX_CONNECTIVITY)
       SEVERITY(MED) INTERVAL(001:00) DATE('date_of_the_change')
       PARM('MINCFS(2)')
       REASON('Your reason for making the update.')
       VERBOSE(YES)
```

Reference:
For more information, see:

- *z/OS MVS Setting Up a Sysplex*
- Achieving the Highest Levels of Parallel Sysplex Availability (http://www.redbooks.ibm.com/redbooks/SG246061)

Messages:
This check issues the following exception messages:

- IXCH0220E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CFRM_MSGBASED

Description:
Checks to see if CFRM is enabled to use the specified event processing protocol when a structure is defined in the CFRM active policy in a multisystem-capable sysplex.

This is a sysplex-wide check (GLOBAL), which means that it runs on one system but reports on sysplex-wide values and practices. A global check shows up as disabled for all systems in the sysplex, except for the one where it is actually running.

Reason for check:
CFRM message-based event management improves recovery time for users of CF structures. The CFRM message-based event management protocol was introduced in z/OS V1R8 and can be used for CFRM event management of structures other than XCF-signaling.

IBM suggests that, once all systems are at z/OS V1R8 and there is no intention of falling back to a lower level of z/OS, message-based event processing should be enabled because of the large performance, availability, and scalability benefits that it can provide in some Parallel Sysplex environments.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
{MSGBASED | POLBASED}

- MSGBASED - specifies that the check should issue an exception if the sysplex is not using message-based CFRM processing.
- POLBASED - specifies that the check should issue an exception if the sysplex is not using policy-based CFRM processing.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CFRM_MSGBASED)
       SEVERITY(MED) INTERVAL(004:00) DATE (20090707)
       PARM('MSGBASED')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0253E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CLEANUP_VALUE

Description:
Check that the XCF cleanup interval is set to a reasonable value to hasten the removal of a failed system from the sysplex. Cleanup interval is the maximum number of seconds allowed for members of a group to clean up their processing before the system is put into a wait state.

Reason for check:
It is recommended that the XCF cleanup time be set to 15 seconds.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
The recommended XCF cleanup time in seconds. (must be an integer in the range of 0 to 86400)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_CLEANUP_VALUE)
         SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
         PARM('15')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_DEFAULT_MAXMSG

Description:
For each path check that there is a MAXMSG of at least the indicated minimum value specified by or inherited from the COUPLExx, transport class definition, or path definition.

Reason for check:
It is recommended for availability that there is a minimum MAXMSG value of 2000 for each transport class.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
The minimum MAXMSG value for transport classes. (must be an integer in the range of 1 to 999999

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_DEFAULT_MAXMSG)
         SEVERITY(LOW) INTERVAL(024:00) DATE('date_of_the_change')
         PARM('2000')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0427E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_FDI

Description:
The Failure Detection Interval (FDI) is the amount of time that a system can appear to be unresponsive before XCF will take action to remove the system from the sysplex. Internally we refer to this as the effective FDI, externally it is often designated by the word INTERVAL (referring to the INTERVAL parameter in COUPLExx and on the SETXCF COUPLE command). Check that the effective FDI meets the following requirement:

```
SpinFDI <= EffectiveFDI <= MULT*SpinFDI + INC
```

Reason for check:
It is recommended that the user let the system default the effective FDI to the SpinFDI by not specifying the INTERVAL keyword. The INTERVAL keword allows customers to specify an effective FDI that is larger than the Spin FDI. When specified, the INTERVAL value should be at least as large as the SpinFDI to give the system enough time to resolve a spin loop timeout before it gets removed from the sysplex but no so large that the rest of the sysplex suffers sympathy sickness. See message IXCH0510E for a detailed discussion of best practice for FDI.

z/OS releases the check applies to:
z/OS V1R4 and later. Note, however, that the check has been updated for z/OS V1R11.

Parameters accepted:

1. MULT (must be an integer in the range of 1 to 9)
2. INC (must be an integer in the range of 0 to 86400)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_FDI)
SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
PARM('MULT(2),INC(5)')
REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0510E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_MAXMSG_NUMBUF_RATIO

Description:
Check each inbound signal path and ensure that each can support at least the indicated minimum number of messages from the sending system.

Reason for check:
It is recommended that each inbound signal path have enough buffer space to allow at least 30 messages to be received simultaneously.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
The minimum number of XCF messages that an inbound XCF signal path should support to avoid message backup. (must be an integer in the range 1 to 999999)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_MAXMSG_NUMBUF_RATIO)
         SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
         PARM('30')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0443E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SFM_ACTIVE

Description:
The check validates the recommended settings in the Sysplex Failure Management (SFM) policy and reports an exception when the actual SFM status is not consistent with the recommended status.

Reason for check:
It is recommended that:

- Sysplex Failure Management (SFM) be active to handle failure conditions in a sysplex with little or no operator involvement.
- The policy specifies one of the automatic responses (isolation, deactivation, or reset) rather than prompt for all systems.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
Recommended SFM status (must be either ACTIVE or INACTIVE).

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SFM_ACTIVE)
     SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
     PARM('ACTIVE')
     REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0514E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SFM_CFSTRHANGTIME

Description:
This check monitors the current setting of the CFSTRHANGTIME value in the sysplex failure management (SFM) policy to make sure it has not changed from the setting most desirable for your installation. It does this by comparing the SFM CFSTRHANGTIME value with the check parameter CFSTRHANGTIME value that you specify, issuing an exception message if the two are not consistent. The CFSTRHANGTIME SFM parameter specifies the amount of time you are willing to wait before the system takes automatic action to relieve hangs caused when a connector fails to respond to structure-related events in a timely manner.

Reason for check:
Installation should specify SFM policy parameter CFSTRHANGTIME so that the system automatically takes action to relieve hangs in CF structure-related processes caused by a connector's failure to respond to structure-related events in a timely manner. Note that the CFSTRHANGTIME SFM policy attribute applies to any system using SFM, whether or not SFM is in use throughout the sysplex.

The IBM default value for CFSTRHANGTIME is 900 seconds (15 minutes). However, installations must ensure that the check parameters reflect an appropriate value. Consider CFSTRHANGTIME carefully to arrive at a value that reflects the needs, goals, and prior experiences for a given configuration.

In general, it is desirable to select a fairly aggressive value to limit sympathy sickness when connectors fail to respond to events in a timely manner, but this must be balanced against the possibility of terminating connectors prematurely.

z/OS releases the check applies to:
z/OS V1R12 and later.

Parameters accepted:
Yes. PARM('CFSTRHANGTIME(NO | *seconds*)'):

NO Indicates that the system should not take automatic action to relieve a hang in a structure-related process.

seconds
Specifies the time interval, in seconds, that a coupling facility structure connector can remain unresponsive before the system takes action to relieve a hang in a structure-related process.

900 is the default value for this check parameter.

This check generates an exception if the check parameter you specify is not consistent with the existing SFM policy CFSTRANGTIME parameter. For instance, the check generates an exception if:

- The SFM policy specifies or defaults to CFSTRHANGTIME(NO), but the check parameter specifies a value other than NO.
- The SFM policy specifies a CFSTRHANGTIME decimal number of seconds, but the check parameter is either NO or a value less than the value specified in the SFM policy.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SFM_CFSTRHANGTIME)
       SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
       PARM('CFSTRHANGTIME(900)')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0531E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SFM_CONNFAIL

Description:
Check that the sysplex failure management (SFM) policy specifies the recommended action to be taken on loss of signaling connectivity.

Reason for check:
It is recommended that the CONNFAIL attribute of sysplex failure management (SFM) be specified to allow SFM to reconfigure the sysplex when one or more systems loses signalling connectivity.

Parameters accepted:
The CONNFAIL(YES|NO) keyword (must be YES in non-GDPS environment; must be NO in a GDPS environment).

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SFM_CONNFAIL)
       SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
       PARM('CONNFAIL(YES)')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0519E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SFM_SSUMLIMIT

Description:
Check that the sysplex failure management (SFM) policy specifies the recommended action to be taken when a system becomes status update missing but continues to send signals.

Reason for check:
It is recommended that the SFM SSUMLIMIT attribute be specified to allow time to investigate the reason for the status update missing and to take action to correct the condition. The IBM default value is 900 seconds (15 minutes). However, it is the installation's responsibility to ensure that the check parameters reflect an appropriate value. Consider SSUMLIMIT carefully to arrive at a value that reflects the needs, goals, and prior experiences for a given configuration. If the goal is to remove systems quickly when status update missing occurs, 60 seconds is a good value.

Parameters accepted:
SSUMLIMIT setting must be 'NONE' or a decimal value between 0 and 86400 inclusive.

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SFM_SSUMLIMIT)
       SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
       PARM('SSUMLIMIT(900)')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0522E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SFM_SUM_ACTION

Description:
Check that the sysplex failure management (SFM) policy specifies the recommended indeterminate status actions for the local system.

Reason for check:
It is recommended that ISOLATETIME(0) be specified to allow SFM to fence and partition a system without operator intervention and without undue delay.

Parameters accepted:

SUMACTION
SFM Indeterminate status action. Must be one of ISOLATE, RESET, DEACTIVATE, or PROMPT.

SUMINTERVAL
SFM indeterminate status interval. Decimal value between 0 and 86400 seconds inclusive. Not required if ACTION(PROMPT) is specified.

z/OS releases the check applies to:
z/OS V1R10 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SFM_SUM_ACTION)
       SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
       PARM('SUMACTION(ISOLATE),SUMINTERVAL(0)')
       REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0516E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SIG_CONNECTIVITY

Description:
Check that multiple pathin/pathout pairs are in the working state for each system in the sysplex connected to the current system.

Reason for check:
For availability It is recommended for availability that at least 2 pathin/pathout pairs are in the working state for each system in the sysplex connected to the current system.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
The minimum number of pathin/pathout pair counts. (must be an integer in the range of 1 to 99999)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SIG_CONNECTIVITY)
         SEVERITY(MED) INTERVAL(000:30) DATE('date_of_the_change')
         PARM('2')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0414E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SIG_PATH_SEPARATION

Description:
Check for single points of failure for paths to all connected systems.

Reason for check:
It is recommended that there be no single points of failure for the XCF signaling paths between any pair of systems in the sysplex.

z/OS releases the check applies to:
z/OS V1R7 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SIG_PATH_SEPARATION)
       SEVERITY(MED) INTERVAL(000:30) DATE(20070707)
       REASON('XCF signaling connections should have no'
              'single point of failure.')
```

Parameters:
None.

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0443E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SIG_STR_SIZE

Description:
Check that there are enough lists and list entries in every signaling structure to support full signaling connectivity among all systems in the sysplex.

Reason for check:
The Coupling Facility Structure Sizer Tool (CFSizer) should be used to size XCF signaling structures. CFSizer is available on the web at:

`http://www.ibm.com/systems/support/z/cfsizer/`

To calculate the size of the signaling list structure, CFSizer uses the specified system count (which should match the MAXSYSTEM value formatted in the primary sysplex couple data set) and the CLASSLEN for the transport class (to which the signaling structure is to be assigned).

z/OS releases the check applies to:
z/OS V1R4 and up.

Verbose support:
Yes. All connected structures are included in the structure size report when the check is run in verbose mode. You can put a check into verbose mode using any of the following methods:

- Specify the UPDATE,filters,VERBOSE=YES parameter either on the MODIFY command or in a POLICY statement in an HZSPRMxx parmlib member.
- Overwrite the NO value with the YES value in the VERBOSE column of the SDSF CK command display.

Parameters accepted:
The parameters are only supported on z/OS V1R10 and up:

SYSTEMS(ACTIVE | MAXSYSTEM)
Specifies the target sysplex system count required for signaling support, which will be used to calculate the number of signaling paths each structure must support.

ACTIVE
Checks that all signaling structures in use by XCF are large enough to support the number of active systems in the sysplex. This is the default value.

MAXSYSTEM
check that all signaling structures in use by XCF are large enough to support the maximum number of systems that can be in the sysplex. MAXSYSTEM resolves to the value specified when the primary sysplex couple data set was formatted by the IXCL1DSU utility.

User override of IBM values:
The following shows the default keywords for the check, which you can

override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SIG_STR_SIZE)
       SEVERITY(MED) INTERVAL(002:00) DATE(20071101)
       PARM('SYSTEMS(ACTIVE)')
       REASON('XCF signaling structures should be of'
              'sufficient size to support all systems in the'
              'target sysplex.')
       VERBOSE(NO)
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex* and *z/OS MVS System Commands*.

Messages:
This check issues the following exception messages:

- IXCH0247E
- IXCH0248E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SYSPLEX_CDS_CAPACITY

Description:
Check that the maximum number of systems, groups, and members have not at some time reached a threshold determined by the best practice amount of space required for growth of systems, groups, and members.

Reason for check:
It is recommended that the sysplex couple dataset is formatted large enough to allow for the growth of 1 system, 2 groups and 5 members.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:

1. Recommended growth space for systems. (must be an integer in the range of 0 to 32)
2. Recommended growth space for groups. (must be an integer in the range of 0 to 2045)
3. Recommended growth space for members. (must be an integer in the range of 0 to 2047)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SYSPLEX_CDS_CAPACITY)
         SEVERITY(MED) INTERVAL(000:30) DATE('date_of_the_change')
         PARM('1,2,5')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0602E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SYSSTATDET_PARTITIONING

Description:
Checks that the system status detection (SSD) partitioning protocol is enabled by the user. If not enabled, reports the environmental factors that prevent this system using the SSD protocol. Factors checked include:

- Availability of sysplex couple data sets formatted to support the protocol
- The setting of installation controls governing the use of the protocol

The setting of installation controls governing the use of the protocol

Reason for check:
It is recommended that the SSD partitioning protocol be enabled to ensure that failed systems are removed from the sysplex expeditiously and with a minimum of operator involvement.

Parameters accepted:
ENABLED(YES | NO)

ENABLED(YES), the default, will cause the check to return an exception when the SSD protocol is disabled.

z/OS releases the check applies to:
z/OS V1R11 and up

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_SYSSTATDET_PARTITIONING)
SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
PARM('ENABLED(YES)')
REASON('Your reason for making the update.')
```

Debug support:
No

Verbose support:
No

Reference:
For a discussion of the partitioning process and the SSD partitioning protocol, see*z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0526E
- IXCH529E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_TCLASS_CLASSLEN

Description:
Check that there are at least a certain number of different transport classes with unique class lengths defined. This check is appropriate for both monoplex and sysplex mode configurations, although it will return more useful results in sysplex mode.

Reason for check:
It is recommended that there are at least 2 different transport classes with unique class lengths defined.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
Minimum number of different transport classes with unique class lengths. (must be an integer in the range of 1 to 17). Use a parameter setting of 1 for monoplex mode.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_TCLASS_CLASSLEN)
        SEVERITY(MED) INTERVAL(024:00) DATE('date_of_the_change')
        PARM('2')
        REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0424E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_TCLASS_CONNECTIVITY

Description:
Check that all defined transport classes are assigned at least to the indicated number of pathouts (outbound paths). This check is appropriate for both monoplex and sysplex mode configurations.

Reason for check:
It is recommended that all defined transport classes have at least 1 pathout assigned to the class per target system.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
Minimum number of operational signalling paths for a transport class. (must be an integer in the range of 1 to 99999)

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_TCLASS_CONNECTIVITY)
         SEVERITY(MED) INTERVAL(004:00) DATE('date_of_the_change')
         PARM('1')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0420E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_TCLASS_HAS_UNDESIG

Description:
Check that all transport classes are set up to service the pseudo-group name 'UNDESIG'. This ensures that any XCF message can use each transport class. This check is appropriate for both monoplex and sysplex mode configurations.

Reason for check:
It is recommended that all transport classes are set up to service the pseudo-group name 'UNDESIG'.

z/OS releases the check applies to:
z/OS V1R4 and later.

Parameters accepted:
None.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE CHECK(IBMXCF,XCF_TCLASS_HAS_UNDESIG)
         SEVERITY(LOW) INTERVAL(024:00) DATE('date_of_the_change')
         REASON('Your reason for making the update.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
This check issues the following exception messages:

- IXCH0417E
- IXCH0206E

See the IXCH messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

z/OS File System checks (IBMZFS)

ZOSMIGV1R11_ZFS_INTERFACELEVEL

Description:
Verifies that the system is running sysplex_admin_level=2 for zFS V1R11 toleration support.

Reason for check:
zFS should be running at sysplex_admin_level=2 for all members of the sysplex. Once this is done, zFS V1R11 may be brought into the sysplex.

z/OS releases the check applies to:
z/OS V1R9 and V1R10

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMZFS,ZOSMIGV1R11_ZFS_INTERFACELEVEL)
  SEVERITY(LOW)
  INTERVAL(ONETIME)
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For additional information about zFS toleration support, see *z/OS Migration*.

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- IOEZH0011E

See *z/OS Distributed File Service Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGREC_ZFS_RM_MULTIFS

Description:
Verifies that the system has no multi-file system aggregates attached.

Reason for check:
Multi-file system aggregates should not be used. Compatibility mode aggregates should be used.

z/OS releases the check applies to:
z/OS V1R9 and later.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMZFS,ZOSMIGREC_ZFS_RM_MULTIFS)
  SEVERITY(LOW)
  INTERVAL(ONETIME)
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For additional information about the removal of zFS multi-file system aggregates, see *z/OS Distributed File Service zFS Administration*.

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- IOEZH0021E

See *z/OS Distributed File Service Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R11_ZFS_RM_MULTIFS

Description:
Verifies that a system running in a sysplex environment has no multi-file system aggregates attached.

Reason for check:
Multi-file system aggregates should not be used. Compatibility mode aggregates should be used.

z/OS releases the check applies to:
z/OS V1R9 through z/OS V1R11

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMZFS,ZOSMIGV1R11_ZFS_RM_MULTIFS)
  SEVERITY(LOW)
  INTERVAL(ONETIME)
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For additional information about the removal of zFS multi-file system aggregates, see *z/OS Distributed File Service zFS Administration*.

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- IOEZH0021E

See *z/OS Distributed File Service Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV1R13_ZFS_FILESYS

Description:
Verifies that the sysplex is running with sysplex=filesys for zFS V1R13 toleration support.

Reason for check:
zFS should be running at sysplex=filesys for all members of the sysplex. Once this is done, zFS V1R13 may be brought into the sysplex.

z/OS releases the check applies to:
z/OS V1R11 and z/OS V1R12 with apar OA35465.

User override of IBM values:
The following shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement may be copied and modified to override the check defaults:

```
UPDATE
  CHECK(IBMZFS,ZOSMIGV1R13_ZFS_FILESYS)
  SEVERITY(LOW)
  INTERVAL(ONETIME)
  DATE('date_of_the_change')
  REASON('Your reason for making the update.')
```

Parameters accepted:
No

Reference:
For additional information about zFS toleration support, see *z/OS Migration* for z/OS V1R13.

Debug support:
No

Verbose support:
No

Messages:
This check issues the following exception messages:

- IOEZH0014E

See *z/OS Distributed File Service Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

ZOSMIGV2R1_ZFS_VERIFY_CACHESIZE

Description:
Determines if the system is running with the default settings for IOEFSPRM configuration options META_CACHE_SIZE, METABACK_CACHE_SIZE and USER_CACHE_SIZE.

For user cache size, check if the current size is less than the z/OS V2R1 default user cache size. For metadata cache size and metadata backing cache size, check if the sum of the two caches is less than the sum of the z/OS V2R1 defaults.

Reason for check:
Running with a very small cache size could affect zFS performance. This check issues an exception message if either or both of the conditions are true:

- The sum of the current metadata cache size and metadata backing cache size is less than the sum of the V2R1 default metadata cache size and metadata backing cache size.
- The current user cache size is less than the z/OS V2R1 default user cache size.

z/OS releases the check applies to:
z/OS V1R12 and z/OS V1R13.

User override of IBM values:
The following example shows the default keywords for the check, which you can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement can be copied and modified to override the check defaults:

```
UPDATE
       CHECK(IBMZFS,ZOSMIGV2R1_ZFS_VERIFY_CACHESIZE)
       SEVERITY(LOW)
       INTERVAL(ONETIME)
       PARM('META_CACHE=v2r1_size1,METABACK_CACHE=v2r1_size2,USER_CACHE=v2r1_size3')
       DATE('date_of_the_change')
       REASON('Your reason for making the update.')
```

Note: The cache sizes (*v2r1_size1*, *v2r1_size2* and *v2r1_size3*) are the internally calculated defaults as of z/OS V2R1 and later, based on the current configuration option settings and the amount of real storage during zFS initialization.

Debug support:
No

Verbose support:
No

Parameters accepted:
Specifying one or more keywords of META_CACHE, METABACK_CACHE and USER_CACHE are acceptable. Each keyword defines a decimal number to be compared with the current cache size. The decimal number is limited to a

length of 10 characters and a maximum value of 2147483647 (2G-1). A K, M or G can be appended to the number to mean kilobytes, megabytes or gigabytes, respectively.

- The valid range for META_CACHE is 1 M through 1024 M.
- The valid range for METABACK_CACHE is 1 M through 2048 M.
- The valid range for USER_CACHE is 10 M to 64 G.

For example:

```
PARM('USER_CACHE=1G')           /*  To override user_cache_size default */
PARM('META_CACHE=200000000,USER_CACHE=100000K')
PARM('META_CACHE=50M, METABACK_CACHE=32M,USER_CACHE=100M')
```

Default:
The default is the internally-calculated cache size beginning in z/OS V2R1 based on the current configuration option settings and the real storage usage during zFS initialization.

Reference:
See the performance and tuning section in *z/OS Distributed File Service zFS Administration* to determine if the current setting would affect zFS performance.

See IOEFSPRM or **zfsadm config** in *z/OS Distributed File Service zFS Administration* to set or dynamically change the cache setting.

Messages:
This check issues the following exception message:

- IOEZH0032E
- IOEZH0033E

See *z/OS Distributed File Service Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information about using security labels.

ZFS_VERIFY_CACHESIZE

Description:
Checks whether the system is running with the default settings for IOEFSPRM configuration options META_CACHE_SIZE, METABACK_CACHE_SIZE and USER_CACHE_SIZE.

For user-defined user cache size, check if the current size is less than the default user cache size. For either or both user-defined metadata cache size and metadata backing cache size, check if the sum of the two metadata caches is less than the sum of the defaults.

Reason for check:
Running with a very small cache size could affect zFS performance. This check issues an exception message if either or both of the conditions are true:

- For user-defined user cache size, check if the current size is less than the default user cache size. For either or both user-defined metadata cache size and metadata backing cache size, check if the sum of the two metadata caches is less than the sum of the defaults.
- The user-defined user cache size is less than the default user cache size.

z/OS releases the check applies to:
z/OS V2R1 and later.

User override of IBM values:
The following example shows the default keywords for the check, which you

can override on either a POLICY statement in the HZSPRMxx parmlib member or on a MODIFY command. This statement can be copied and modified to override the check defaults:

```
UPDATE
        CHECK(IBMZFS,ZFS_VERIFY_CACHESIZE)
        SEVERITY(LOW)
        INTERVAL(ONETIME)
        PARM('META_CACHE=size1, METABACK_CACHE=size2,USER_CACHE=size3')
        DATE('date_of_the_change')
        REASON('Your reason for making the update.')
```

Note: The cache sizes (*size1*, *size2* and *size3*) are the internally-calculated defaults based on the current configuration option settings and the amount of real storage during zFS initialization.

Debug support:
No

Verbose support:
No

Parameters accepted:
Specifying one or more keywords of META_CACHE, METABACK_CACHE and USER_CACHE is acceptable. Each keyword defines a decimal number to be compared with the current cache size. The decimal number is limited to a length of 10 characters and a maximum value of 2147483647 (2G-1). A `K`, `M` or `G` can be appended to the number to mean kilobytes, megabytes or gigabytes, respectively.

- The valid range for META_CACHE is 1 M through 1024 M.
- The valid range for METABACK_CACHE is 1 M through 2048 M.
- The valid range for USER_CACHE is 10 M to 64 G.

For example:

```
PARM('USER_CACHE=1G')                      /*  To override the user_cache_size default */
PARM('META_CACHE=200000000,USER_CACHE=100000K')
PARM('META_CACHE=50M, METABACK_CACHE=32M,USER_CACHE=100M')
```

Default:
The default is the internally-calculated cache size beginning in z/OS V2R1 based on the current configuration option settings and the amount of real storage during zFS initialization.

Reference:
None.

Messages:
This check issues the following exception messages:

- IOEZH0044E
- IOEZH0045E

See *z/OS Distributed File Service Messages and Codes*.

SECLABEL recommended for multilevel security users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information about using security labels.

Part 4. Appendixes

Appendix. Accessibility

Accessible publications for this product are offered through the z/OS Information Center.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

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

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Dotted decimal syntax diagrams

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

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually

exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

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

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

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

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

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

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

  Note:

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

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

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

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

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

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

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

- For information about software support lifecycle, see: http://www.ibm.com/software/support/systemsz/lifecycle/
- For information about currently-supported IBM hardware, contact your IBM representative.

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 at (no navtitle).

Index

A

accessibility 631
  contact IBM 631
  features 631
ACTIVATE parameter
  HZSPRMxx 70, 87
  MODIFY command 70, 87
ADD PARMLIB parameter
  HZSPRMxx 86
  MODIFY command 86
ADD POLICY parameter
  HZSPRMxx 87
  MODIFY command 87
ADDNEW parameter
  HZSPRMxx 70
  MODIFY command 70
ADDREPLACE POLICY parameter
  HZSPRMxx 87
  MODIFY command 87
allocate the HZSPDATA data set 10
assistive technologies 631
automating
  check output
    exception messages 32

C

Catalog
  check description 392
categories
  HZSPRMxx
    syntax 50
  MODIFY command
    syntax 50
CFLEVEL 19
  Flash Express xix
changes
  z/OS V2R1 xix
check description
  Catalog 392
  Communications Server 394
  consoles 411
  Contents supervision 422
  DAE 428
  DFSMS OPEN/CLOSE/EOV 431
  Global Resource Serialization 432
  HSM 437
  ICSF 441
  Infoprint Server 449
  JES2 checks 463
  PDSE 466
  PFA 467
  Predictive failure analysis 467
  RACF 467
  RCF 500
  reconfiguration 500
  RRS 508
  RSM 517
  RSMM 502
  SDSF 524
check description *(continued)*
  SDUMP 526
  SLIP 527
  SMB 528
  SMS 530
  system logger 541
  System trace 549
  Timer supervisor 551
  TSO/E 552
  VLF 569
  VSAM 571
  VSAM RLS 572
  VSM 579
  XCF 590
  z/OS UNIX 554
check messages 25
  message table 205
    CSECT 231, 232
    example 205
  planning 201
  tags 214
  variables 200
check output 25
  evaluating 30
  exception messages
    automating 32
  resolving exceptions 30
  state
    reading 34
check routine
  environment 109, 138
  function codes 113
    remote check 145
  gotchas 109
  HZSFMSG macro 117, 150
  HZSPQE fields, using 111, 144
  input registers 110
  issuing messages in 117, 150
  output registers 110
  programming considerations 109, 138
  recovery 110, 127, 139, 161
  reporting exceptions 118, 151, 182
  requirements 109, 138
  restrictions 109, 138
  sample 106, 136
  writing 106, 137
check terminology 96
checks
  deleting 48
  description 381
  GRS_AUTHQLVL_SETTING 432
  GRS_CONVERT_RESERVES 434
  GRS_EXIT_PERFORMANCE 435
  GRS_GRSQ_SETTING 436
  GRS_Mode 433
  GRS_RNL_IGNORED_CONV 436
  GRS_SYNCHRES 434
  managing 43
    using MODIFY 47
    using SDSF 45
  obtaining additional 23

checks *(continued)*
RACF installation defined 467
RACF_AIM_STAGE 471
RACF_CERTIFICATE_EXPIRATION 473
RACF_FACILITY_ACTIVE 488
RACF_GRS_RNL 475
RACF_IBMUSER_REVOKED 490
RACF_OPERCMDS_ACTIVE 488
RACF_SENSITIVE_RESOURCES 482
RACF_TAPEVOL_ACTIVE 488
RACF_TEMPDSN_ACTIVE 488
RACF_TSOAUTH_ACTIVE 488
RACF_UNIX_ID 491
RACF_UNIXPRIV_ACTIVE 488
RRS_DUROffloadSize 511
RRS_MUROffloadSize 511
RRS_RMDataLogDuplexMode 509
RRS_RMDOffloadSize 510
RRS_RSTOffloadSize 508, 512
RRS_Storage_NumLargeMSGBlks 513, 514
RRS_Storage_NumServerReqs 515
RRS_Storage_NumTransBlks 516
RSM_AFQ 520
RSM_HVSHARE 517
RSM_MAXCADS 519
RSM_MEMLIMIT 518
RSM_REAL 521
RSU_RSU 522
SVA_AUTOIPL_DEFINED 464
SVA_AUTOIPL_DEV_VALIDATION 465
TSOE_PARMLIB_ERROR 553
TSOE_USERLOGS 552
USS_CLIENT_MOUNTS 554
USS_FILESYS_CONFIG 556
USS_HFS_DETECTED 558
USS_KERNEL_PVTSTG_THRESHOLD 559
USS_KERNEL_STACKS_THRESHOLDS 560
USS_MAXSOCKETS_MAXFILEPROC 554, 561
USS_PARMLIB 562
USS_PARMLIB_MOUNTS 565
ZFS_VERIFY_CACHESIZE 627
ZOSMIGREC_ZFS_RM_MULTIFS 623
ZOSMIGV1R10_RMM_REJECTS_DEFINED 502
ZOSMIGV1R10_RMM_VOL_REPLACE_LIM 503
ZOSMIGV1R10_RMM_VRS_DELETED 504
ZOSMIGV1R11_RMM_DUPLICATE_GDG 505
ZOSMIGV1R11_RMM_REXX_STEM 506
ZOSMIGV1R11_RMM_VRSEL_OLD 507
ZOSMIGV1R11_ZFS_INTERFACELEVEL 623
ZOSMIGV1R11_ZFS_RM_MULTIFS 624
ZOSMIGV1R13_DEFAULT_UNIX_ID 496
ZOSMIGV1R13_ZFS_FILESYS 625
ZOSMIGV2R1_ZFS_VERIFY_CACHESIZE 626
Communications Server
check description 394
completion codes
HZSPRINT utility 39
consoles
check description 411
Contents supervision
check description 422
creating HZSPRMxx parmlib members 20
creating security definitions 14
assigning a user ID with superuser authority 14
for a IBM Health Checker for z/OS log stream 13
for HZSPRINT utility 16
in an MLS environment 19

D

DAE
check description 428
DEACTIVATE parameter
HZSPRMxx 70
MODIFY command 70
defining a log stream 12
DELETE parameter
HZSPRMxx 71
MODIFY command 71
deleting checks 48
description
checks 381
developing checks 105, 135, 169
check routine 106, 137
REXX check 170
sample 106, 136, 169
DFSMS OPEN/CLOSE/EOV
check description 431
DISPLAY parameter
HZSPRMxx 71
output 89
MODIFY command 71
output 89

E

example
message input 205
policy 63
exception messages
automating 32
evaluating 30
resolving 25, 30

F

F command
parameters 65
syntax 65
filters
HZSPRMxx 69
MODIFY command 69
Flash Express
statement of direction xix
function codes 113
remote check 145

G

Global Resource Serialization
check description 432
GRS_AUTHQLVL_SETTING 432
GRS_CONVERT_RESERVES 434
GRS_EXIT_PERFORMANCE 435
GRS_GRSQ_SETTING 436
GRS_Mode 433
GRS_RNL_IGNORED_CONV 436
GRS_SYNCHRES 434

H

HSM
check description 437

HZSADDCHECK exit routine
environment 194
input registers 194
multiple checks in 195
output registers 195
sample 136
writing 191
HZSADDCHECK exit routine r
adding checks to the system 196
HZSADDCK macro 278
HZSCHECK macro 293
HZSCPARS macro 294
HZSFMSG macro 338
in check routine 117, 150
HZSLFMSG function
in REXX check 180
HZSPDATA data set
allocating 10
sizing 11
HZSPDATA parameter
HZSPRMxx 73
MODIFY command 73
HZSPQE 111, 144
HZSPREAD macro 339, 348
HZSPRINT utility
completion codes 39
output example 39
security definitions for 16
setting up 12
using 37
HZSPRMxx
ACTIVATE parameter 70, 87
ADD PARMLIB parameter 86
ADD POLICY parameter 87
ADDNEW parameter 70
ADDREPLACE POLICY parameter 87
DEACTIVATE parameter 70
DELETE parameter 71
DISPLAY parameter 71
output 89
filters 69
HZSPDATA parameter 73
LOGGER parameter 74
parameters 65
POLICY parameter 87, 88
REFRESH parameter 74
REMOVE POLICY parameter 87, 88
REPLACE,PARMLIB parameter 86
RUN parameter 74
specifying members 64
STOP parameter 75
syntax 65
using categories 50
using wildcards 65
UPDATE parameter 75
HZSPRMxx parmlib member
creating 20
HZSPROC parameter 73
MODIFY command 73
HZSQUERY macro 378

I

IBM Health Checker for z/OS
planning checks 95
setting up 7
software requirements 10
ICSF
check description 441
Infoprint Server
check description 449

J

JES2 checks
check description 463

K

keyboard
navigation 631
PF keys 631
shortcut keys 631

L

log stream
defining 12
LOGGER parameter
HZSPRMxx 74
MODIFY command 74

M

macros
HZS 259
HZSADDCK 259
HZSFMSG 259
HZSQUERY 259
managing checks 43, 45
policy 52, 53
example 63
message input
creating 199
message table 205
CSECT 231, 232
example 205
planning 201
tags 214
variables 200
message table
sample 106, 136, 169
messages
in check routine 117, 150
Metal C
checks 99
MODIFY command
ACTIVATE parameter 70, 87
ADD PARMLIB parameter 86
ADD POLICY parameter 87
ADDNEW parameter 70
ADDREPLACE POLICY parameter 87
DEACTIVATE parameter 70
DELETE parameter 71
DISPLAY parameter 71
output 89
filters 69
HZSPDATA parameter 73
LOGGER parameter 74
managing checks with 47
parameters 65
POLICY parameter 87, 88

MODIFY command *(continued)*
reading check state 34
REFRESH parameter 74
REMOVE POLICY parameter 87, 88
REPLACE,PARMLIB parameter 86
RUN parameter 74
STOP parameter 75
syntax 65
using categories 50
using wildcards 65
UPDATE parameter 75

N

navigation
keyboard 631
Notices 635

O

output registers
check routine 110
output, check 25
automating 32

P

parameters
HZSPRMxx 65
MODIFY command 65
PDSE
check description 466
PFA
check description 467
planning checks 95
identify potential checks 96
policy
example 63
managing checks with 52, 53
using dates on 58
POLICY parameter
HZSPRMxx 87, 88
MODIFY command 87, 88
Predictive failure analysis
check description 467

R

RACF
check description 467
RACF installation defined check 467
RACF_AIM_STAGE 471
RACF_CERTIFICATE_EXPIRATION 473
RACF_FACILITY_ACTIVE 488
RACF_GRS_RNL 475
RACF_IBMUSER_REVOKED 490
RACF_OPERCMDS_ACTIVE 488
RACF_SENSITIVE_RESOURCES 482
RACF_TAPEVOL_ACTIVE 488
RACF_TEMPDSN_ACTIVE 488
RACF_TSOAUTH_ACTIVE 488
RACF_UNIX_ID 491
RACF_UNIXPRIV_ACTIVE 488
RCF
check description 500
reading
check output
state 34
reconfiguration
check description 500
recovery
check routine 110, 139
REFRESH parameter
HZSPRMxx 74
MODIFY command 74
REMOVE POLICY parameter
HZSPRMxx 87, 88
MODIFY command 87, 88
REPLACE PARMLIB parameter
HZSPRMxx 86
REPLACE,PARMLIB parameter
MODIFY command 86
REXX check
issuing messages in 180
recovery 187
writing 170
RMM
check description 502
RRS
check description 508
RRS_DUROffloadSize 511
RRS_MUROffloadSize 511
RRS_RMDataLogDuplexMode 509
RRS_RMDOffloadSize 510
RRS_RSTOffloadSize 508, 512
RRS_Storage_NumLargeMSGBlks 513, 514
RRS_Storage_NumServerReqs 515
RRS_Storage_NumTransBlks 516
RSM
check description 517
RSM_AFQ 520
RSM_HVSHARE 517
RSM_MAXCADS 519
RSM_MEMLIMIT 518
RSM_REAL 521
RSM_RSU 522
RUN parameter
HZSPRMxx 74
MODIFY command 74

S

sample
check routine 106, 136
HZSADDCHECK exit routine 136
message table 106, 136
SDSF
check description 524
managing checks with 45
reading check state 34
SDUMP
check description 526
security definitions
creating 14
assigning a user ID with superuser authority 14
for a IBM Health Checker for z/OS log stream 13
for HZSPRINT utility 16
in an MLS environment 19
sending comments to IBM xv
setting up
IBM Health Checker for z/OS 7
setting up the HZSPRINT utility 12

shortcut keys 631
sizing the HZSPDATA data set 11
SLIP
  check description 527
SMB
  check description 528
SMS
  check description 530
state
  check 34
statement of direction
  coupling facility list structures xix
STOP parameter
  HZSPRMxx 75
  MODIFY command 75
summary of changes xvii, xix
SVA_AUTOIPL_DEFINED 464
SVA_AUTOIPL_DEV_VALIDATION 465
syntax
  HZSPRMxx 65
  MODIFY command 65
System logger
  check description 541
System trace
  check description 549

T

Timer supervisor
  check description 551
trademarks 639
TSO/E
  check description 552
TSOE_PARMLIB_ERROR 553
TSOE_USERLOGS 552

U

UPDATE parameter
  HZSPRMxx 75
  MODIFY command 75
user interface
  ISPF 631
  TSO/E 631
USS_CLIENT_MOUNTS 554
USS_FILESYS_CONFIG 556
USS_HFS_DETECTED 558
USS_KERNEL_PVTSTG_THRESHOLD 559
USS_KERNEL_STACKS_THRESHOLD 560
USS_MAXSOCKETS_MAXFILEPROC 554, 561
USS_PARMLIB 562
USS_PARMLIB_MOUNTS 565

V

V2R1 changes xix
viewing
  check output 25
VLF
  check description 569
VSAM
  check description 571
VSAM RLS
  check description 572
VSM
  check description 579

W

writing checks 105, 135, 169
  sample 106, 136, 169

X

XCF
  check description 590

Z

z/OS UNIX
  check description 554
ZFS_VERIFY_CACHESIZE 627
ZOSMIGREC_ZFS_RM_MULTIFS 623
ZOSMIGV1R10_RMM_REJECTS_DEFINED 502
ZOSMIGV1R10_RMM_VOL_REPLACE_LIM 503
ZOSMIGV1R10_RMM_VRS_DELETED 504
ZOSMIGV1R11_RMM_DUPLICATE_GDG 505
ZOSMIGV1R11_RMM_REXX_STEM 506
ZOSMIGV1R11_RMM_VRSEL_OLD 507
ZOSMIGV1R11_ZFS_INTERFACELEVEL 623
ZOSMIGV1R11_ZFS_RM_MULTIFS 624
ZOSMIGV1R13_DEFAULT_UNIX_ID 496
ZOSMIGV1R13_ZFS_FILESYS 625
ZOSMIGV2R1_ZFS_VERIFY_CACHESIZE 626

Product Number: 5650-ZOS

Printed in USA

SC23-6843-02