z/OS

IBM

IBM Health Checker for z/OS User's Guide

Version 1 Release 9

SA22-7994-05

z/OS

IBM Health Checker for z/OS User's Guide

Version 1 Release 9

SA22-7994-05

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 389.

Sixth Edition, September 2007

This edition applies to Version 1 Releases 9 of z/OS (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers’ comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this document
- Page number or topic related to your comment

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

This was a team effort!

This book was a team effort:
It was put together with help from all the following people:

Allen Carney
Bill Keller
Bill Schoen
Bobby Gardinor
Brian Thomas
Charles Mari
Chris Hastings
Daniel Acevedo
Dave Gill
Dave Stilwell
Dave Surman
David Hom
David Yackel
Debbie Beatrice

Don Goldie
Doug Zobre
Elpida Tzortzatos
Geoff Miller
Jason Grusauskas
Jim Becker
Jim Guilianelli
Joseph Marsh
Karen Ransom
Ken Jonas
Kevin Minerley
Luis Miranda
Mark Nelson
Marne Walle

Kristine Logan

Mary Anne Hogarty
Mike Phillips
Nicole Fagen
Pete Redmond
Peter Relson
Ralph Sharpe
Rob Snyder
Sharon Phillips
Shirley Jubinsky
Simala Qamar
Steve Partlow
Teresa Hood
Thomas Shaw
Yoan Johnson

Contents

Figures

Tables

About This Document

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.

This edition supports z/OS (5694-A01), and z/OS.e (5655-G52).

IBM Health Checker for z/OS is available for z/OS V1R4, V1R5, V1R6, and later users. This document applies to z/OS R4, R5, R6, R7, R8, and R9.

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, "Reference," on page 195 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 75 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, "Reference," on page 195 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:

| `//http://www.ibm.com/servers/eserver/zseries/zos/downloads/`

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.

z/OS information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS™ and z/OS.e, see the online document at:

`http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.`

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM® messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message explanations for z/OS elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux™:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX® System Services).
- Your Microsoft® Windows® workstation. You can install code to access IBM message explanations on the *z/OS Collection*(SK3T-4269), using LookAt from a Microsoft Windows command prompt (also known as the DOS command line).
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from a disk on your *z/OS Collection*(SK3T-4269), or from the LookAt Web site (click **Download**, and select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

Summary of changes

Summary of changes
for SA22-7994-05
z/OS Version 1 Release 9

The document contains information previously presented in *IBM Health Checker for z/OS: User's Guide*, SA22-7994-03, which supports z/OS Version 1 Releases 8.

This document includes new and changed sections:

- New information about "Understanding system data issued with the check messages" on page 21.
- New SYSNAME parameter for use with the LOGSTREAM parameter of HZSPRINT. See "Using the HZSPRINT utility" on page 29.
- Information on clearing parameter errors and understanding check deletion. See "Using the MODIFY *hzsproc* command to manage checks" on page 38.
- Updates to "Policy statement examples" on page 50.
- New and updated parameters for the F *hzsproc* command and HZSPRMxx parmlib member, including support for displaying policy information, and new parameters for System REXX check support. See "Statements and parameters" on page 56.
- Planning information for REXX exec checks. See:
  – "What kind of check do you want to write?" on page 81
  – "REXX checks" on page 83
- New recovery information for local and remote checks:
  – "The well-behaved local check routine - recommendations and recovery considerations" on page 103
  – "Recommendations and recovery considerations for remote checks" on page 127
- New chapter on writing REXX exec checks. See Chapter 8, "Writing REXX checks," on page 131.
- Information on writing an optional HZSADDCHECK exit routine for local and REXX exec checks separated into a new chapter, with support for REXX exec checks added. See Chapter 9, "Writing an HZSADDCHECK exit routine," on page 147.
- New chapter on Chapter 11, "IBM Health Checker for z/OS System REXX Functions," on page 199, HZSLSTRT, HZSLFMSG, and HZSLSTOP.
- Support for REXX exec checks added to "HZSADDCK macro — HZS add a check" on page 218.
- New and changed checks. See Chapter 13, "IBM Health Checker for z/OS checks," on page 301.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes
for SA22-7994-04
z/OS Version 1 Release 8
as updated April 2007

The document contains information previously presented in *IBM Health Checker for z/OS: User's Guide*, SA22-7994-03, which supports z/OS Version 1 Release 8.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes
for SA22-7994-03
z/OS Version 1 Release 8

The document contains information previously presented in *IBM Health Checker for z/OS: User's Guide*, SA22-7994-02, which supports z/OS Version 1 Releases 4, 5, 6, and 7.

This document includes new and changed sections:

- Support for multiple IBM Health Checker for z/OS policies. See "Creating IBM Health Checker for z/OS policies" on page 44.
- Updates to "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc* command" on page 53.
- Support for writing remote checks (checks that run in caller's address space). See Chapter 7, "Writing remote check routines," on page 109.
- New check parameter parsing service. See "Using the check parameter parsing service (HZSCPARS)" on page 92 and "HZSCPARS macro — HZS Check Parameter Parsing" on page 289.
- A local check HZSADDCHECK exit routine is now optional. You can also use ADD or ADDREPLACE CHECK in an HZSPRMxx parmlib member to define check defaults and add the check to IBM Health Checker for z/OS. See
  - "Remote checks" on page 82
  - "ADD or ADDREPLACE CHECK parameters" on page 66
  - Chapter 9, "Writing an HZSADDCHECK exit routine," on page 147
- Changes to <msglist> tags to take advantage of enhancements for checks running on z/OS V1R8 and higher systems. See "Decide what release your check will run on" on page 161 and "Message list tag - <msglist>" on page 172.
- Support for national language translation of check exception messages. See:
  - "Decide whether to translate your exception messages into other national languages" on page 161
  - "<msgtext>" on page 175
  - "Support for translating messages to other languages" on page 191
- New and changed checks:
  - New "ASM checks (IBMASM)" on page 302
  - New "Communications Server checks (IBMCS)" on page 306
  - New and changed "RACF checks (IBMRACF)" on page 331
  - New RRS check, "RRS_ArchiveCFStructure" on page 348
- Changes to macros supporting new functions:
  - "HZSADDCK macro — HZS add a check" on page 218
  - "HZSFMSG macro — Issue a formatted check message" on page 236
  - "HZSQUERY macro — HZS Query" on page 259
  - "HZSCHECK macro — HZS Check command request" on page 275
- New macro, "HZSCPARS macro — HZS Check Parameter Parsing" on page 289

Summary of changes
for SA22-7994-01
z/OS Version 1 Releases 4, 5, 6, and 7
as updated October 2005

The document contains information previously presented in *IBM Health Checker for z/OS: User's Guide*, SA22-7994-00, which supports z/OS Version 1 Releases 4, 5, 6, and 7.

This document includes new and changed sections:
- "Approaches to automation with IBM Health Checker for z/OS" on page 24
- "Finding check message documentation with LookAt" on page 33
- "Can I put non-policy statements in my HZSPRMxx member?" on page 50
- "Local check routine basics" on page 87
- "Issuing messages in your check routine with the HZSFMSG macro" on page 95
- "Defining the variables for your messages" on page 97
- "Dynamically adding local or REXX exec checks to IBM Health Checker for z/OS" on page 151
- "Debugging checks" on page 106
- Chapter 10, "Creating the message input for your check," on page 155
- "How messages and message variables are issued at check runtime" on page 156

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes
for SA22-7994-02
z/OS Version 1 Releases 4, 5, 6, and 7
as updated January 2006

The document contains information previously presented in *IBM Health Checker for z/OS: User's Guide*, SA22-7994-01, which supports z/OS Version 1 Releases 4, 5, 6, and 7.

This document includes new and changed sections:
- "Setting up security for the HZSPRINT utility" on page 11
- "Obtain checks for IBM Health Checker for z/OS" on page 17
- "System logger checks (IBMIXGLOGR)" on page 357

Summary of changes
for SA22-7994-01
z/OS Version 1 Releases 4, 5, 6, and 7
as updated October 2005

The document contains information previously presented in *IBM Health Checker for z/OS: User's Guide*, SA22-7994-00, which supports z/OS Version 1 Releases 4, 5, 6, and 7.

This document includes new and changed sections:
- "Approaches to automation with IBM Health Checker for z/OS" on page 24
- "Finding check message documentation with LookAt" on page 33
- "Can I put non-policy statements in my HZSPRMxx member?" on page 50
- "Local check routine basics" on page 87
- "Issuing messages in your check routine with the HZSFMSG macro" on page 95
- "Defining the variables for your messages" on page 97

- “Dynamically adding local or REXX exec checks to IBM Health Checker for z/OS” on page 151
- “Debugging checks” on page 106
- Chapter 10, “Creating the message input for your check,” on page 155
- “How messages and message variables are issued at check runtime” on page 156

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

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.

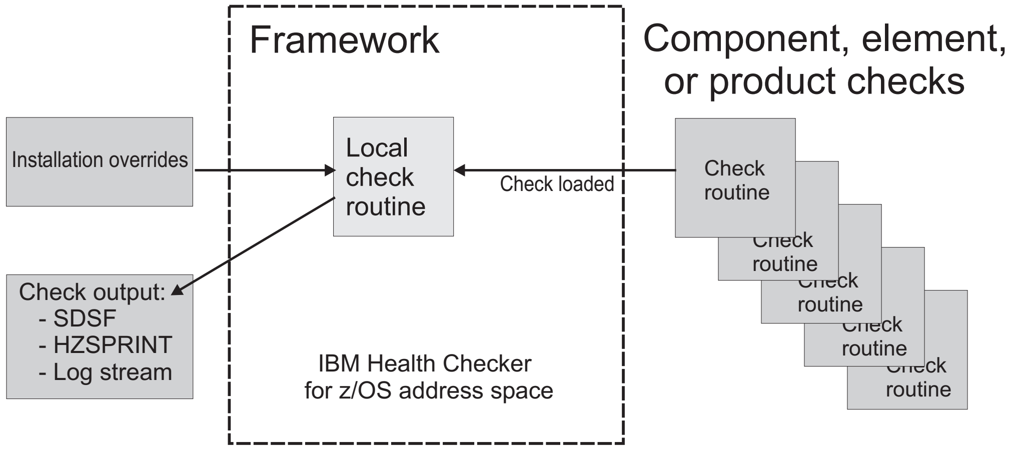


Figure 1. IBM Health Checker for z/OS with a local check

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 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 19.
- 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 39.
- Updating check values temporarily using SDSF or the MODIFY *hzsproc* command. See "Making dynamic, temporary changes to checks" on page 36.
- Updating check values permanently using HZSPRMxx. See "Making persistent changes to checks" on page 44.

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/
- *System z Platform Test Report for z/OS and Linux Virtual Servers* at: http://www.ibm.com/servers/eserver/zseries/zos/integtst/
- 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*, SA22-7592
  - *z/OS MVS Planning: Global Resource Serialization*
  - *z/OS MVS Planning: Operations*, SA22-7601
  - *z/OS MVS Setting Up a Sysplex*, SA22-7625
  - *z/OS Security Server RACF Command Language Reference*
  - *z/OS Security Server RACF Security Administrator's Guide*
  - *z/OS UNIX System Services Planning*, GA22-7800

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 301.

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 35.

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.

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 17 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 35.

Use the following steps to set up and start IBM Health Checker for z/OS:

1. "Software requirements for IBM Health Checker for z/OS"
2. "Allocate the HZSPDATA data set to save check data between restarts" on page 8
3. "Set up the HZSPRINT utility" on page 8
4. "Define log streams to keep a record of the check output" on page 8, as needed
5. "Create security definitions" on page 10
6. Set up customization and security for SDSF support for IBM Health Checker for z/OS in IBM Health Checker for z/OS Small Programming Enhancement in *z/OS SDSF Operation and Customization*.
7. "Create multilevel security definitions" on page 14
8. "Optionally create HZSPRMxx from the HZSPRM00 parmlib member" on page 15
9. "Start IBM Health Checker for z/OS" on page 16
10. "Obtain checks for IBM Health Checker for z/OS" on page 17

Software requirements for IBM Health Checker for z/OS

The IBM Health Checker for z/OS is shipped as part of z/OS V1R7 and z/OS V1R8.

| 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:

| `//http://www.ibm.com/servers/eserver/zseries/zos/downloads/`

IBM Health Checker for z/OS can run on a parallel sysplex, monoplex, or XCF local mode environment running z/OS V1R4, V1R5, V1R6 or V1R7. Note that different checks have different system level requirements - see Chapter 13, "IBM Health Checker for z/OS checks," on page 301 for check specific information.

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 the IBM Health Checker for z/OS start up procedure, HZSPROC. See "Start IBM Health Checker for z/OS" on page 16.

Set up the HZSPRINT utility

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

1. Get the JCL for the HZSPRINT utility from member HZSPRINT in SYS1.SAMPLIB .
2. "Setting up security for the HZSPRINT utility" on page 11.
3. See "Using the HZSPRINT utility" on page 29.

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 a good idea, you must 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.

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 ″Set up the SMS environment for DASD data sets″ 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 10.
   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 Setting Up Policies section of *z/OS MVS Setting Up a Sysplex*.
   c. See "Security for printing check output from a log stream" on page 14 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)
     ```
4. To disable a log stream, issue the following MODIFY command:
   ```
   f hzsproc,logger=off
   ```

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(8000)
     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/servers/eserver/zseries/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(64000)
     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 11
- “Setting up security for IBM Health Checker for SDSF support” on page 14

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 (UID(0))
   - A home directory of HOME('/')
   - A program of PROGRAM('/bin/sh')

For example, you might define the user ID as follows::

```
ADDUSER hcsuperid
        OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
        NOPASSWORD
ADDGROUP OMVSGRP OMVS(GID(46))
CONNECT hcsuperid GROUP(OMVSGRP)
```

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*

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) RACLIST(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*.

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.”
- 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 14.

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(checkowner,checkname)'
```

The following HZSPRINT EXEC statement examples show different ways users can specify the checkname 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 19 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 name
`CHECK(*,`*`checkname`*`)` `CHECK(*,*)`	QUERY: Read access to all checks	HZS.*sysname*.QUERY
	MESSAGES: Read access to individual check	HZS.*sysname*.*check_owner*.MESSAGES **or** 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 **or** HZS.*sysname*.*check_owner*.*check_name*.MESSAGES
`CHECK(`*`checkowner`*`,`*`checkname`*`)`	QUERY: Read access to individual check	HZS.*sysname*.*check_owner*.QUERY **or** HZS.*sysname*.*check_owner*.*check_name*.QUERY
	MESSAGES: Read access to individual check	HZS.*sysname*.*check_owner*.MESSAGES **or** 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(hcprint) 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 29 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

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 10. 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 301 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)
  ```

Optionally create HZSPRMxx from the HZSPRM00 parmlib member

You do not have to set up an HZSPRMxx parmlib member to get IBM Health Checker up and running. And at first, you’ll 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 should use an HZSPRMxx parmlib member to 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)
  ```

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

You can create the policy statements in your HZSPRMxx parmlib member using input from:
- The sample syntax in the HZSPRM00 member in SYS1.PARMLIB.
- The **Parameters accepted** portion of each check description in Chapter 13, "IBM Health Checker for z/OS checks," on page 301.
- "Making persistent changes to checks" on page 44

Then you'll specify the parmlib member you create in the IBM Health Checker for z/OS procedure, *hzsproc*. See "Specifying the HZSPRMxx members you want the system to use" on page 51.

Start IBM Health Checker for z/OS

To start IBM Health Checker for z/OS, you use the HZSPROC started procedure. The HZSPROC JCL procedure looks as follows:

```
//HZSPROC JOB JESLOG=SUPPRESS
//HZSPROC  PROC HZSPRM='00'
//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.

1. Copy the sample IBM Health Checker for z/OS procedure, HZSPROC, into a PROCLIB data set.
2. Update the procedure to:
   - Make sure that the procedure includes the name of the HZSPDATA data set defined in "Allocate the HZSPDATA data set to save check data between restarts" on page 8.
   - Make sure that the procedure includes the name of the HZSPRMxx member defined in "Optionally create HZSPRMxx from the HZSPRM00 parmlib member" on page 15.

   ```
   //HZSPROC JOB JESLOG=SUPPRESS
   //HZSPROC  PROC HZSPRM='00'
   //HZSSTEP  EXEC   PGM=HZSINIT,REGION=0K,TIME=NOLIMIT,
   //      PARM='SET PARMLIB=&HZSPRM'
   //HZSPDATA DD   DSN=SYS1.&SYSNAME..HZSPDATA,DISP=OLD
   //        PEND
   //        EXEC HZSPROC
   ```

   If you convert started tasks to jobs in your installation, see "Convert Procedures to Jobs" in the *z/OS MVS JCL Reference*. When you add a JOB statement for HZSPRINT to the JCL, make sure to include JESLOG=SUPPRESS.
3. Update the COMMNDxx parmlib member with the IBM Health Checker for z/OS procedure, as shown in the following example:

   ```
   COM='START hzsproc'
   ```
4. Start IBM Health Checker for z/OS:

- Start IBM Health Checker for z/OS with one or more HZSPRMxx parmlib members using one of the following commands:

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

  Where *xx* is the suffix of the HZSPRMxx member you want to use.
- You can start IBM Health Checker for z/OS **without** specifying an HZSPRMxx parmlib member using the following command:

  ```
  START hzsproc
  ```

  The system uses default HZSPRMxx member HZSPRM00 if you do not specify a member on the HZSPRM= parameter.

In subsequent IPLs, the IBM Health Checker for z/OS procedure will start automatically, as prompted in the COMMNDxx parmlib member. If you start HZSPROC without specifying an HZSPRMxx member, the system uses HZSPRM=00 as the default.

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

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

This output reflects the fact that the HZSPDATA data set as yet contains no data.

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 151.

Current checks at the time this document was published are described in Chapter 13, “IBM Health Checker for z/OS checks,” on page 301.

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 22.
- **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 "Set up the HZSPRINT utility" on page 8 and "Using the HZSPRINT utility" on page 29.
- **SDSF** - see "Using SDSF to manage checks" on page 37
- **A log stream** - see "Define log streams to keep a record of the check output" on page 8

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_MASTER)
  START TIME: 07/05/2005 14:45:22.739250
  CHECK DATE: 20040816  CHECK SEVERITY: HIGH

  HZS1003E CHECK(IBMCNZ,CNZ_SYSCONS_MASTER):
  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: 07/05/2005 14:45:22.740948  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 to the message buffer. The example below shows the output from a check, including a report and an exception message in the message buffer:

```
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.

  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 active MCS and SMCS
   consoles, only the consoles active on this system are checked.  For
   inactive MCS and SMCS consoles, all consoles 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
           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. If an
   EMCS console is not active, find out which product activated it and
   contact the product owner.  If the EMCS console is no longer
   needed, use the EMCS console removal service (IEARELEC) to remove
   the EMCS console definition.

 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: 06/08/2005 09:49:17.451937  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 21
- "Understanding system data issued with the check messages" on page 21
- "Understanding exception messages" on page 22
- "Evaluating check output and resolving exceptions" on page 23
- "Approaches to automation with IBM Health Checker for z/OS" on page 24
- "Understanding check state and status" on page 26
- "Using the HZSPRINT utility" on page 29
- "Finding check message documentation with LookAt" on page 33

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 23 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)
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
```

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.

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” on page 65 and “WTOTYPE” on page 65 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
```

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 24.

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 35.
- 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 26

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 301.

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” on page 25.

Our approach to automation on a test sysplex:

1. **Automate start up:** We set up our test systems so that IBM Health Checker for z/OS starts automatically every time a system IPLs. We do this by specifying the IBM Health Checker for z/OS procedure in the COMMNDxx parmlib member as described in “Start IBM Health Checker for z/OS” on page 16.
2. **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
```
3. **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 26. 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 33.
- 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 301.

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_MASTER               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” on page 28
2. “IBM Health Checker for z/OS controlled states” on page 28

Check status: For check status, see “Check status” on page 29.

User controlled states

Table 2. 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*).
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 3. 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 disables them (see “Disabled state”).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. See “Global state” on page 29. 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.

Table 3. States controlled by IBM Health Checker for z/OS (continued)

Check state	Description
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 4. Check state combinations

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 “Disabled state” on page 28). The check will not run. **INACTIVE(ENABLED)** or **IE**: A user has deactivated the check (see “Inactive state” on page 28). 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 (see “Disabled state” on page 28) and a user deactivated it (see “Inactive state” on page 28). 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 *z/OS MVS System Messages, Vol 6 (GOS-IEA)* for a list of all the possible values for the check status.

Using the HZSPRINT utility

The HZSPRINT utility allows you to look at check output. HZSPRINT writes the current 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 11.

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

```
//HZSPRINT EXEC PGM=HZSPRNT,TIME=1440,REGION=0M,
//    PARM=('CHECK(check_owner,check_name)')
//*   PARM=('CHECK(check_owner,check_name)',
//*        'EXCEPTIONS')
//*   PARM=('LOGSTREAM(logstreamname)')
//*   PARM=('LOGSTREAM(logstreamname)',
//*        'CHECK(owner,name)')
//*   PARM=('LOGSTREAM(logstreamname)','EXCEPTIONS',
//*        'CHECK(owner,name)')
//*   PARM=('LOGSTREAM(logstreamname)','EXCEPTIONS')
//*   PARM=('LOGSTREAM(logstreamname)','SYSNAME(sysname)')
//*   PARM=('LOGSTREAM(logstreamname)','SYSNAME(sysname)',
//*        'CHECK(owner,name)')
//*   PARM=('LOGSTREAM(logstreamname)','EXCEPTIONS',
//*        'SYSNAME(sysname)',
//*        'CHECK(owner,name)')
//*   PARM=('LOGSTREAM(logstreamname)','EXCEPTIONS',
//*        'SYSNAME(sysname)')
//SYSOUT   DD SYSOUT=A,DCB=(LRECL=256)
```

HZSPRINT parameters:

'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 11.

,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'
```

,LOGSTREAM(*log_stream_name***)**
Optional parameter LOGGER specifies that you want to print the specified log stream.

SYSNAME(*system_name***)**
Optional parameter SYSNAME lets you limit the output in SYSOUT to output from checks running on the specified system. *system_name* is the name of a system where the checks were executed. 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.

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.

800
: No input parameters were specified for this check.

801
: Missing `CHECK(`, which is required on `S HZSPRINT` command.

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.

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 11.

1201
: The system could not open the specified SYSOUT data set. Check the SYSOUT data set requirements in “Set up the HZSPRINT utility” on page 8.

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 -1603
: Internal error. Contact the IBM Support Center.

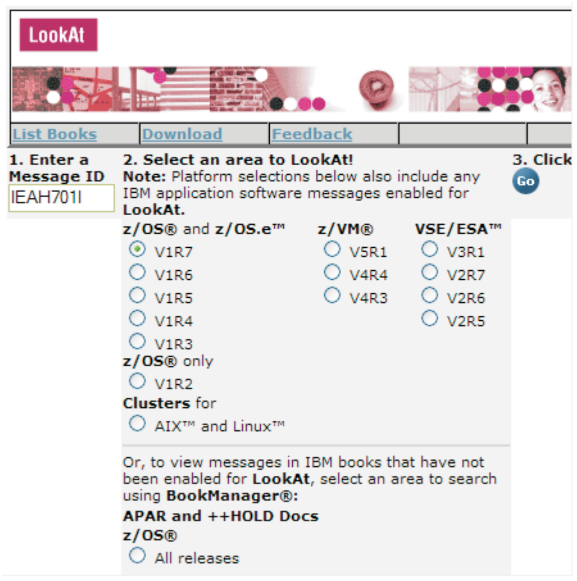
Finding check message documentation with LookAt

To find check message documentation 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.



1. Enter a Message ID: IEAH701I

2. Select an area to LookAt!
Note: Platform selections below also include any IBM application software messages enabled for **LookAt.**

z/OS® and z/OS.e™: [✓ V1R7] V1R6 V1R5 V1R4 V1R3
z/VM®: V5R1 V4R4 V4R3
VSE/ESA™: V3R1 V2R7 V2R6 V2R5
z/OS® only: V1R2
Clusters for: AIX™ and Linux™

Or, to view messages in IBM books that have not been enabled for **LookAt**, select an area to search using **BookManager®:**
APAR and ++HOLD Docs
z/OS®: All releases

3. Click

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
- Deleting 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 37.

  MODIFY command. See "Making dynamic, temporary changes to checks" on page 36.
- **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 44.

Table 5. 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 • Displaying checks • Refreshing 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.	**SDSF or MODIFY** *hzsproc***,UPDATE**
Persistent changes See "Making persistent changes to checks" on page 44.	**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 37.
- The MODIFY command - See "Cheat sheet: examples of MODIFY ***hzsproc*** commands" on page 39.

You can :

- Take one time only actions against checks, such as:
  - Adding and deleting checks
  - Displaying checks
  - Refreshing checks
  - Running checks
- Update check values with changes that last until the next refresh 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. 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 IBM Health Checker for z/OS policies" on page 44.

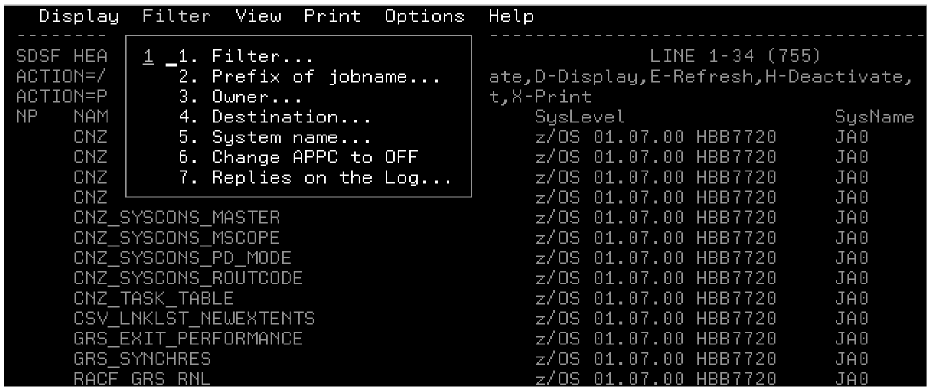
Using SDSF to manage checks

For IBM Health Checker for z/OS, SDSF provides the CK command to display and manage checks.

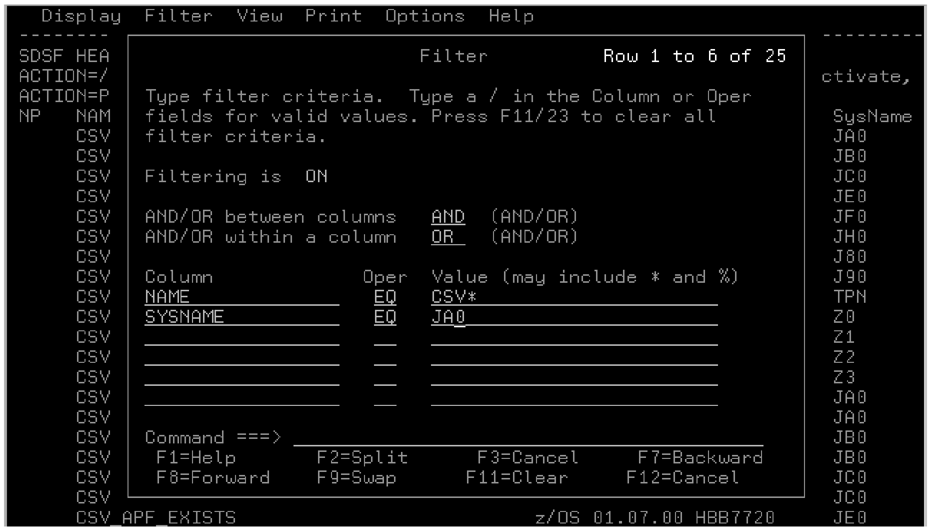
Using SDSF you can issue one-time or temporary actions against active checks, including:
- Viewing check output
- Changing check states
- Adding and deleting new checks
- Displaying check information
- Running checks
- Making temporary updates to check values in use, such as check interval, category, severity, or check message routing code. These updates will last until the check is refreshed.

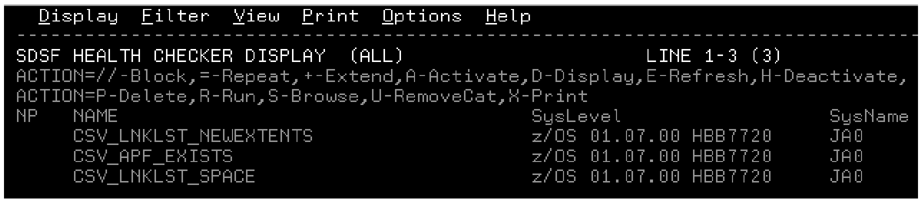
Like all SDSF primary displays, CK can also be accessed from a pull-down when SDSF is running as an ISPF dialog. To display the IBM Health Checker for z/OS CK action characters, use the **SET ACTION SHORT** or **SET ACTION LONG** command.

You can also:
- **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:



```
   Display  Filter  View  Print  Options  Help
 --------                                 -------------------------------------
 SDSF HEA   1 _1. Filter...                             LINE 1-34 (755)
 ACTION=/      2. Prefix of jobname...    ate,D-Display,E-Refresh,H-Deactivate,
 ACTION=P      3. Owner...                t,X-Print
 NP   NAM      4. Destination...              SysLevel                 SysName
      CNZ      5. System name...              z/OS 01.07.00 HBB7720     JA0
      CNZ      6. Change APPC to OFF          z/OS 01.07.00 HBB7720     JA0
      CNZ      7. Replies on the Log...       z/OS 01.07.00 HBB7720     JA0
      CNZ                                     z/OS 01.07.00 HBB7720     JA0
      CNZ_SYSCONS_MASTER                      z/OS 01.07.00 HBB7720     JA0
      CNZ_SYSCONS_MSCOPE                      z/OS 01.07.00 HBB7720     JA0
      CNZ_SYSCONS_PD_MODE                     z/OS 01.07.00 HBB7720     JA0
      CNZ_SYSCONS_ROUTCODE                    z/OS 01.07.00 HBB7720     JA0
      CNZ_TASK_TABLE                          z/OS 01.07.00 HBB7720     JA0
      CSV_LNKLST_NEWEXTENTS                   z/OS 01.07.00 HBB7720     JA0
      GRS_EXIT_PERFORMANCE                    z/OS 01.07.00 HBB7720     JA0
      GRS_SYNCHRES                            z/OS 01.07.00 HBB7720     JA0
      RACF_GRS_RNL                            z/OS 01.07.00 HBB7720     JA0
```


```
  Display  Filter  View  Print  Options  Help
 --------                                                                ---------
 SDSF HEA                          Filter         Row 1 to 6 of 25
 ACTION=/                                                                ctivate,
 ACTION=P   Type filter criteria.  Type a / in the Column or Oper
 NP   NAM   fields for valid values. Press F11/23 to clear all            SysName
      CSV   filter criteria.                                              JA0
      CSV                                                                 JB0
      CSV   Filtering is  ON                                              JC0
      CSV                                                                 JE0
      CSV   AND/OR between columns   AND  (AND/OR)                        JF0
      CSV   AND/OR within a column   OR   (AND/OR)                        JH0
      CSV                                                                 J80
      CSV   Column              Oper  Value (may include * and %)         J90
      CSV   NAME                 EQ   CSV*                                TPN
      CSV   SYSNAME              EQ   JA0                                 Z0
      CSV                                                                 Z1
      CSV                                                                 Z2
      CSV                                                                 Z3
      CSV                                                                 JA0
      CSV                                                                 JA0
      CSV   Command ===>                                                  JB0
      CSV    F1=Help     F2=Split      F3=Cancel     F7=Backward          JB0
      CSV    F8=Forward   F9=Swap      F11=Clear     F12=Cancel           JC0
      CSV                                                                 JC0
      CSV_APF_EXISTS                           z/OS 01.07.00 HBB7720      JE0
```


```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF HEALTH CHECKER DISPLAY  (ALL)                    LINE 1-3 (3)
 ACTION=//-Block,=-Repeat,+-Extend,A-Activate,D-Display,E-Refresh,H-Deactivate,
 ACTION=P-Delete,R-Run,S-Browse,U-RemoveCat,X-Print
 NP   NAME                                 SysLevel                  SysName
      CSV_LNKLST_NEWEXTENTS                z/OS 01.07.00 HBB7720     JA0
      CSV_APF_EXISTS                       z/OS 01.07.00 HBB7720     JA0
      CSV_LNKLST_SPACE                     z/OS 01.07.00 HBB7720     JA0
```

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

- **Sort** checks reporting exceptions in descending order using SDSF command `SORT RESULT D`.
- **Display** checks using the D or DL commands next to the check.

You can get sysplex-wide information about checks using SDSF's server and WebSphere MQ.

For complete information on the SDSF CK command, see the following:

- See the information on security for the CK function in *z/OS SDSF Operation and Customization*.
- **SDSF online help** for information about the columns and all the functions related to the CK panel, such as action characters, overtypeable columns, and commands.
- See how to customize CK columns in *z/OS SDSF Operation and Customization*.

Using the MODIFY *hzsproc* command to manage checks

MODIFY (F *hzsproc*) commands are useful for making dynamic, temporary changes to checks. See "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc* command" on page 53 for complete syntax information. In this section, we'll cover the following:

- "Cheat sheet: examples of MODIFY ***hzsproc*** commands" on page 39
- "Why does my check reappear after I delete it? Understanding delete processing" on page 40
- "But my check doesn't reappear after ADDNEW - what happened to it?" on page 41

- “How can I delete checks while IBM Health Checker for z/OS is terminating?” on page 42
- “Using the category filter to manage checks” on page 42

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 “Syntax and parameters for HZSPRMxx and MODIFY *hzsproc* command” on page 53 for complete syntax information.

Table 6. 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 42. 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 28.
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 28.
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 40.
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.

Table 6. F hzsproc command examples (continued)

Action	Command example
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 301). 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 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 routines 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.

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 on page 43.

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 IBM Health Checker for z/OS policies."
- 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 "ADD or ADDREPLACE CHECK parameters" on page 66.
- 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 parameter" on page 61.

Creating 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.

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.

The syntax of the policy statements is as follows:

```
{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 syntax of the UPDATE options you can use to update check values on a policy statement is as follows:

```
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}]
        [,PARM=parameter,REASON=reason,DATE={date | (date,NOCHECK)}]
        [,REASON=reason]
        [,REPCAT=(cat1[,cat2[,...,cat16]])]
        [,REMCAT=(cat1[,cat2[,...,cat16]])]
        [,ROUTCODE=(routcode1,...,routcoden)]
        [,SEVERITY={HIGH|MEDIUM|LOW|NONE}]
        [,WTOTYPE={CRITICAL|EVENTUAL|INFORMATIONAL|HARDCOPY|NONE}]
|       [,REXXTIMELIMIT=timelimit]
```

- You can filter the update or deletions requested on a policy statement by check, category, or owning exit routine.

See "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc* command" on page 53 for complete syntax details.

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 48 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 COMMANDxx parmlib member
   - The IBM Health Checker for z/OS procedure, *hzsproc*

   See "Specifying the HZSPRMxx members you want the system to use" on page 51.

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 50
- "Policy statement examples" on page 50
- "Can I create policy statements using the MODIFY command?" on page 51
- "Specifying the HZSPRMxx members you want the system to use" on page 51

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 48
- 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:

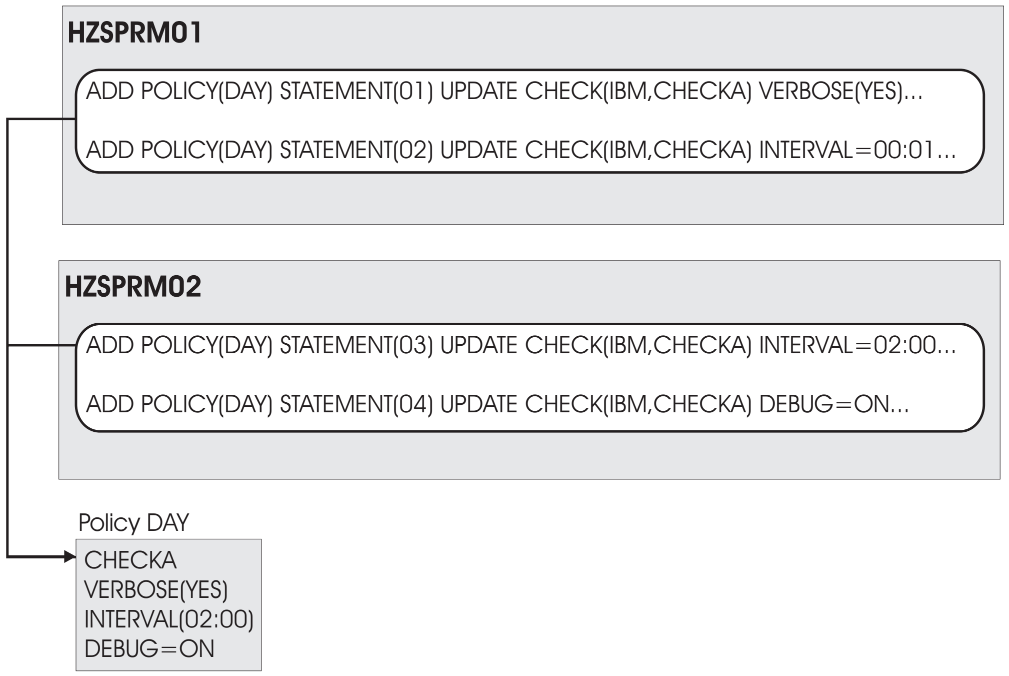

HZSPRM01
ADD POLICY(DAY) STATEMENT(01) UPDATE CHECK(IBM,CHECKA) VERBOSE(YES)...
ADD POLICY(DAY) STATEMENT(02) UPDATE CHECK(IBM,CHECKA) INTERVAL=00:01...
HZSPRM02
ADD POLICY(DAY) STATEMENT(03) UPDATE CHECK(IBM,CHECKA) INTERVAL=02:00...
ADD POLICY(DAY) STATEMENT(04) UPDATE CHECK(IBM,CHECKA) DEBUG=ON...
Policy DAY
CHECKA
VERBOSE(YES)
INTERVAL(02:00)
DEBUG=ON

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 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:

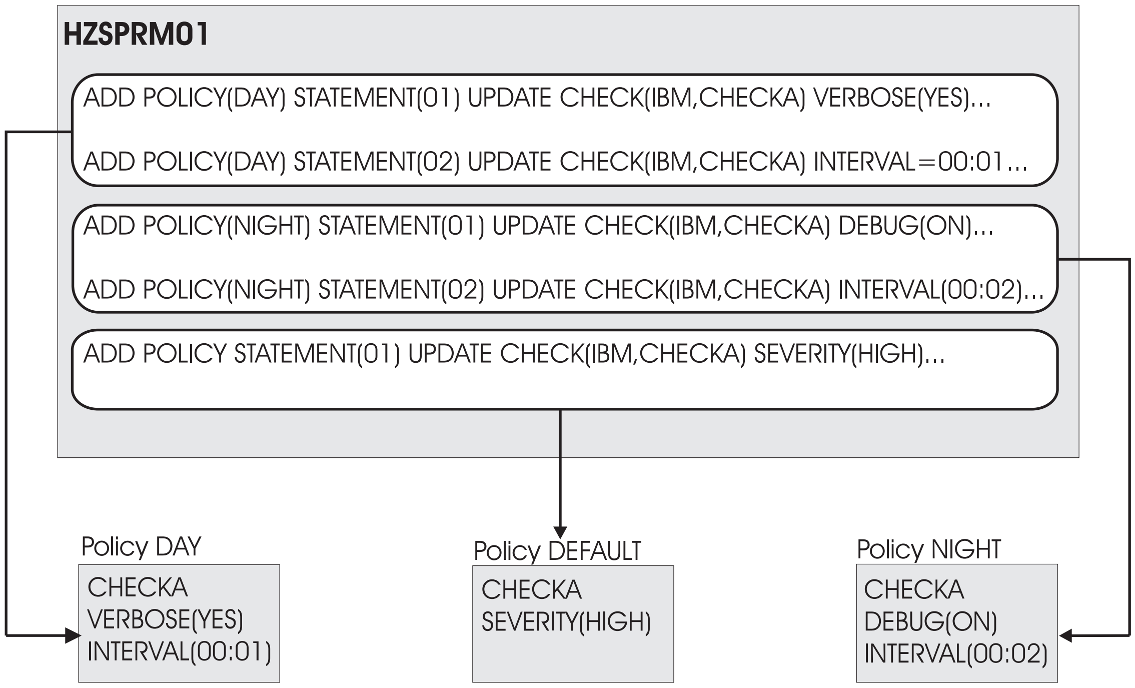

HZSPRM01
ADD POLICY(DAY) STATEMENT(01) UPDATE CHECK(IBM,CHECKA) VERBOSE(YES)...
ADD POLICY(DAY) STATEMENT(02) UPDATE CHECK(IBM,CHECKA) INTERVAL=00:01...
ADD POLICY(NIGHT) STATEMENT(01) UPDATE CHECK(IBM,CHECKA) DEBUG(ON)...
ADD POLICY(NIGHT) STATEMENT(02) UPDATE CHECK(IBM,CHECKA) INTERVAL(00:02)...
ADD POLICY STATEMENT(01) UPDATE CHECK(IBM,CHECKA) SEVERITY(HIGH)...
Policy DAY
CHECKA
VERBOSE(YES)
INTERVAL(00:01)
Policy DEFAULT
CHECKA
SEVERITY(HIGH)
Policy NIGHT
CHECKA
DEBUG(ON)
INTERVAL(00:02)

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 48, 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

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 have applied using the following MODIFY command:

```
F hzsproc,DISPLAY,CHECKS(*,*),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.

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 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)
```

- When date is specified with NOCHECK, the policy statement is applied to the matching check or checks.
- When date is specified without NOCHECK, **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.

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.

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 40 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

HZSPRMxx members can be specified when starting IBM Health Checker for z/OS or dynamically to direct the system to process the corresponding parmlib member.:

- 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 default member, HZSPRM00.

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

```
//HZSPROC JOB JESLOG=SUPPRESS
//HZSPROC  PROC HZSPRM='00'
//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)`. You can also use an ADD or REPLACE command in place of SET, because the command is issued during the initialization phase.

- 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.

Syntax and parameters for 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 54 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. We recommend that you use the **MODIFY** command for these:
  - UPDATE
  - ACTIVATE
  - DEACTIVATE
- 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 only in an **HZSPRMxx** parmlib member. See "Creating IBM Health Checker for z/OS policies" on page 44.
- 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 "ADD or ADDREPLACE CHECK parameters" on page 66.

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 301 for check default information.

Syntax rules for HZSPRMxx

Follow the rules in General syntax rules 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 ″*/″.

Statements and parameters

```
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
  }

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}]
        [,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}]


{ADD | ADDREPLACE},CHECK=(check_owner,check_name)
                   ,{CHECKROUTINE=routinename
                    | EXEC=execname
                       ,REXXHLQ=hlq
                      [,REXXTIMELIMIT=timelimitvalue]
                   {  [,REXXTSO=YES]
                  | [,REXXTSO=NO
                       [,REXXIN={NO | YES}
                          ]
                   }
                    }
                    ,MESSAGETABLE=msgtablename
                    ,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]

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 42. The CATEGORY filters can be one of the following:

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.

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."

`ACTIVATE, filters` is equivalent to the UPDATE,*filters*,ACTIVE command. See "UPDATE ACTIVE and INACTIVE parameters " on page 62.

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 40.

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” on page 57.

DEACTIVATE is the same as the UPDATE,*filters*,INACTIVE command. See “UPDATE ACTIVE and INACTIVE parameters ” on page 62.

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 “filters” on page 57.

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 40 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" on page 57.

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 "Example of DISPLAY SUMMARY message output" on page 72. 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" on page 73) 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”” on page 74.

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” on page 74) or HZS0204I (“Example of DISPLAY POLICY SUMMARY message output” on page 74) for SUMMARY.

POLICY=*policyname*
Specifies the 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 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” on page 74), 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.

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" on page 57).

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 46.

You must specify filter CHECK=(*check_owner,check_name*) with REFRESH. Other filters are optional. See "filters" on page 57.

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" on page 57.

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}]
       [,EINTERVAL={SYSTEM|HALF|hhh:mm}]
       [,PARM=parameter,REASON=reason,DATE={date | (date,NOCHECK)}]
       [,REASON=reason]
|      [,REPCAT=(cat1[,cat2[,...,cat16]])]
|      [,REXXTIMELIMIT=timelimit]
       [,REMCAT=(cat1[,cat2[,...,cat16]])]
       [,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" on page 57.

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 26. These parameters are equivalent to "ACTIVATE parameter" on page 57 and "DEACTIVATE parameter" on page 58.

ADDCAT=(*cat1***,...,***catn)*
ADDCAT adds specified checks into each of the listed categories.

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)`.

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 301 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 301 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. (See "WTOTYPE" on page 65 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 Valid combinations for 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, kicked off by refresh processing.
- *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 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 specified exception interval time starts ticking away when a check finishes running.

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.

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. 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. 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.

WTOTYPE={<u>CRITICAL</u>|EVENTUAL|INFORMATIONAL|HARDCOPY|NONE}

WTOTYPE specifies the type of WTO that should be issued when the check finds an exception. This parameter includes all WTOs issued by a check.

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
                    ,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]
```

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.
- 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 136 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. A value of 0 is treated the same as no time limit. The default is NO_TIMELIMIT.

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 131 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 131 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 136 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*
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. The message table must be built using HZSMSGEN. The message table module must be in an APF-authorized library.

SEVERITY={HIGH | MEDIUM | LOW | NONE}
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.
- 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.

INTERVAL={ONETIME|*hhh*:*mm*}
Required parameter **INTERVAL** specifies how often the check you are adding or replacing should run:

- **ONETIME** specifies that the check should run only once, kicked off by refresh processing.
- *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 specified interval time starts ticking away when a check finishes running.

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 26. These parameters are equivalent to “ACTIVATE parameter” on page 57 and “DEACTIVATE parameter” on page 58.

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 specified exception interval time starts ticking away when a check finishes running.

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.

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.

| **REPLACE,PARMLIB**

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

| Replaces the list of HZSPRMxx parmlib members with the specified parmlib
| member 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
```

Activates the specified policy, making 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 48.

{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” on page 72 and “DATE parameter” on page 72 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 name of the policy in which you are adding or replacing policy statements. 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. 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″” on page 62.

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” on page 57.

REASON=*reason*
See the “REASON parameter” on page 64.

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*
The name of the policy for which you are removing a policy statement. 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.

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_MASTER               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  10.27.07 HZS INFORMATION   FRAME LAST   F      E   SYS=SY1
OUTSTANDING EXCEPTIONS-13:
  (SEVERITY NONE: 0   LOW: 8  MEDIUM: 2  HIGH: 3)
ELIGIBLE CHECKS: 33  (CURRENTLY RUNNING: 0)
INELIGIBLE CHECKS: 1   DELETED CHECKS: 0
ASID: 002C   LOG STREAM: - NOT DEFINED
```

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.

Here is a list of some of the kinds of things you can write a check to look for:

- 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:

- 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 147 and “HZSADDCK macro — HZS add a check” on page 218.
  - Include information in your check output about why the check user is getting an exception message for a default or common valid setting. See Chapter 10, “Creating the message input for your check,” on page 155.

For those of you who would like to develop your own checks, there are 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/`

You can use these checks, which you can install and execute, as a good reference and starting point for developing checks. The samples include:
- A **local check routine** that includes two checks: a check that runs one time and a check that runs at intervals.
- An **HZSADDCHECK exit routine** to add local checks to the system.
- A **message input data set** for generating the message table.

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
- Display output, such as the existing Coupling Facility Structure/Status report, which can help you identify checks.

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 35.
- **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 37.
  – The MODIFY (F) *hcproc*,REFRESH operator command. See "Syntax and parameters for HZSPRMxx and MODIFY *hzsproc* command" on page 53.

What kind of check do you want to write?

You can develop the following basic types of checks:

- "Local checks"
- "Remote checks" on page 82
- "REXX checks" on page 83

Each type of check issues the same kind of messages for IBM Health Checker for z/OS users.

Local checks

Local checks are written in assembler and run in the IBM Health Checker for z/OS address space, *hzsproc*. 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:

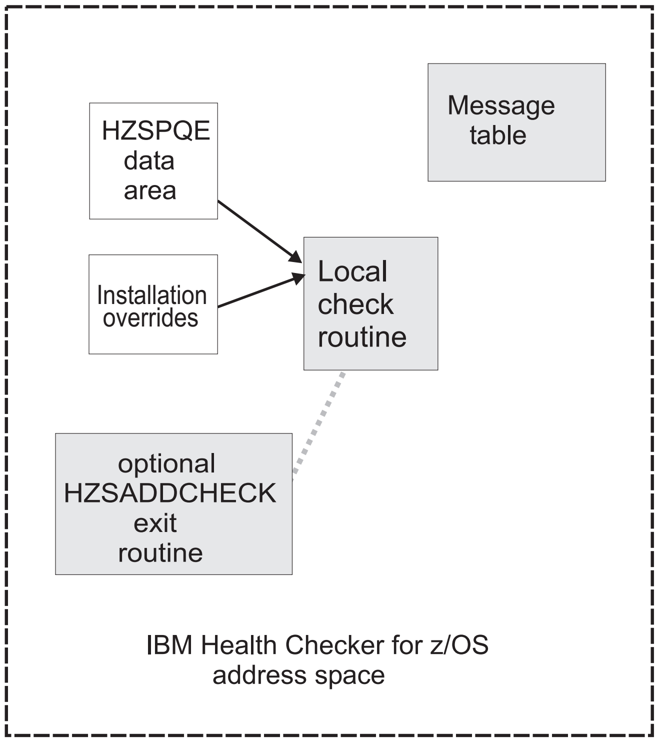

Message table
HZSPQE data area
Installation overrides
Local check routine
optional HZSADDCHECK exit routine
IBM Health Checker for z/OS address space

Figure 5. The parts of a local check

Remote checks

Remote checks are written in assembler and 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 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:

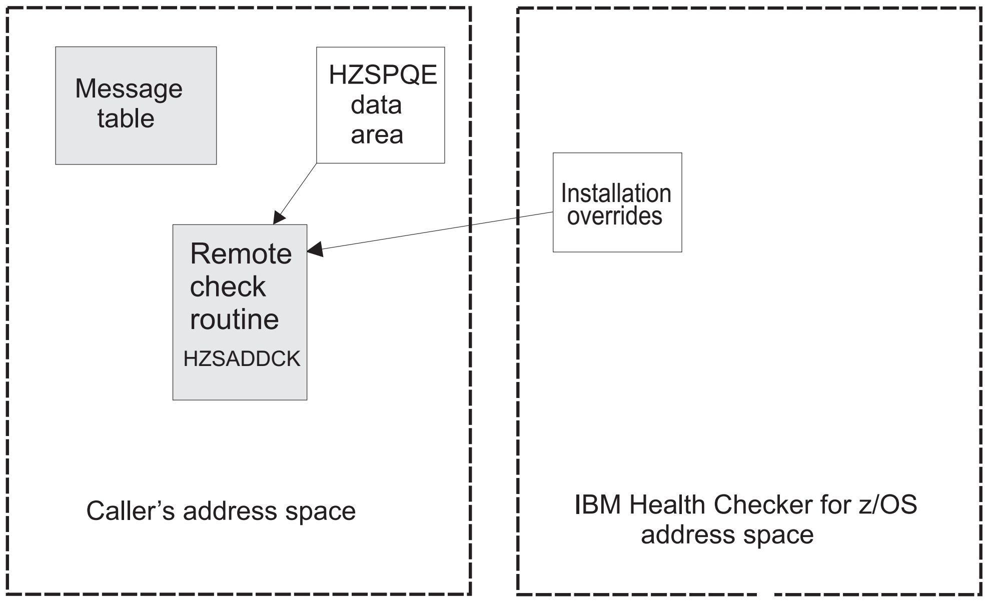


Figure 6. The parts of a remote check

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 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.

The following figure shows the parts of a **REXX** check. The shaded items show the parts that a check developer must provide:

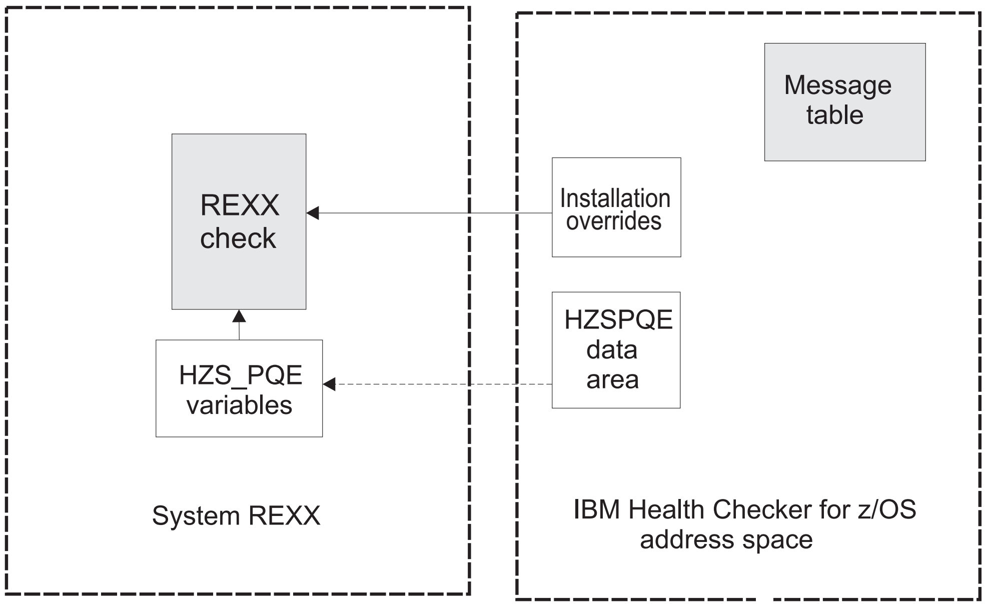


Figure 7. The parts of a REXX check

Summary of checks - differences and similarities

The following table shows some of the differences between local and remote checks:

Table 7. Summary local, remote, and REXX checks

Local checks	Remote checks	REXX checks
How do I know which type of check to write?		
Write an local check to look at system storage or use assembler services. 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.) Write a local check to look at system storage or use assembler services. A local check has the advantage of receiving IBM Health Checker for z/OS recovery support.	Write a remote check to look at system storage or use assembler services. 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.
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?		

Table 7. Summary local, remote, and REXX checks (continued)

Local checks	Remote checks	REXX checks
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?		
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 87	Yes - see Chapter 7, "Writing remote check routines," on page 109	No - see Chapter 8, "Writing REXX checks," on page 131
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
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.

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 87.
   - For a remote check, see Chapter 7, “Writing remote check routines,” on page 109.
   - If you are writing a REXX check, see Chapter 8, “Writing REXX checks,” on page 131.
2. Create a message table for the check output. The message table defines the check output messages issued by the check routine. See Chapter 10, “Creating the message input for your check,” on page 155.
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 301.

Chapter 6. Writing local check routines

A local check runs in the IBM Health Checker for z/OS address space, *hzsproc*. To learn about the differences between local and remote checks and deciding which type you want to write, see “Remote checks” on page 82.

In this chapter, we'll cover the following:

- “Sample local checks”
- “Local check routine basics”
- “Defining a local check to IBM Health Checker for z/OS” on page 89
- “Programming considerations” on page 89
- “Sample reentrant entry and exit linkage” on page 91
- “Using the check parameter parsing service (HZSCPARS)” on page 92
- “Using the HZSPQE data area in your local check routine” on page 92
- “Function codes for local check routines” on page 93
- “Issuing messages in your check routine with the HZSFMSG macro” on page 95
- “Defining the variables for your messages” on page 97
- “The well-behaved local check routine - recommendations and recovery considerations” on page 103
- “Debugging checks” on page 106
- Chapter 9, “Writing an HZSADDCHECK exit routine,” on page 147

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 - assembler samples in SYS1.SAMPLIB:

- **HZSSADCK** - Sample HZSADDCHECK exit routine.
- **HZSSCHKR** - Sample local check routine.
- **HZSSMSGT** - Sample message input.

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 (UID(0)).

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). The HZSPQE for a check contains:

- The defaults defined for the check.
- A 2K check work area and a pointer to a 4K dynamic work area.
- 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 92.

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 (“Sample reentrant entry and exit linkage” on page 91) 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 92.
3. The meat of the check - checking for potential problems on a system.
4. Issuing messages using the HZSFMSG macro (“Issuing messages in your check routine with the HZSFMSG macro” on page 95)
5. Defining your message variables in the HZSMGB data area (“Defining the variables for your messages” on page 97)

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.

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" on page 66.
- 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 147. 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 93
  - "Release codes for remote check routines" on page 117

  See "The well-behaved local check routine - recommendations and recovery considerations" on page 103.
- 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 103.

Sample reentrant entry and exit linkage

The following sample shows how to code the reentrant entry and exit linkage for a check routine called THECHECK:

```
THECHECK CSECT
THECHECK AMODE 31
THECHECK RMODE 31
         USING THECHECK,15
* The check routine is not required to save regs
         LA    12,STATAREA
         DROP  15
         USING STATAREA,12
         LR    15,0              Copy dynamic area for use
         USING DYNAREA,15
         ST    14,RETURN_ADDR    Save return address
* Chain saveareas (even though we did not save regs)
         ST    0,8(,13)          Chain new area to previous
         ST    13,4(,15)         Chain previous area to new
         LR    13,15             4K dynamic area address
         DROP  15
         USING DYNAREA,13
         L     2,0(,1)           Access PQE address
         USING HZSPQE,2
*
* Code should use relative branch for maximum addressability
*
*
* Examine PQE for entry code, check function, PQEChkWork, etc.
* Write messages using HZSFMSG
*
* The check routine is not required to restore regs
         L     14,RETURN_ADDR    Restore return reg
         BR    14                Return
STATAREA DS    0D                Static area
         LTORG                   literals
         HZSPQE ,
* Room for 4096 bytes using the input area in reg 0
DYNAREA  DSECT
SAVEAREA DS    CL72
RETURN_ADDR DS A
         HZSFMSG MF=(L,FMSGL),PLISTVER=MAX  HZSFMSG list form
DYNAREA_LEN EQU *-DYNAREA
         END
```

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. 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 236 for explanations of all the REASON=PARS*xxxx* values.

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 289 for complete information.

Note that your check routine must still issue the HZSFMSG REQUEST=STOP request when HZSCPARS it finds a parameter error - see ""Check function code for local checks"" on page 94 and ""INITRUN and RUN release codes for remote checks"" on page 118.

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 and any installation overrides to those defaults. 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 8. 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 93. This bit applies only to local checks.

Table 8. Important fields in the HZSPQE data area for a local check routine (continued)

Field name	Meaning
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 117.
PqeChkInfo section - contains the defaults defined in the HZSADDCHECK exit routine for the check	
PQE_Entry_Code	Contains the identifer (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.
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.	

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:

Local check routine

Table 9. 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.	
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 236. 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 check routine with the HZSFMSG macro" on page 95.	• 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.	

Table 9. Summary of function codes for local checks (continued)

Function	Check and system actions	When is it invoked?
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.	

Issuing messages in your 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 236). 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 input data set, see Chapter 10, "Creating the message input for your check," on page 155.
- 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 97.

You can issue the following kinds of messages in your check routine:

- Exception messages and other check results messages (CHECKMSG request). For an overview of the various message types, see Table 17 on page 174.
- 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 256.

For information on coding the message texts and explanation for messages, see Chapter 10, “Creating the message input for your check,” on page 155.

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.

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. 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 98.
- **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 101.

Figure 13 on page 157 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 input data set, 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 100 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 100 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

Important fields in the HZSMGB data area include:

Table 10. Important 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.

Table 10. Important fields in the HZSMGB data area for check message variables (continued)

Field name	Meaning
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 102 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 fields in the HZSMGB data area include:

Table 11. Important fields in the HZSMGB1 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_MsgInsertDesc_Length	The length of the variable. For a null variable, use a length of zero.

Table 11. Important fields in the HZSMGB1 data area for check message variables (continued)

Field name	Meaning
MGB1_MsgInsertDesc_Addr	The address of the variable. For a null variable, you need not set this field.

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 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 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 check routine with the HZSFMSG macro" on page 95 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 161

  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 106.

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 218.

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 91.

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 ″Preallocate Data Sets for SYSMDUMP 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 "HZSFMSG ABEND Codes" on page 252 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 82.

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 82.

In this chapter, we'll cover the following:

- “Sample checks”
- “Remote check routine basics” on page 110
- “Programming considerations” on page 111
- “Preparing for check definition - making sure IBM Health Checker for z/OS is up and running” on page 112
- “Allocate a pause element token using IEAVAPE” on page 113
- “Issue the HZSADDCK macro to define check defaults to IBM Health Checker for z/OS” on page 113
- “Pause the remote check routine with IEAVPSE” on page 115
- “Using HZSCHECK REQUEST=OPSTART and REQUEST=OPCOMPLETE to communicate check start and stop to IBM Health Checker for z/OS” on page 116
- “Using the check parameter parsing service (HZSCPARS)” on page 116
- “Using the HZSPQE data area in your remote check routine” on page 116
- “Release codes for remote check routines” on page 117
- “Issuing messages in your check routine with the HZSFMSG macro” on page 119
- “Defining the variables for your messages” on page 122
- “Recommendations and recovery considerations for remote checks” on page 127
- “Debugging checks” on page 129

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 - assembler samples in SYS1.SAMPLIB:

- **HZSSMSGT** - Sample message input.
- **HZSSRCHK** - Sample remote check routine.

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. 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 92.

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 92.
2. The meat of the check - checking for potential problems on a system.
3. Issuing messages using the HZSFMSG macro (“Issuing messages in your check routine with the HZSFMSG macro” on page 95)
4. Defining your message variables in the HZSMGB data area (“Defining the variables for your messages” on page 97)

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 217.

Requirements

- Minimum authorization for your remote check task is 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.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 117.
- 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. 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.

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 check defaults 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.

Remote check routine

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 301.)
- **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 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.

Example of the 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 218.

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 275.

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. 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 236 for explanations of all the REASON=PARS*xxxx* values.

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 289 for complete information.

Note that your check routine must still issue the HZSFMSG REQUEST=STOP request when HZSCPARS it finds a parameter error - see “″Check function code for local checks″” on page 94 and “″INITRUN and RUN release codes for remote checks″” on page 118.

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 12. 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 identifer (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.
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

When IBM Health Checker for z/OS unpauses a remote check task that issued the IEAVPSE service, the remote check receives a 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.

For remote checks, the 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.

- 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 13. 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 236. • 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..	• Refresh • When a check is added • When a check transitions to the active enabled state • At specified check interval • When a check parameter changes
	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.	
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.	A user deletes the check using F *hzsproc* or the HZSPRMxx parmlib member.
	What does the system do? The system stops calling the check.	

Table 13. 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.	• Refresh
	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.	
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.	• When the IBM Health Checker for z/OS address space goes down
	What does the system do? The system stops calling the check.	
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.	• When the IBM Health Checker for z/OS address space is restarted
	What does the system do? Continues to initialize IBM Health Checker for z/OS.	
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.	• 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 stops calling the check.	

Issuing messages in your 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 236). 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 input data set, see Chapter 10, "Creating the message input for your check," on page 155.

- 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 97.

You can issue the following kinds of messages in your check routine:
- Exception messages and other check results messages (CHECKMSG request). For an overview of the various message types, see Table 17 on page 174.
- 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 check defaults to IBM Health Checker for z/OS” on page 113.
- 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 256.

For information on coding the message texts and explanation for messages, see Chapter 10, "Creating the message input for your check," on page 155.

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.

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. 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 123.
- **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 125.

Figure 13 on page 157 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 input data set, 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 10 on page 124 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 10 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

Important fields in the HZSMGB data area include:

Table 14. Important 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 11 on page 126 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 fields in the HZSMGB data area include:

Table 15. Important fields in the HZSMGB1 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_MsgInsertDesc_Length	The length of the variable. For a null variable, use a length of zero.

Table 15. Important fields in the HZSMGB1 data area for check message variables (continued)

Field name	Meaning
MGB1_MsgInsertDesc_Addr	The address of the variable. For a null variable, you need not set this field.

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 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 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 check routine with the HZSFMSG macro” on page 95 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 161

  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.”

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 218.

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 111.

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 ″Preallocate Data Sets for SYSMDUMP 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 “HZSFMSG ABEND Codes” on page 252 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*

In this chapter, we'll cover the following:
- "Sample REXX checks"
- "REXX check basics"
- "Using input data sets in a TSO-environment REXX check" on page 135
- "Using REXXIN data sets" on page 135
- "Using REXXOUT data sets" on page 136
- "Defining a REXX check to IBM Health Checker for z/OS" on page 139
- "Issuing messages in your REXX check with the HZSLFMSG function" on page 141
- "The well-behaved REXX check - recommendations and recovery considerations" on page 144
- "Debugging REXX checks" on page 146

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:
- - Sample REXX checks.
- **HZSSMSGT** - Sample message input, which is common to all check types.

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.

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. 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 203.
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 236. HZSLFMSG 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 139.

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.

- 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 199 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 16. 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 identifer (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.

See “HZSLSTRT function” on page 200 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 in your REXX check with the HZSLFMSG function” on page 141
  - 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.

  You use the HZSLFMSG function to issue a message and define variables, but you define the actual text and explanation for your REXX check messages in your message input data set - see “HZSLFMSG function” on page 203 and Chapter 10, “Creating the message input for your check,” on page 155.
- 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 199 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 139.

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 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 203 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 203 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 “ADD or ADDREPLACE CHECK parameters” on page 66.
3. Use the ADD,PARMLIB command to add the new parmlib member containing the REXX check definition. For example:

   ```
   F hzsproc,ADD,PARMLIB=xx
   ```

You can also 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 147. The HZSADDCHECK exit routine describes the information about your REXX check or checks.

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, 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*) - 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. A value of 0 is treated the same as no time limit. The default is that this is no time limit.
- 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, 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 again.
- Stop and start IBM Health Checker for z/OS to delete the check definition.
- If you make a mistake when defining a check in an HZSADDCHECK exit routine, you must delete the check (for example, by creating a policy statement that deletes the check) and then delete the erroneous exit routine using SETPROG. You can then add the corrected HZSADDCHECK exit routine again.

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.

Issuing messages in your REXX check with the HZSLFMSG function

This section covers issuing messages from your REXX check. For information on coding the message texts and explanation for messages, see Chapter 10, “Creating the message input for your check,” on page 155.

REXX check messages are important because they report the results of the check to an installation. 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.

See “Planning your check messages” on page 159.

To issue a message with check results in your REXX check, you must use the HZSLFMSG function (“HZSLFMSG function” on page 203).

You'll use the HZSLFMSG function to:

- Issue one of the following requests:
  – **HZSLFMSG_REQUEST="CHECKMSG" request** - 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 input data set. See Chapter 10, “Creating the message input for your check,” on page 155.
  – **HZSLFMSG_REQUEST="HZSMSG" request** - 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.
  – **HZSLFMSG_REQUEST="STOP" request** - Indicates that the system should stop calling this check . The message text is provided by IBM Health Checker for z/OS.
- Indicate the message number you want to issue with a HZSLFMSG_MESSAGENUMBER=*msgnum* input variable.
- 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.

The following example shows how a REXX check uses the HZSLFMSG function to issue an exception message.

```
/*************************************** ***************************/
/* Build and write exception message                               */
/*                                                                 */
/* In the sgml source HZSSMSGT message number 1, has 4 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 in the insert should be compatible with the class attribute of the associated message variable in the message input data set. A class attribute of hex, decimal or timestamp in the message input data set will treat the insert data as a hexadecimal string.
- Variable HZSLFMSG_INSERT.1 expects to receive hexadecimal data. In the message input data set, variable 1 has a class attribute of hex:

  ```
  <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 input data set has a class attribute of:

```
 <mv class="decimal">variable 1</mv>
```

In that case, the REXX check might use the following HZSLFMSG input variable:

```
HZSLFMSG_INSERT.1 = '0A''X   /* The decimal value 10 is displayed */
```

In general, the REXX values you use will be text and usually do not require additional translation.

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" on page 142 for an example of how to issue an exception message from a REXX check.

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
```

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 1 (ABA-AOM)*.
- 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 136.

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 136.

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.

| 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
| "ADD or ADDREPLACE CHECK parameters" on page 66.

| You cannot add remote checks to the system with a HZSADDCHECK exit routine.
| See "Issue the HZSADDCK macro to define check defaults to IBM Health Checker
| for z/OS" on page 113.

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 routine

For a sample HZSADDCHECK exit routine, look for HZSSADCK in SYS1.SAMPLIB.

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 218** in your HZSADDCHECK exit routine to describe your check. This section also includes "Examples" on page 234.
- **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 301.)

- **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*) - 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. A value of 0 is treated the same as no time limit. The default is that this is no time limit.

- 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.

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 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 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 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 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 153.
- 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" on page 152
  - "Using a routine to add checks to the system dynamically" on page 152

  You can also simply define the check defaults and values directly in an HZSPRMxx parmlib member and bypass the HZSADDCHECK exit routine entirely. See "ADD or ADDREPLACE CHECK parameters" on page 66.

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 36.
- 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 28.

Using operator commands to add checks to the system dynamically

1. Make the check routine, HZSADDCHECK exit routine, 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, 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 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

The following example shows the logic of a routine that:

- **1** Adds exit routine, HCEXIT, to the HZSADDCHECK exit.
- **2** Issues the HZSCHECK service to call the HZSADDCHECK exit to run the exit routine.

```
 CSVDYNEX REQUEST=ADD,
          1
          EXITNAME==CL16'HZSADDCHECK',
          MODNAME==CL8'HCEXIT',
          MESSAGE=ERROR,
          RETCODE=RC,
          RSNCODE=RS

 IF rc < 8 THEN
2
HZSCHECK REQUEST=ADDNEW,
          RetCode=RC,
          RsnCode=RS
```

Debugging HZSADDCHECK exit routine abends

For an HZSADDCHECK exit routine abend, 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 147, 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 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 brings the system down with it, (GRS or RACF, for example), you do not need to do any deletion or check clean up. However, if your product or component does occasionally come up and down (USS, for example) you might want to delete the checks and the HZSADDCHECK exit routine as you come down.

The following example shows the kind of code you would add to delete the check HZSADDCHECK exit routine and delete the check:

```
CSVDYNEX REQUEST=DELETE,
          EXITNAME=HZSADDCHECK,  /* Health Checker Name */
          MODNAME=IBMHCADC,      /* Delete exit routine */
          FORCE=YES,
          RetCode=HCRetCode,
          RsnCode=HCRsnCode

HZSCHECK REQUEST=DELETE,       /* Tell Health Checker to  */
                               /* delete IBMABC checks    */
          CHECKOWNER=IBMABC,
          CHECKNAME=*
          RetCode=HCRetCode,
          RsnCode=HCRsnCode
```

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.

To code messages for each 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 159.
2. **Create a message input data set** that contains both message texts **and explanations** for checks. See “Creating the message input data set” on page 162.
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 185
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 input data set” on page 187.
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. The message table is loaded into IBM Health Checker for z/OS in private.
6. **From your check routine:**
   - **Use the HZSFMSG macro to issue messages** . See “Issuing messages in your check routine with the HZSFMSG macro” on page 95.
   - **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 97. You can have up to 20 message variables per message, each used one time only.
   - **Remote checks must load the message table into storage.**

The following figure shows how all the parts fit together in the process of creating a message table:

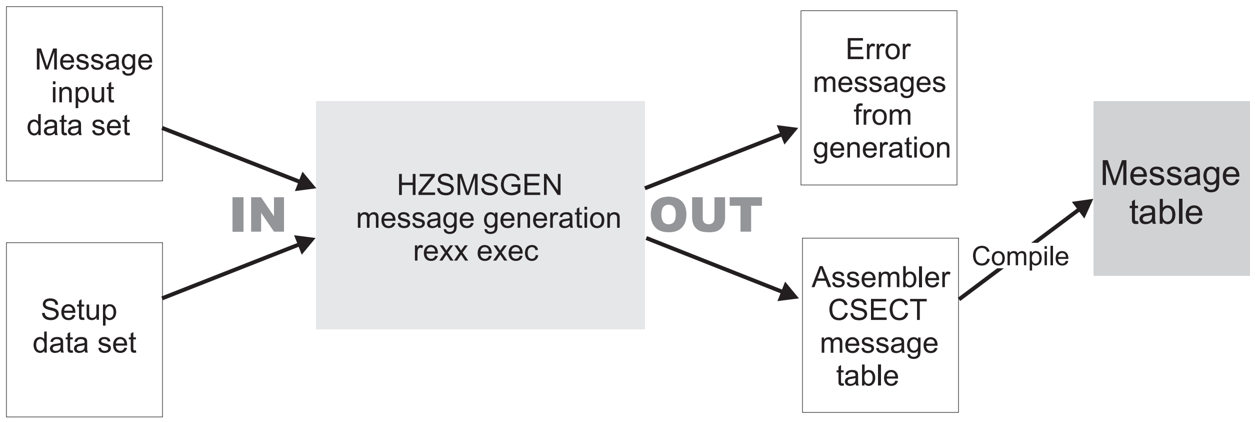

Message input data set
Setup data set
IN
HZSMSGEN message generation rexx exec
OUT
Error messages from generation
Assembler CSECT message table
Compile
Message table

Figure 12. Inputs and outputs for creating a complete message table

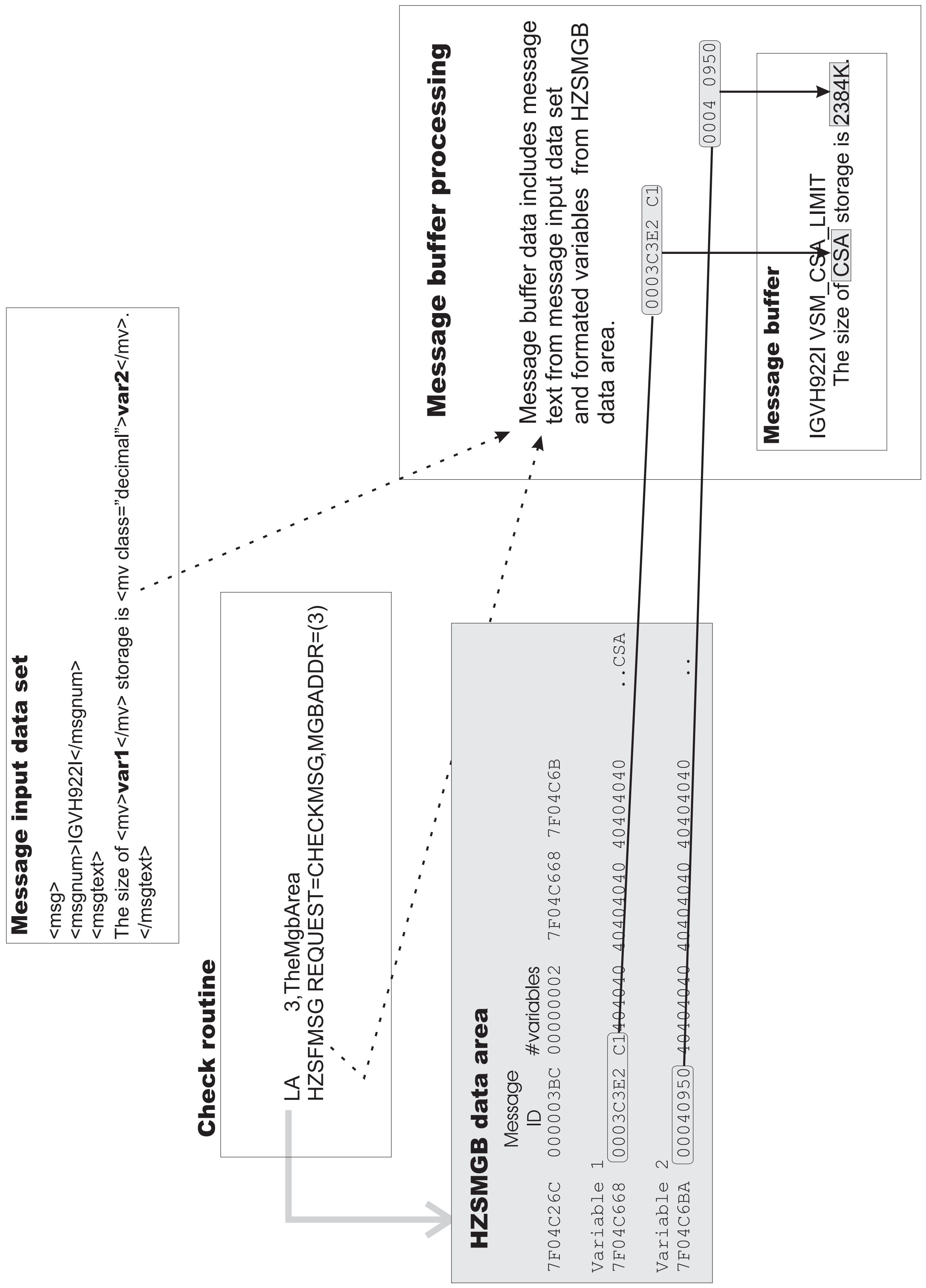
How messages and message variables are issued at check runtime

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 input data set describes the attributes of the variable, which determine how the variable is formatted. See “Variables for message text” on page 175.
- At runtime, when the HFSFMSG 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 13 on page 157 shows how messages with variables get resolved at check runtime.


Message input data set
Check routine
HZSMGB data area
Message buffer processing
Message buffer data includes message text from message input data set and formated variables from HZSMGB data area.
Message buffer

```
<msg>
<msgnum>IGVH922I</msgnum>
<msgtext>
The size of <mv>var1</mv> storage is <mv class="decimal">var2</mv>.
</msgtext>
```

```
LA        3,TheMgbArea
HZSFMSG REQUEST=CHECKMSG,MGBADDR=(3)
```

```
          Message
          ID        #variables
7F04C26C  000003BC  00000002   7F04C668  7F04C6B

Variable 1
7F04C668  0003C3E2  C1404040  40404040  40404040     ..CSA

Variable 2
7F04C6BA  00040950  40404040  40404040  40404040     ..
```

```
0003C3E2 C1
0004 0950
```

```
IGVH922I VSM_CSA_LIMIT
    The size of CSA storage is 2384K
```

Figure 13. 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"
- "Planning your information messages" on page 160
- "Planning your report messages" on page 160
- "Planning your debug messages" on page 161

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 103 and "Recommendations and recovery considerations for remote checks" on page 127 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 236.

You must decide the following when planning your messages:
- "Decide what release your check will run on" on page 161
- "Decide whether to translate your exception messages into other national languages" on page 161
- "Rely on IBM Health Checker for z/OS to issue basic check information for you" on page 162

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 22 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 message buffer.

See Table 20 on page 179 for the categories you'll need to include in your exception message details.

See “Exception message example” on page 163.

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 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 165.

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 166.

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 103 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 170.

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 input data set. So you must determine in advance whether:

- 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 172 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 172 for complete information.

Decide whether to translate your 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 “<msgtext>” on page 175. 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 exception messages at the same time you generate the message input data set CSECT using the HZSMSGEN exec, see “Support for translating messages to other languages” on page 191.

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″" on page 22 and "Exception message example" on page 163.
-
- 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 input data set

Sample message input data set

For a sample message input data set, see the IBM Health Checker for z/OS Web page:

`http://www-03.ibm.com/servers/eserver/zseries/zos/hchecker/`

In the message input data set, 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 input data set will be an interface which installations can automate.

This section covers the following:
- "Examples of message input" on page 163
- "Syntax of message input" on page 171
  - "Special formatting tags for the message input data set" on page 180
  - "How messages are formatted in the message buffer" on page 182
  - "Using symbols in the message input data set" on page 184

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 that the reported situation will require eventual 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
    APF-authorized data set, the USER column contains "
```

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 183 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 184 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 input data set, 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 184 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 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 106 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
*    5694-A01 (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 input data set:

- “Message input tags” on page 172
- “Special formatting tags for the message input data set” on page 180
- “How messages are formatted in the message buffer” on page 182
- “Using symbols in the message input data set” on page 184

Restrictions:

The following characters are tag characters, and you can only use them in tags in your message input data set. Note, however, that IBM Health Checker for z/OS does support symbols in the message input data set.

- Do not use < (less-than) and > (greater-than) - You can use &lt; and &gt; as substitutes.
- Do not use & (ampersand) - You can use &amp; as a substitute.

Message input tags

Copyright information

<lines id="*checkownername*" **props="copyright" > *** *copyright information* *****
</lines>

The message input data set 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
*    5694-A01 (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 input data set**. 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 input data set” on page 187.

xreftext="*csectname*"
Specify the name of the generated message table CSECT. The default message table module name is HZSHMTBL. *csectname* must be 1-8 alphanumeric characters. xreftext="*csectname*" is required.

rules="*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 input data set does not conform to the rules level selected, the HZSMSGEN exec will not be able to generate the message input data set.

rules="1"
Indicates that the check can run on systems at the z/OS V1R7 level or below, depending on what releases the individual check supports - see Chapter 13, “IBM Health Checker for z/OS checks,” on page 301

301. At rules level 1, a check message input data set 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>.

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” on page 175.
- 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.

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 may describe either z/OS Health Checker message behavior or a Common based event. 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 17. 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
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 "<msgtext>" on page 175.
- <msgitem> - See "message explanation" on page 178.

Restriction: You must code the following tags in consecutive lines without comments or blank lines between them:

<msg class="*msgtype*"></msg>
<msgnum xreftext="*nnnn*">*cccc*H*mmmms*</msgnum>
<msgtext>....

Message number tag - <msgnum>

<msgnum xreftext="*msgnumber*"**>***cccc***H***mmmms***</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 191 for guidelines on how to code the message text for translating messages.

You can have message variables in a message text - see "Variables for message text."

You **cannot** use paragraph (<p></p>) 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 182.

<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="*varable _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 156.

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:

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 the eighth character.

<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 18. 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 191.

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 19. 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
- 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

- <p> (paragraph) - text in a message item must be within paragraph tags (<p></p>).
- <lines> (lines) - allows you to control lines of text. Use <lines></lines> to generate a blank line.

See “Special formatting tags for the message input data set” on page 180.

You will have multiple <msgitem class=″*itemclass*″> tags for a message (see “Examples of message input” on page 163 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’.

Table 20. 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 input data set” on page 180. 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.
″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’.

Table 20. Description of <msgitem> classes required for all message explanations (continued)

″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. If your component is OCO (Object Code Only), use the component name instead of a module name. Consider using an entity declaration in HZSSDSN to create a symbol for the module. 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 "Great symbols for multiple users" on page 186.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 input data set

<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 "Variables for message text" on page 175 for complete information about variables.

<!-- comment --> tags and blank lines
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 input data set (<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 185 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 <p> 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.
This one is too long and is going to wrap around in a way I dislike, so I shouldn't really
</lines>
```

<lines> example 2: You cannot use <p></p> 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
```

<p></p>

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 <p> 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 21. 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. For example, for check GRS_MODE, the reason defined in the HZSADDCHECK exit routine is as follows:

  ```
  GRS should run in STAR mode to improve performance
  ```

  See Chapter 9, "Writing an HZSADDCHECK exit routine," on page 147.
- **Installation reason**: This field is displayed if the installation overwrites the HZSADDCHECK exit routine reason with a new reason value.
- **Check Parameters**: This field displays the parameters (defined in the HZSADDCHECK exit routine) that are passed to the check routine when it runs.

Using symbols in the message input data set

You can specify symbols in your message input data set 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 input data set CSECT.) See "Using pre-defined system symbols."
- Symbols defined in the source message input data set or setup file that resolve when you generate the CSECT for the message input data set. See "Defining your own symbols for check messages" on page 185.

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 input data set in order to use them. The system resolves the following pre-defined symbols that resolve when the check runs:

Table 22. 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.
&hzsexitrtn;	8	The name of the HZSADDCHECK exit routine.
&hzsparmsource ;	16	Resolves to 'Installation' or 'Owner' to indicate whether the default PARMS from the HZSADDCHECK exit routine are in effect, or user overrides are in effect.
&hzssev;	6	Resolves to the severity set defined at runtime from either the HZSPRMxx parmlib member or the HZSADDCHECK exit routine (HIGH | MEDIUM | LOW)
&hzsparms;	126	Active check parameters
&hzsckname;	32	The check name, as defined in the HZSADDCHECK exit routine
&hzsowner;	16	The check owner, which is the component or subsystem as defined in the HZSADDCHECK exit routine. 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`

Table 22. A summary of pre-defined symbols that resolve when the check runs (continued)

Predefined symbol	Maximum number of characters	Symbol resolves to
&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 input data set:

Table 23. A summary of pre-defined symbols that resolve when you generate the CSECT for the message input data set

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.
&gt;		Greater than symbol, >
&lt;		Less than symbol, <
&amp;		Ampersand symbol, &

If you want to use other symbols besides the predefined system symbols in your message input, you must define them in the source message input data set (before the <msglist> tag) or in the message setup data set, see “Defining your own symbols for check messages.”

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 input data set 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 input data set.
- 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 input data set. When you generate the message input data set, 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 input data set**, before the <msglist> tag. Here's an example of defining symbols in the message input data set 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 14. Example of a setup data set that defines symbols used in the message input data set

Using either of these methods, the symbols are resolved in the CSECT when you generate the message input data set.

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 input data set 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 input data set, 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 input data set, 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 input data set

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 input data set into a compilable assembler CSECT using the message generation exec HZSMSGEN. 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 191.

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 input data set.
   - Member HZSSSYMD containing a sample local symbol.

```
//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;

//* ***************************************************************
//* *                                                             *
//* * $MAC(HCHECK) COMP(SCHZS) PROD(HBB7730):                    *
//* *     HCHECKER SAMPLE MESSGAGE GENERATION JOB                 *
//* *                                                             *
//* * PROPRIETARY STATEMENT:                                      *
//* *                                                             *
//* *     LICENSED MATERIALS - PROPERTY OF IBM                    *
//* *     5694-A01                                                *
//* *     (C) COPYRIGHT IBM CORP. 2006                            *
//* *     STATUS = HBB7730                                        *
//* *                                                             *
//* *  HZSMSGEN - Z/OS HEALTH CHECKER MESSAGE GENERATION          *
//* *  -------------                                              *
//* *                                                             *
//* *  FUNCTION - Generate an assembler csect using a             *
//* *             structured message script                       *
//* *                                                             *
//* *                                                             *
//* *  RC  8    Some messages contain errors                      *
//* *      16   No messages could be processed                    *
//* *      20   The assembler csect could not be allocated        *
//* *      24   The message source could not be allocated         *
//* *      28   The entity symbol source dataset is unusable      *
//* *      32   The message report or SYSTSPRT was unusable.      *
//* *      36   The NLS skeleton ouput data set could not be      *
//* *           allocated when NLSCHECK(Y) was specified.         *
//* *      40   The NLS skeleton prologue data set could not be *
//* *           allocated when NLSCHECK(Y) was specified.         *
//* *      44   A TSO/E environment does not exist.               *
//* *                                                             *
//* *      SYSTSPRT is required to support errors                 *
//* *      an abendu0102 will occur if SYSTSPRT is                *
//* *      not provided                                           *
//* *                                                             *
//* *                                                             *
//* *                                                             *
//* *  INSTRUCTIONS                                               *
//* *                                                             *
//* *  1. Customize the job card                                  *
//* *                                                             *
//* *  2. Set variable SYSPROC to the name of the library         *
//* *     containing the message generation exec HZSMSGEN.        *
//* *                                                             *
//* *  3. Set variable HZSMDSN to the name of the library         *
//* *     containing the input message script. A fixed record     *
//* *     length of 80 is recommended for this data set.          *
//* *     The message source input cannot have sequence           *
//* *     numbers in it.                                          *
//* *                                                             *
//* *     If you are using the ISPF editor to create this         *
//* *     file, set the sequence numbers off by typing            *
//* *     NUMBER OFF on the command line before you begin         *
//* *     entering data.  If sequence numbers already exist       *
//* *     type UNNUM to remove them.                              *
//* *                                                             *
//* *                                                             *
//* *  4. Set variable HZSADSN to the name of the library         *
//* *     to contain the output assembler CSECT. A fixed record *
//* *     length of 80 is recommended for this data set.          *
//* *                                                             *
//* *     This job will fail with a RC = 20 when a DUMMY          *
//* *     data set is used.                                       *
//* *                                                             *
//* *  5. Set variable HZSSDSN to the name of a sequential        *
//* *     data set or a member of a PDS.  This data set           *
//* *     contains symbol  statements.                  *
//* *                                                             *
//* *  6. Change the parameter NLSCHECK(N) to NLSCHECK(Y) to      *
//* *     verify messages conform to guidelines required for      *
//* *     HZSMSGEN NLS skeleton generation. If you want to        *
//* *     generate  a skeleton follow optional steps below.       *
//* *                                                             *
//* * Optional data sets for NLS message skeletons.             *
//* *                                                           *
//* *  Uncomment the DD statements HZSNPSKE and HZSNLSKE.       *
//* *                                                           *
//* *  See z/OS MVS Assembler Services Guide for addition       *
//* *  detail on how the system uses messages skeleton to       *
//* *  translate.                                               *
//* *                                                           *
//* *                                                           *
//* *  7. 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.                    *
//* *                                                           *
//* *     See MVS Assembler Services Guide for additional       *
//* *     information on translating messages.                  *
//* *                                                           *
//* *                                                           *
//* *  8. 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.            *
//* *                                                           *
//* *                                                           *
//* * SYNTAX of PARMS provided to HZSMSGEN:                     *
//* *                                                           *
//* *  NLSCHECK  - (Y|YES)  IF HZSNLSKE and HZSNPSKE exists     *
//* *                       NLS skeletons will be generated     *
//* *                       for each WTO issued by an           *
//* *                       exception message. HZSM0017 is      *
//* *                       issued when a the WTO is not        *
//* *                       clearly defined for NLS.            *
//* *                                                           *
//* *            (N|NO)    The WTO text is not checked. NLS     *
//* *                       skeletons are not created.          *
//* *                                                           *
//* *            Default: NLSCHECK(Y)                           *
//* *                                                           *
//* *                                                           *
//* *  SOURCE    - (ERROR|ALL)                                  *
//* *                       ERROR indications message           *
//* *                       source only appears for messages    *
//* *                       that contain an error.              *
//* *                                                           *
//* *                       ALL includes the entire message     *
//* *                       source in the output                *
//* *                                                           *
//* *            Default: SOURCE(ERROR)                         *
//* *                                                           *
//* *                                                           *
//* *  NOTE: Errors are reported by the SYSTSPRT DD             *
//* *                                                           *
//* *                                                           *
//* * CHANGE-ACTIVITY:                                          *
//* *  $L0=HCHECK  HZS7720,040821, PDZJ: HEALTH CHECKER         *
//* *  $L1=ME01048 HZS7720,040901, PDZJ: PROLOGUE               *
//* *  $L2=MExxxxx HBB7730,050912, PDZJ: skeleton support       *
//* *                                                           *
//* *                                                           *
//* *************************************************************
//*
//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)
/*
```

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 191). If you want to generate message skeletons for message translation, you must uncomment the HZSNPSKE and HZSLNSKE DD statements:
   - Set HZSNPSKE 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 HZSNLSKE and the completed message skeletons for the check messages.
   - Set HZSNLSKE 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 HZSNLSKE data set:

     ```
     .VENUNHBB7730 5694-A010601
     .*
     .*******************************************************************
     .*
     .* COPYRIGHT  -
     .*       5637-A01
     .*       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 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 input data set. For example, if you want to generate NLS skeletons for your messages, you must break up message text in the message input data set 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> tags″ " on page 180 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 18 on page 177 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 22 on page 184.

- 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 can vary if you specify field size, <mv class="gmltime" xreftext=" fieldsize(11)" for example, the format used in the skeletons will also vary by length as follows:

```
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.
20   mm/dd/yyyy.hh.mm.ss.         &DATE=DATEMDY4. &TIME=TIMEHMSC.
19   mm/dd/yyyy.hh.mm.ss          &DATE=DATEMDY4. &TIME=TIMEHMSC.
18   mm/dd/yyyy.hh.mm.s           &DATE=DATEMDY4. &hh..&mm..&s.
17   mm/dd/yyyy.hh.mm.            &DATE=DATEMDY4. &hh..&mm..
16   mm/dd/yyyy.hh.mm             &DATE=DATEMDY4. &TIME=TIMEHMC.
15   mm/dd/yyyy.hh.m              &DATE=DATEMDY4. &hh..&m.
14   mm/dd/yyyy.hh.               &DATE=DATEMDY4. &hh..
13   mm/dd/yyyy.hh                &DATE=DATEMDY4. &hh.
12   mm/dd/yyyy.h                 &DATE=DATEMDY4. &h.
11   mm/dd/yyyy                   &DATE=DATEMDY4.
10   mm/dd/yyyy                   &DATE=DATEMDY4.
1-9  mm/dd/yyyy                   &username.
```

Part 3. Reference

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 200 - 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 203 - The REXX check invokes HZSLFMSG to issue messages.
- “HZSLSTOP function” on page 213 - 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 24. 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.

Output variables

The following REXX variable are returned by HZSLSTRT:

Table 25. HZSLSTRT output variables

Variable name	Description
RC	The return code for the HZSLSTRT function. The possible return codes are as follows: **0** **Meaning:** Indicates that the HZSLSTRT function completed successfully. **Action:** None required. **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** **Meaning:** HZSLSTRT did not complete because of an environment error. **Action:** Refer to action under the individual reason code returned in HZSLSTRT_RSN **16** **Meaning:** HZSLSTRT did not completed because of an component error. **Action:** Refer to action under the individual reason code returned in HZSLSTRT_RSN

Table 25. HZSLSTRT output variables (continued)

Variable name	Description
HZSLSTRT_RSN	The reason code explaining a RESULTvalue 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. **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: **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. **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_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_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,

Table 25. HZSLSTRT output variables (continued)

Variable name	Description
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

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.

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.

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 236.

Invocation: CALL HZSLFMST

Input variables

The following REXX variables are input to HZSLFMSG:

Table 26. 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.
HZSLFMSG_REQUEST= { 'CHECKMSG' | 'HZSMSG' | 'STOP' }	Identifies the source of the message text. • 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. • 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.

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 27. 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.

Table 27. HZSLFMSG_REQUEST='CHECKMSG' input variables (continued)

Variable name	Description

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 input data set. A class attribute of hex, decimal or timestamp in the message input data set 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 input data set 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 input data set 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.

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 28. 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 28. 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_INSERT1 = *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_INSERT1 = *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 28. 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_INSERT1 = *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_INSERT1 = *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='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_INSERT1 = *parm* - 1 to 16 character name of the parameter in error. • HZSLFMSG_INSERT.2 = *value* - 1 to 17 character parameter value in error.

Table 28. HZSLFMSG_REQUEST='HZSMSG' input variables (continued)

Variable name	Description
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_INSERT1 = *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_INSERT1 = *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_INSERT1 = *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='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_INSERT1 = *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.

Table 28. HZSLFMSG_REQUEST='HZSMSG' input variables (continued)

Variable name	Description
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.2 = *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.

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 29. 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 30. HZSLFMSG output varaibles

Variable name	Description
RESULT	The return code for the HZSLFMSG function. The possible return codes are as follows: **0** **Meaning:** Indicates that the HZSLFMSG function completed successfully. **Action:** None required. **8** **Meaning:** The HZSLFMSG function did not complete because of an error. **Action:** Refer to action under the individual reason code returned in HZSLFMSG_RSN **12** **Meaning:** The HZSLFMSG function did not complete because of an environment error. **Action:** Refer to action under the individual reason code returned in HZSLFMSG_RSN **16** **Meaning:** The HZSLFMSG function did not complete because of a component error. **Action:** Refer to action under the individual reason code returned in HZSLFMSG_RSN

Table 30. HZSLFMSG output varaibles (continued)

Variable name	Description
HZSLFMSG_RSN	The reason code explaining a RESULTvalue 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.
	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 30. 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 “HZSFMSG ABEND Codes” on page 252). **HZSLFMSG_ABENDRESULT** ABEND result string returned by HZSFMSG service. **xxxx08xx** **Meaning:** HzsfmsgRc_EnvParm was returned by the HZSFMSG macro. **Action:** Refer to the action under “HZSFMSG Return and Reason Codes” on page 254 the HZSFMSG macro. **xxxx0Cxx** **Meaning:** HzsfmsgRc_EnvError was returned by the HZSFMSG macro. **Action:** Refer to the action under “HZSFMSG Return and Reason Codes” on page 254 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

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.

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.

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 31. 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.
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 32. HZSLSTOP output variables

Variable name	Description
RC	The return code for the HZSLSTOP function. The possible return codes are as follows: **0** **Meaning:** Indicates that the HZSLSTOP function completed successfully. **Action:** None required. **4** **Meaning:** HZSLSTOP completed with a warning. **Action:** Refer to 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** **Meaning:** HZSLSTRT did not complete because of an environment error. **Action:** Refer to action under the individual reason code returned in HZSLSTRT_RSN **16** **Meaning:** HZSLSTRT did not completed because of an component error. **Action:** Refer to action under the individual reason code returned in HZSLSTRT_RSN

Table 32. HZSLSTOP output variables (continued)

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. **xxxx04xx** **Meaning**: HZSCHECK REQUEST=OPSTART HzscheckRC_Warning was returned from the HZSCHECK macro. **Action:** Refer to the action under the individual Reason code as documented by the HZSCHECK macro. **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. **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: **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 HZSLTOP 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. **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 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

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

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.

Chapter 12. IBM Health Checker for z/OS HZS macros

IBM Health Checker for z/OS includes the following macros.

- Use “HZSADDCK macro — HZS add a check” on page 218 to define a check in a HZSADDCHECK exit routine
- Use “HZSFMSG macro — Issue a formatted check message” on page 236 to issue messages in check routines
- Use “HZSQUERY macro — HZS Query” on page 259 to obtain information about checks that are currently registered with IBM Health Checker For z/OS.
- Use “HZSCHECK macro — HZS Check command request” on page 275 to manage a check and in registration routines, to refresh a check
- Use “HZSCPARS macro — HZS Check Parameter Parsing” on page 289 to parse check parameters.

HZSADDCK macro — HZS add a check

Description

The HZSADDCK macro is used by the HZSADDCHECK dynamic exit routine 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 and the check routines run in IBM Health Checker for z/OS address space.

Environment

The requirements for the caller are:

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-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

- This macro must be invoked from an exit routine associated with the HZSADDCHECK dynamic exit.
- 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

The HZSADDCK macro is written as follows:

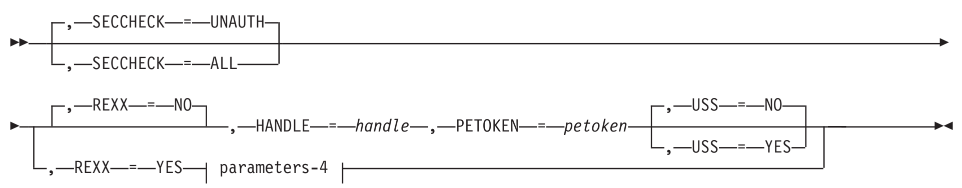
main diagram

```
►►──┬──────┬──HZSADDCK──b──CHECKOWNER──=──checkowner──,──CHECKNAME──=──checkname──────────────────►
    └─name─┘

   ┌─,──REMOTE──=──NO──┐
►──┴─────────────────────┤ parameters-1 ├──┬──,──DATE──=──date──────────────────────────────────►
   └─,──REMOTE──=──YES──┤ parameters-2 ├──┘

►─,──REASON──=──reason──,──REASONLEN──=──reasonlen──────────────────────────────────────────────►

►──┬────────────────────────────────────────────────────────┬──┬───────────┬──┬──────────────┬──►
   ├─,──PARMS──=──NO_PARMS──┬──┬─,──PARMSLEN──=──────────────┤  ├─,──LOCAL──┤  ├─,──ACTIVE────┤
   └─,──PARMS──=──parms─────┘  └─,──PARMSLEN──=──parmslen──┘  └─,──GLOBAL─┘  └─,──INACTIVE──┘

►──┬─,──SEVERITY──=──LOW──┬─────────────────────────────────────────────────────────────────────►
   ├─,──SEVERITY──=──MED──┤
   └─,──SEVERITY──=──HI───┘

                                   ┌─,──EIHOURS──=────────────┐  ┌─,──EIMINUTES──=──────────────┐
►──┬─,──INTERVAL──=──ONETIME──┴──────────────────────────┴──┴──────────────────────────────┴──┬──►
   │                               └─,──EIHOURS──=──eihours──┘  └─,──EIMINUTES──=──eiminutes──┘ │
   └─,──INTERVAL──=──TIMER──┤ parameters-3 ├────────────────────────────────────────────────────┘

   ┌─,──VERBOSE──=──NO───┐
►──┴─────────────────────┴──┬─────────────────────────┬──┬─────────────────────────┬───────────►
   └─,──VERBOSE──=──YES──┘  └─,──RETCODE──=──retcode──┘  └─,──RSNCODE──=──rsncode──┘

   ┌─,──PLISTVER──=──IMPLIED_VERSION──┐  ┌─,──MF──=──S──────────────────────────────────────────┐
►──┴──────────────────────────────────┴──┼─────────────────────────────────────────────────────┼──►◄
   ├─,──PLISTVER──=──MAX──────────────┤  │                                     ┌─,──D────┐     │
   ├─,──PLISTVER──=───────────────────┤  ├─,──MF──=──(──L──,──list addr──────┴─────────┴──)──┤
   └─,──PLISTVER──=───────────────────┘  │                                     └─,──attr─┘     │
                                          │                                     ┌─,──COMPLETE─┐ │
                                          └─,──MF──=──(──E──,──list addr──────┴─────────────┴──)┘
```

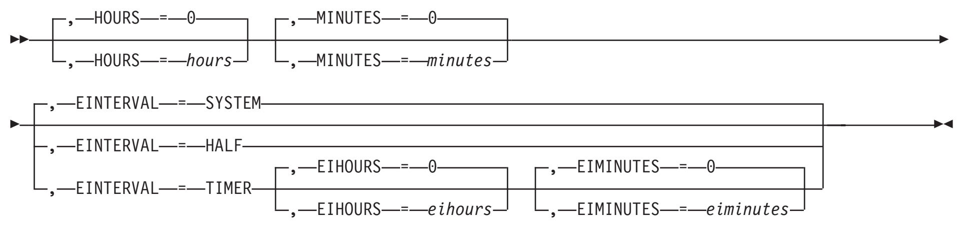
parameters-1

```
►►──,──CHECKROUTINE──=──checkroutine──┬────────────────────────────────┬──────►
                                       └─,──ENTRYCODE──=──entrycode──┘

                                                ┌─,──USS──=──NO───┐
►─,──EXITRTN──=──exitrtn──,──MSGTBL──=──msgtbl──┴─────────────────┴──────────►◄
                                                └─,──USS──=──YES──┘
```

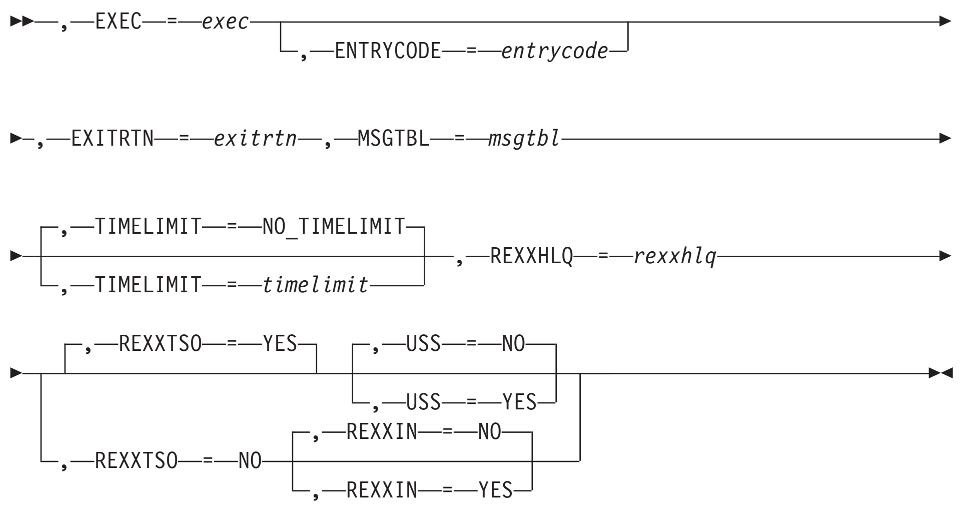
parameters-2



parameters-3



parameters-4


,—EXEC—=—*exec*
,—ENTRYCODE—=—*entrycode*
,—EXITRTN—=—*exitrtn*—,—MSGTBL—=—*msgtbl*
,—TIMELIMIT—=—NO_TIMELIMIT
,—TIMELIMIT—=—*timelimit*
,—REXXHLQ—=—*rexxhlq*
,—REXXTSO—=—YES
,—USS—=—NO
,—USS—=—YES
,—REXXTSO—=—NO
,—REXXIN—=—NO
,—REXXIN—=—YES

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.

,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.

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.

,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. *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. *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. *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. *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_EntryCode.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,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.

,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. *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. *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.

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.

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. *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	PETOKEN	TIMELIMIT
HANDLE	REXXHLQ	

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.

,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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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″.
| 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 Unix System Services. This information is used when Unix System Services 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 Unix System Services before allowing the Unix System Services shutdown to complete. Also, when Unix System Services are not

available, checks that have indicated they use those services are not run. Thus, indicating ″YES″ if the check actually does not use Unix System Services 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 Unix System Services.

,USS=YES
indicates the check does use Unix System Services.

,USS=NO
,USS=YES
When REXX=NO and REMOTE=YES are specified, an optional parameter that indicates whether the check uses Unix System Services. When Unix System Services are not available, checks that have indicated they use those services are not run. Thus, indicating ″YES″ if the check actually does not use Unix System Services 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 Unix System Services.

,USS=YES
indicates the check does use Unix System Services.

,USS=NO
,USS=YES
When REXXTSO=YES, REXX=YES and REMOTE=YES are specified, an optional parameter that indicates whether the check uses Unix System Services. When Unix System Services are not available, checks that have indicated they use those services are not run. Thus, indicating ″YES″ if the check actually does not use Unix System Services 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 Unix System Services.

,USS=YES
indicates the check does use Unix System Services.

,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.

HZSADDCK ABEND Codes

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.

HZSADDCK 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 33. 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.

Table 33. Return and Reason Codes for the HZSADDCK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
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.
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.

Table 33. Return and Reason Codes for the HZSADDCK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
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.
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.
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
10	—	**Equate Symbol**: HzsaddckRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.

Table 33. Return and Reason Codes for the HZSADDCK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
10	xxxx1001	**Equate Symbol**: HzsaddckRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer

Examples

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
```

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=LCHECKRTN,EXITRTN=LEXITRTN,        *
               MSGTBL=LMSGTBL,DATE=LDATE,                      *
               REASON=LREASON,REASONLEN=LREASONLEN,            *
               ENTRYCODE=LENTRYCODE,                           *
               SEVERITY=HI,INTERVAL=TIMER,                     *
               HOURS=LHOURS,MINUTES=LMINUTES,                  *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,              *
               MF=(E,ADDCKL)
*
* Place code to check return/reason codes here
*
LOWNER     DC   CL16'MYCOMPANY'
LNAME      DC   CL32'MYCOMPONENT_CHECK_OTHER_WIDGETS'
```

```
LCHECKRTN  DC   CL8'MYMODULE'
LEXITRTN   DC   CL8'MYEXITRT'
LMSGTBL    DC   CL8'MYMSGTBL'
LDATE      DC   CL8'20050601'
LREASON    DC   CL32'Verify other widgets are present'
LREASONLEN DC   A(L'LREASON)
LENTRYCODE DC   F'2'
LHOURS     DC   H'1'
LMINUTES   DC   H'15'
         HZSZCONS              Return code information
DYNAREA  DSECT
LRETCODE   DS   F
LRSNCODE   DS   F
```

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:

Minimum authorization:	Problem state. 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

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

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

```
►►─┬──────┬─HZSFMSG─ƀ─┬─REQUEST─=─CHECKMSG─┤ parameters-1 ├─┬──────────►
   └─name─┘           ├─REQUEST─=─HZSMSG─┤ parameters-2 ├───┤
                      └─REQUEST─=─STOP─┤ parameters-3 ├─────┘

                                         ┌─,─MGBFORMAT─=─0─┐
►─┬──────────────────────────────────┬───┼─────────────────┼──────────►
  └─,─ABENDRESULT─=─abendresult──────┘   └─,─MGBFORMAT─=─1─┘

►─┬─────────────────────────┬─┬────────────────────────┬──────────────►
  └─,─RETCODE─=─retcode─────┘ └─,─RSNCODE─=─rsncode────┘

  ┌─,─PLISTVER─=─IMPLIED_VERSION─┐
►─┼──────────────────────────────┼────────────────────────────────────►
  ├─,─PLISTVER─=─MAX─────────────┤
  └─,─PLISTVER─=─1───────────────┘

  ┌─,─MF─=─S────────────────────────────────────────┐
►─┼─────────────────────────────────────────────────┼─────────────────►◄
  │                                  ┌─,─0D───┐     │
  ├─,─MF─=─(─L─,─list addr───────────┼────────┼─)───┤
  │                                  └─,─attr─┘     │
  │                                  ┌─,─COMPLETE─┐ │
  └─,─MF─=─(─E─,─list addr───────────┴────────────┴─)┘
```

parameters-1

```
►►──┬─,──MGBADDR──=──mgbaddr─┬──────────────────────────────────────────────────────────────►
    └─,──MGB──=──mgb─────────┘

   ┌─,──REMOTE──=──NO────────────────────────────────────────────────────────────────────┐
►──┼─────────────────────────────────────────────────────────────────────────────────────┼──►◄
   │                                                ┌─,──REXX──=──NO─┐                    │
   └─,──REMOTE──=──YES──,──HANDLE──=──handle─────────┴────────────────┴──,──MSGTABLE──=──msgtable─┘
```

parameters-2

```
►►──┬─,──REASON──=──ERROR──,──DIAG──=──diag──────────────────┬──────────────────►
    ├─,──REASON──=──PARS1201──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1202──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1203──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1204─────────────────────────────────┤
    ├─,──REASON──=──PARS1205──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1206──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1207──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1208──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1209──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1210──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1211──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1212──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1213──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1214──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1215──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1216──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    ├─,──REASON──=──PARS1217─────────────────────────────────┤
    ├─,──REASON──=──PARS1218──┬─,──MGBADDR──=──mgbaddr─┬─────┤
    │                         └─,──MGB──=──mgb─────────┘     │
    └─,──REASON──=──PARS1219─────────────────────────────────┘

  ┌─,──REMOTE──=──NO───────────────────────────────┐
►─┼────────────────────────────────────────────────┼──────────────────────────►◄
  └─,──REMOTE──=──YES──,──HANDLE──=──handle────────┘
```

parameters-3

```
►►──┬─,──REASON──=──BADPARM──────────────────┬──────────────────────────►
    ├─,──REASON──=──ERROR──,──DIAG──=──diag──┤
    └─,──REASON──=──ENVNA────────────────────┘

   ┌─,──REMOTE──=──NO─────────────────────────────────────────────┐
►──┼──────────────────────────────────────────────────────────────┼─────►◄
   │                                               ┌─,──REXX──=──NO─┐ │
   └─,──REMOTE──=──YES──,──HANDLE──=──handle───────┴────────────────┴─┘
```

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.

,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.

,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=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.

,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 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 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=PARS1202 and REQUEST=HZSMSG are 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=PARS1203 and REQUEST=HZSMSG are 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=PARS1205 and REQUEST=HZSMSG are 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=PARS1206 and REQUEST=HZSMSG are 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=PARS1207 and REQUEST=HZSMSG are 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=PARS1208 and REQUEST=HZSMSG are 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=PARS1209 and REQUEST=HZSMSG are 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=PARS1210 and REQUEST=HZSMSG are 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=PARS1211 and REQUEST=HZSMSG are 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=PARS1212 and REQUEST=HZSMSG are 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=PARS1213 and REQUEST=HZSMSG are 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=PARS1214 and REQUEST=HZSMSG are 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=PARS1215 and REQUEST=HZSMSG are 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=PARS1216 and REQUEST=HZSMSG are 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=PARS1218 and REQUEST=HZSMSG are 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.

,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*

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.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX

,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.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

,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

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=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 HZS1002I.

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).

,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=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.

REQUEST=CHECKMSG
REQUEST=HZSMSG
REQUEST=STOP
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=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.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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=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.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

HZSFMSG ABEND Codes

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

This convention is used for the following abend reasons: xxxx4108 xxxx4109 xxxx410A xxxx410B xxxx410C xxxx410D xxxx410E xxxx4110 xxxx1013 xxxx1015 xxxx4116 xxxx1017 xxxx1018 xxxx4016

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.

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

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.

HZSFMSG 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 34. 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. :dt.xxxx0858 :dd.Name: HzsfmsgRsn_BadHandle :dd.Meaning: The handle provided with the HANDLE parameter is not valid. :dd.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. ¦¦>
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 34. 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

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, set usings
         LA    2,TheMGBArea
         ST    2,TheMGBAddr
         USING HZSMGB,2
         MVC   MGB_MessageNumber,=F'1' Message 1
         MVC   MGB_insert_cnt,=F'2'  Two inserts
         LA    3,Insert1Area         Address of first insert
         ST    3,MGB_Inserts         Save insert address
         LA    3,Insert2Area         Address of second insert
         USING MGB_MsgInsertD,3
         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
         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
```

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
```

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:

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 may be authorized for read access to any of the following:
  - when the check owner has wildcard character(s), or when the check owner has no wildcard characters and the check name has no wildcard characters, 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
- 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-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 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).

The user-provided answer area via the ANSAREA parameter has the same requirements and restrictions as the control parameters.

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.

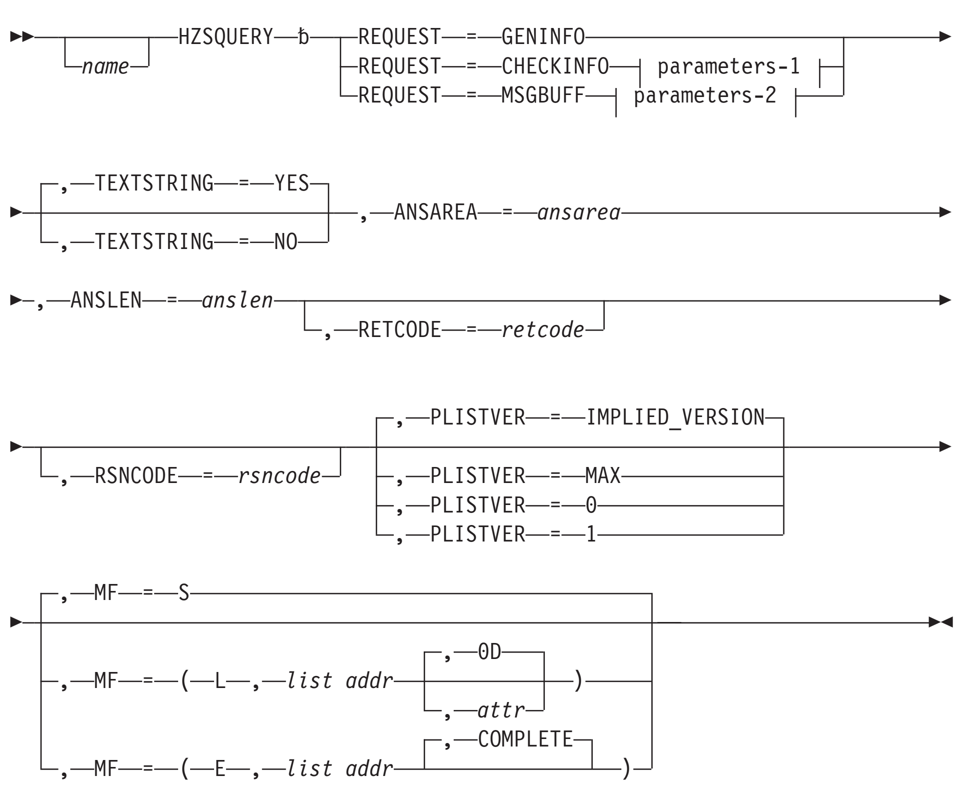
Performance Implications

None.

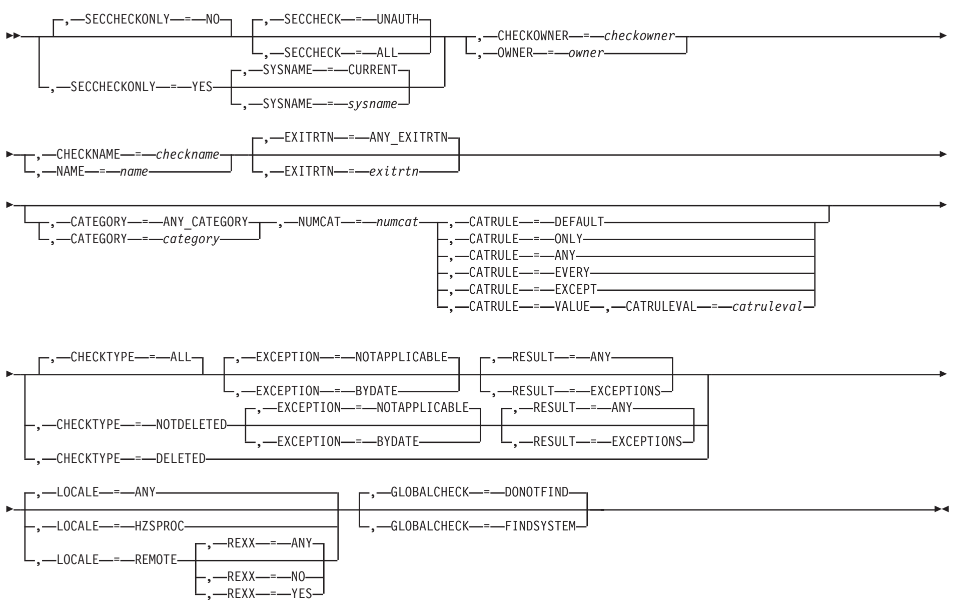
Syntax

The HZSQUERY macro is written as follows:

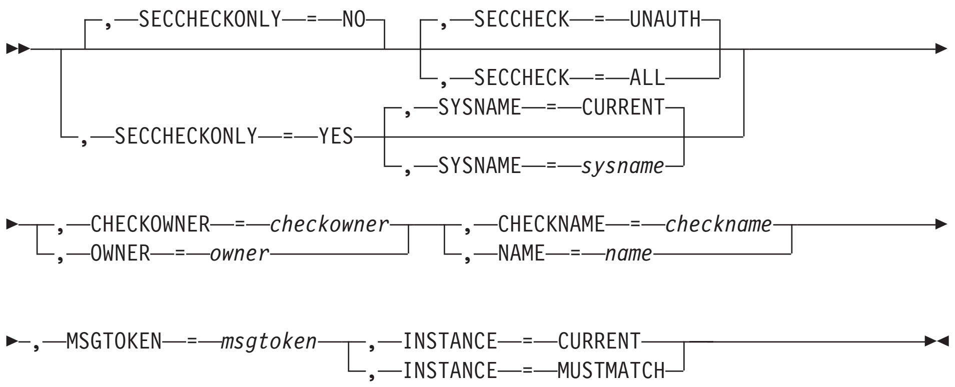
main diagram



parameters-1



parameters-2



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.

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.

,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_CATRULES_yyy equates generated by HZSQUERY MF=(L,xxx).

To code: Specify the RS-type address, or address in register (2)-(12), of an 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 REQUEST=MSGBUFF is 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 REQUEST=MSGBUFF is 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 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
When CHECKTYPE=ALL and REQUEST=CHECKINFO are specified, an optional parameter that indicates what exception processing to do The default is EXCEPTION=NOTAPPLICABLE.

,EXCEPTION=NOTAPPLICABLE
that indicates that exception processing is not applicable to this request

,EXCEPTION=BYDATE
that indicates to find only checks that have 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=NOTAPPLICABLE
,EXCEPTION=BYDATE
When CHECKTYPE=NOTDELETED and REQUEST=CHECKINFO are specified, an optional parameter that indicates what exception processing to do The default is EXCEPTION=NOTAPPLICABLE.

,EXCEPTION=NOTAPPLICABLE
that indicates that exception processing is not applicable to this request

,EXCEPTION=BYDATE
that indicates to find only checks that have date exceptions (i.e., a policy statement that matches this check was not applied because its date was older than the check's date). Note that DELETED checks are not considered to have policy date exceptions.

,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 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
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 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.

,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).

,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 REQUEST=MSGBUFF is 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 REQUEST=MSGBUFF is 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 an 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 REQUEST=MSGBUFF is 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
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

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, or 1

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 in macro HZSQUAA) 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 indicating that there is one entry provided, with the address of that entry being in field HzsquaahQuaaGAddr.
- 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 in macro HZSQUAA) which contains the number of entries that follows (HzsquaahNumQuaaC) and the address of the first entry (HzsquaahQuaaCAddr).
- entries (mapped by DSECT HZSQUAAC in macro HZSQUAA) each of which has a field that indicates the length of that entry (HzsquaaCLen). 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 in macro HZSQUAA) which contains the number of entries that follows (HzsquaahNumHCKL) and the address of the first entry (HzsquaahHcklAddr).
- 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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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 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 REQUEST=MSGBUFF 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.

,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 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. 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

None.

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 35. 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 indicates how much space is currently required. **Action**: Allocate a larger area and request the function again.
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 **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.

Table 35. Return and Reason Codes for the HZSQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0847	**Equate Symbol**: HzsqueryRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. You might not have set up its access register properly.
8	xxxx0848	**Equate Symbol**: HzsqueryRsn_BadAnsAreaALET **Meaning**: Bad answer area ALET. **Action**: Make sure that the ALET associated with the answer area is valid. You might not have set up its access register properly.
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. You might not have set up its access register properly.
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.
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
10	—	**Equate Symbol**: HzsqueryRc_CompError **Meaning**: Component Error **Action**: Refer to action under the individual reason code.

Table 35. Return and Reason Codes for the HZSQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
10	xxxx1001	**Equate Symbol**: HzsqueryRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer

Examples

None.

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:

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 may be authorized for access to any of the following:
  - when the check owner has wildcard character(s), or when the check owner has no wildcard characters and the check name has no wildcard characters, 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
- 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-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 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

```
                              ┌─REMOTE──=──ANY─┐
►►─┬──────┬─HZSCHECK──b──────┴────────────────┴─┤ parameters-1 ├─┬──────────►
   └─name─┘                 └─REMOTE──=──YES─┤ parameters-2 ├───┘

   ┌─,─SECCHECKONLY──=──NO─┐     ┌─,─SECCHECK──=──UNAUTH─┐
►──┼───────────────────────┴─────┼───────────────────────┼─┬──────────────────►
   │                             └─,─SECCHECK──=──ALL────┘ │
   │                            ┌─,─SYSNAME──=──CURRENT─┐  │
   └─,─SECCHECKONLY──=──YES─────┼───────────────────────┼──┘
                                └─,─SYSNAME──=──sysname─┘

   ┌─,─CART──=──NO_CART─┐  ┌─,─CONSID──=──NO_CONSID─┐
►──┼────────────────────┼──┼────────────────────────┼─────────────────────────►
   └─,─CART──=──cart────┘  └─,─CONSID──=──consid────┘

►──┬─────────────────────────┬──┬─────────────────────────┬──────────────────►
   └─,─RETCODE──=──retcode───┘  └─,─RSNCODE──=──rsncode───┘

   ┌─,─PLISTVER──=──IMPLIED_VERSION─┐
►──┼────────────────────────────────┼─────────────────────────────────────────►
   ├─,─PLISTVER──=──MAX─────────────┤
   ├─,─PLISTVER──=──0───────────────┤
   └─,─PLISTVER──=──1───────────────┘

   ┌─,─MF──=──S──────────────────────────────────────────┐
►──┼─────────────────────────────────────────────────────┼────────────────────►◄
   │                                   ┌─,─0D───┐        │
   ├─,─MF──=──(──L──,──list addr───────┼────────┼──)─────┤
   │                                   └─,─attr─┘        │
   │                                   ┌─,─COMPLETE─┐    │
   └─,─MF──=──(──E──,──list addr───────┴────────────┴──)─┘
```

parameters-1

```
►►──┬─,─REQUEST──=──DELETE──┤ parameters-3 ├─┬────────────────────────────►◄
    ├─,─REQUEST──=──ADDNEW───────────────────┤
    └─,─REQUEST──=──RUN──┤ parameters-4 ├────┘
```

parameters-2

```
►►──,──HANDLE──=──handle──────────────────────────────────────────────────────────────────►

                                              ┌─,──REXX──=──NO─┐
►──┬─,──REQUEST──=──OPSTART──┬─────────────┬──┴────────────────┴─,──PETOKEN──=──petoken──┬─────►◄
   │                         └─,──PQE──=──pqe─┘                                          │
   │                                                            ┌─,──REXX──=──NO─┐       │
   └─,──REQUEST──=──OPCOMPLETE──┬────────────────────────────────┬┴────────────────┴───────┘
                                ├─,──PQECHKWORK──=──NO_PQECHKWORK─┤
                                └─,──PQECHKWORK──=──pqechkwork────┘
```

parameters-3

```
                                                              ┌─,──EXITRTN──=──ANY_EXITRTN─┐
►►──,──CHECKOWNER──=──checkowner──,──CHECKNAME──=──checkname──┼────────────────────────────┴──►
                                                              └─,──EXITRTN──=──exitrtn─────┘

►──┬──────────────────────────────────────────────────────────────────────────────────────┬──►◄
   └─┬─,──CATEGORY──=──ANY_CATEGORY─┬─,──NUMCAT──=──numcat─┬─,──CATRULE──=──DEFAULT─┬─────┘
     └─,──CATEGORY──=──category─────┘                      ├─,──CATRULE──=──ONLY────┤
                                                           ├─,──CATRULE──=──ANY─────┤
                                                           ├─,──CATRULE──=──EVERY───┤
                                                           └─,──CATRULE──=──EXCEPT──┘
```

parameters-4

```
                                                              ┌─,──EXITRTN──=──ANY_EXITRTN─┐
►►──,──CHECKOWNER──=──checkowner──,──CHECKNAME──=──checkname──┼────────────────────────────┴──►
                                                              └─,──EXITRTN──=──exitrtn─────┘

►──┬──────────────────────────────────────────────────────────────────────────────────────┬──►◄
   └─┬─,──CATEGORY──=──ANY_CATEGORY─┬─,──NUMCAT──=──numcat─┬─,──CATRULE──=──DEFAULT─┬─────┘
     └─,──CATEGORY──=──category─────┘                      ├─,──CATRULE──=──ONLY────┤
                                                           ├─,──CATRULE──=──ANY─────┤
                                                           ├─,──CATRULE──=──EVERY───┤
                                                           └─,──CATRULE──=──EXCEPT──┘
```

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, which 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, which 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 an 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 an 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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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

None.

HZSCHECK 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 36. 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.
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. You might not have set up its access register properly.
8	xxxx084B	**Equate Symbol**: HzscheckRsn_BadParmlistValue **Meaning**: A parameter list field contains an unsupported value. **Action**: Check for possible storage overlay

Table 36. Return and Reason Codes for the HZSCHECK Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx084C	**Equate Symbol**: HzscheckRsn_BadCategoryALET **Meaning**: Bad category ALET. **Action**: Make sure that the ALET associated with the category area is valid. You might not have set up its access register properly.
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. You might not have set up its access register properly.
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. You might not have set up its access register properly.
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	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. You might not have set up its access register properly.
8	xxxx0866	**Equate Symbol**: HzscheckRsn_BadPqeChkWorkArea **Meaning**: Error while reading from the PqeChkWork area **Action**: Make sure that the provided area is valid.

Table 36. 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. You might not have set up its access register properly.
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.
10	xxxx1001	**Equate Symbol**: HzscheckRsn_IntError **Meaning**: Unexpected internal error **Action**: Report the problem to the system programmer

Examples

None.

HZSCPARS macro — HZS Check Parameter Parsing

Description

The HZSCPARS macro provides functions dealing with parsing the check parameter.

Environment

The requirements for the caller are:

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

```
►►──┬──────┬──HZSCPARS──b──────────────────────────────────────────────────►
    └─name─┘

►──┬─REQUEST──=──PARSE──┤ parameters-1 ├──────────────────────┬────────────►
   ├─REQUEST──=──CHECKPARM──┤ parameters-2 ├──────────────────┤
   ├─REQUEST──=──CHECKNOTPROC──,──CPARAREA──=──cpararea───────┤
   ├─REQUEST──=──CHECKDEC──┤ parameters-3 ├───────────────────┤
   ├─REQUEST──=──CHECKHEX──┤ parameters-4 ├───────────────────┤
   ├─REQUEST──=──CHECKCHAR──┤ parameters-5 ├──────────────────┤
   └─REQUEST──=──FREE──,──CPARAREA──=──cpararea───────────────┘

   ┌─,──REMOTE──=──NO───────────────────────────┐
►──┼────────────────────────────────────────────┼──────────────────────────►
   └─,──REMOTE──=──YES──,──HANDLE──=──handle────┘

►──┬──────────────────────────────┬──┬──────────────────────────────┬──────►
   └─,──RETCODE──=──retcode───────┘  └─,──RSNCODE──=──rsncode───────┘

   ┌─,──PLISTVER──=──IMPLIED_VERSION─┐
►──┼─────────────────────────────────┼─────────────────────────────────────►
   ├─,──PLISTVER──=──MAX─────────────┤
   └─,──PLISTVER──=──0───────────────┘

   ┌─,──MF──=──S─────────────────────────────────────────────┐
►──┼─────────────────────────────────────────────────────────┼────────────►◄
   │                                    ┌─,──0D───┐          │
   ├─,──MF──=──(──L──,──list addr───────┼─────────┼──)───────┤
   │                                    └─,──attr─┘          │
   │                                    ┌─,──COMPLETE─┐      │
   └─,──MF──=──(──E──,──list addr───────┴─────────────┴──)───┘
```

parameters-1

```
►►—,—PARM—=—parm—,—PARMLEN—=—parmlen—┬—,—TOUPPER—=—YES—┬——————————►
                                     └—,—TOUPPER—=—NO——┘

                                   ┌—,—PARMFORMAT—=—EITHER——┐
►—,—CPARAREAADDR—=—cparareaaddr————┤                        ├—————————►◄
                                   └—,—PARMFORMAT—=—KEYWORD—┘
```

parameters-2

```
►►—,—CPARAREA—=—cpararea———————————————————————————————————————————►

                           ┌—,—PARMPOS—=—NOT_POS—┐
►—,—PARMNAME—=—parmname————┤                     ├—————————————————►
                           └—,—PARMPOS—=—parmpos—┘

►——┬———————————————————————————————————┬———————————————————————————►
   └—,—KEYENTRYADDR—=—keyentryaddr—————┘

►——┬—————————————————————————————————————┬—,—MINVALUES—=—minvalues—►
   └—,—FIRSTVALUEADDR—=—firstvalueaddr———┘

►—,—MAXVALUES—=—maxvalues—————————————————————————————————————————►◄
```

parameters-3

```
►►—,—PARMNAME—=—parmname———————————————————————————————————————————►

►—┬—,—KEYENTRY—=—keyentry——————————————————————————————————————┬———►
  └—,—KEYVALUEENTRY—=—keyvalueentry—,—NEXTVALUEADDR—=—nextvalueaddr—┘

►—,—MINVALUEDEC—=—minvaluedec—,—MAXVALUEDEC—=—maxvaluedec——————————►

                           ┌—,—PERCENTVALUE—=—NOT_USED——————┐
►—┬—,—PERCENTOK—=—YES——————┤                                ├—┬————►
  │                        └—,—PERCENTVALUE—=—percentvalue——┘ │
  └—,—PERCENTOK—=—NO——————————————————————————————————————————┘

►—,—KEYINFOAREA—=—keyinfoarea—————————————————————————————————————►◄
```

parameters-4

```
►►──,──PARMNAME──=──parmname──────────────────────────────────────────────►

►──┬─,──KEYENTRY──=──keyentry──────────────────────────────────┬──────────►
   └─,──KEYVALUEENTRY──=──keyvalueentry──,──NEXTVALUEADDR──=──nextvalueaddr─┘

►─,──MINVALUEHEX──=──minvaluehex──,──MAXVALUEHEX──=──maxvaluehex──────────►

►─,──KEYINFOAREA──=──keyinfoarea─────────────────────────────────────────►◄
```

parameters-5

```
►►──,──PARMNAME──=──parmname──────────────────────────────────────────────►

►──┬─,──KEYENTRY──=──keyentry──────────────────────────────────┬──────────►
   └─,──KEYVALUEENTRY──=──keyvalueentry──,──NEXTVALUEADDR──=──nextvalueaddr─┘

►─,──MINLEN──=──minlen──,──MAXLEN──=──maxlen──,──KEYINFOAREA──=──keyinfoarea──►◄
```

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=*cpararea addr*
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 ParseKeywordValueEntry 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 ParseKeywordEntry 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 ParseKeywordEntry 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 ParseKeywordEntry 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 ParseKeywordEntry 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

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 ParseKeywordValueEntry 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 ParseKeywordValueEntry 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 ParseKeywordValueEntry 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.

,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.

,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.

,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.

,NEXTVALUEADDR=*nextvalueaddr*
When KEYVALUEENTRY=*keyvalueentry* and REQUEST=CHECKDEC are specified, a required output parameter that is to contain the address of the next ParseKeywordValueEntry 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 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 of the next ParseKeywordValueEntry 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 of the next ParseKeywordValueEntry 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.

,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.

,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.

,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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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 37. 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.
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.

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`

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.

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 33.

| All checks are **local** checks (run in the IBM Health Checker for z/OS address
| space) unless otherwise noted.

This document covers the following checks:
- "ASM checks (IBMASM)" on page 302
- | "Communications Server checks (IBMCS)" on page 306
- "Consoles checks (IBMCNZ)" on page 314
- "Contents supervision checks (IBMCSV)" on page 321
- "PDSE checks (IBMPDSE)" on page 331
- "Global Resource Serialization checks (IBMGRS)" on page 327
- "RACF checks (IBMRACF)" on page 331
- "RRS checks (IBMRRS)" on page 345
- "RSM checks (IBMRSM)" on page 349
- "SDUMP checks (IBMSDUMP)" on page 354
- "Supervisor (IBMSUP)" on page 355
- "System logger checks (IBMIXGLOGR)" on page 357
- | "TSO/E (IBMTSOE)" on page 359
- "z/OS UNIX System Services checks (IBMUSS)" on page 361
- "VSAM checks (IBMVSAM)" on page 366
- "VSM checks (IBMVSM)" on page 369
- "Cross system coupling facility (XCF) checks (IBMXCF)" on page 376

Several check names have changed since they were released in the prototype. The following table lists the old and new check names:

Table 38. Updated check names

Prototype check name	IBM Health Checker for z/OS check name
EMCS_hardcopy	CNZ_EMCS_Hardcopy_Mscope
SYSCONS_MSCOPE	CNZ_Syscons_Mscope
SYSCONS_ROUTCODES	CNZ_Syscons_Routcode
SYSCONS_PDMODE	CNZ_Syscons_PD_Mode

Table 38. Updated check names (continued)

Prototype check name	IBM Health Checker for z/OS check name
SYSCONS_MASTER	CNZ_Syscons_Master
Console_Master	CNZ_Console_MasterAuth_Cmdsys
Console_MSCOPE_and_Routcodes	CNZ_Console_Mscope_and_Routcode
AMRF_And_MPF_Consistent	CNZ_AMRF_Eventual_Action_Msgs
Console_routcode_11	CNZ_Console_Routcode_11
APF_LIBS	CSV_APF_EXISTS
LINKLIB_SPACE	CSV_LNKLST_SPACE
LINKLIB_EXTENTS	CSV_LNKLST_NEWEXTENTS
GRS_SyncRsv	GRS_SYNCHRES

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.

Best practice:
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 up.

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(IBMASM,ASM_NUMBER_LOCAL_DATASETS),
      INTERVAL(ONETIME),
      SEVERITY(LOW),
      PARM('MINLOCALS(3)'),
      DATE('20041006')
```

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:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMASM,ASM_PAGE_ADD),
      INTERVAL(ONETIME),
      SEVERITY(MED),
      PARM('MINADDS(2)'),
      DATE('20041006')
```

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:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)* .

SECLABEL recommended for multilevel security (MLS) 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 100% of the slots required for all CSA/ECSA and PLPA/EPLPA.

Best practice:
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 up.

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(IBMASM,ASM_PLPA_COMMON_SIZE),
      INTERVAL(ONETIME),
      PARM('THRESHOLD(100%)'),
      DATE('20041006')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, in keyword THRESHOLD, which is an integer of 0-100 indicating the warning threshold percent. The percent sign is optional. The default is THRESHOLD(100%).

Reference:
For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)* .

SECLABEL recommended for multilevel security (MLS) 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 80%.

Best practice:
You should prevent full conditions on the PLPA and Common page data sets.

z/OS releases the check applies to:
z/OS V1R8 and up.

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(IBMASM,ASM_PLPA_COMMON_USAGE),
      INTERVAL(00:30),
      PARM('THRESHOLD(80%)'),
      DATE('20041006')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, in keyword THRESHOLD, which is an integer of 0-100 indicating the warning threshold percent. The percent sign is optional. The default is THRESHOLD(80%).

Reference:
For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)* .

SECLABEL recommended for multilevel security (MLS) 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 30%.

Best practice:
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 up.

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(IBMASM,ASM_LOCAL_SLOT_USAGE),
      INTERVAL(00:30),
      SEVERITY(MED),
      PARM('THRESHOLD(30%)'),
      DATE('20041006')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, in keyword THRESHOLD, which is an integer of 0-100 indicating the warning threshold percent. The percent sign is optional. The default is THRESHOLD(30%).

Reference:
For information on auxiliary storage management initialization, see *z/OS MVS Initialization and Tuning Guide* .

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)* .

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Communications Server checks (IBMCS)

CSTCP_SYSTCPIP_CTRACE_TCPIP*stackname*

Description:
Checks if TCP/IP Event Trace (SYSTCPIP) is active with options other than the default options (MINIMUM, INIT, OPCMDS, or OPMSGS). The TCPIP*stackname* 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.

Best practice:
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 up.

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(IBMCS,CSTCP_SYSTCPIP_CTRACE_TCPIPstackname)
   DATE(20050214)
   REASON('CHECK FOR TCP/IP CTRACE WITH NONDEFAULT OPTIONS')
   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:
See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CSTCP_SYSPLEXMON_RECOV_TCPIP*stackname*

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 TCPIP*stackname* 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.

Best practice:
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 up.

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(IBMCS,CSTCP_SYSPLEXMON_RECOV_TCPIPstackname)
DATE(20060901)
REASON('CHECK THAT SYSPLEXMONITOR RECOVERY IS SPECIFIED WHEN DYNAMICXCF IS SPECIFIED')
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:**
| See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

| **SECLABEL recommended for multilevel security (MLS) users:**
| SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria*
| for information on using SECLABELs.

CSTCP_TCPMAXRCVBUFRSIZE_TCPIP*stackname*

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 TCPIP*stackname* 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.

Best practice:
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 up.

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(IBMCS,CSTCP_TCPMAXRCVBUFRSIZE_TCPIPstackname)
   DATE(20050214)
   REASON('ENSURE TCP RECEIVE BUFFER SIZE IS SUFFICIENT FOR FTP SERVER')
   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:
See *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMCS,CSVTAM_CSM_STG_LIMIT)
   DATE(20050214)
   REASON('CHECK MAXFIX AND MAXECSA')
   PARM('MAXFIX(100M),MAXECSA(100M)')
   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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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 .

Best practice:
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 up.

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(IBMCS,CSVTAM_T1BUF_T2BUF_EE)
DATE(20060901)
REASON('CHECK T1BUF/T2BUF ALLOCATIONS WITH EE')
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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMCS,CSVTAM_T1BUF_T2BUF_NOEE)
DATE(20060901)
REASON('CHECK T1BUF/T2BUF ALLOCATIONS WITHOUT EE')
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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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).

Best practice:
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 and up.

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(IBMCS,CSVTAM_VIT_DSPSIZE)
DATE(20060901)
REASON('CHECK VIT DSPSIZE IS AT MAXIMUM')
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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMCS,CSVTAM_VIT_OPT_ALL)
DATE(20060901)
REASON('CHECK VIT OPT=ALL IS NOT SPECIFIED')
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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMCS,CSVTAM_VIT_PSS_SMS)
DATE(20060901)
REASON('CHECK VIT PSS AND SMS OPTIONS ARE ACTIVE')
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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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).

Best practice:
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 and up.

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(IBMCS,CSVTAM_VIT_SIZE)
DATE(20060901)
REASON('CHECK VIT SIZE IS AT MAXIMUM')
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:
See *z/OS Communications Server: SNA Messages*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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,
        DATE(20050214)
        REASON('AVOID THE LOSS OF CRITICAL ACTION MESSAGES')
        ACTIVE
        SEVERITY(LOW)
        INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on AMRF, see *z/OS MVS Planning: Operations*.

Messages:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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:
sz/OS V1R4 through z/OS V1R7. This check does not apply to releases higher than z/OS V1R7.

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,
        DATE(20050214)
        REASON('ABILITY TO CONTROL THIS SYSTEM')
        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:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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).

Best practice:
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 up.

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,
    DATE(20050214)
    REASON('REDUCES THE NUMBER OF MESSAGE SENT TO A CONSOLE IN THE SYSPLEX')
    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:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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,
    DATE(20050214)
    REASON('TO REDUCE MESSAGE TRAFFIC BY REMOVING MESSAGES THAT ARE ONLY INTENDED FOR THE PROBLEM PROGRAMMER')
    ACTIVE
    SEVERITY(LOW)
    INTERVAL(24:00)
```

Parameters accepted:

No

Reference:

For more information on routing codes, see*z/OS MVS Planning: Operations*.

Messages:

See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:

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 up.

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,
        DATE(20050214)
        REASON('TO AVOID HAVING EMCS CONSOLES PROCESS AN EXCESSIVE NUMBER OF MESSAGES')
        ACTIVE
        SEVERITY(MED)
        INTERVAL(24:00)
```

Parameters accepted:

No

Reference:

For more information on EMCS consoles, see *z/OS MVS Planning: Operations*.

Messages:

See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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,
      DATE(20050214)
      REASON('REDUCES THE TIME A SYSTEM TAKES TO JOIN A SYSPLEX')
      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:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

CNZ_Syscons_Master

Description:
Ensures that the system console has MASTER authority.

Best practice:
Assign MASTER authority to the system console. The system console should have MASTER authority so that it can perform necessary recovery operations in emergency situations.

z/OS releases the check applies to:
z/OS V1R4 through V1R7.

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,
      DATE(20050214)
      REASON('NEEDED TO RESOLVE PROBLEMS IN EMERGENCY SITUATIONS')
      ACTIVE
      SEVERITY(HI)
      INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on systems consoles, see *z/OS MVS Planning: Operations*.

Messages:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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,
    DATE(20050214)
    REASON('TO ENSURE THE SYSTEM CONSOLE IS CONFIGURED TO RECEIVE MESSAGES FROM ONLY THE LOCAL SYSTEM')
    ACTIVE
    SEVERITY(MED)
    INTERVAL(24:00)
```

Parameters accepted:
No.

Reference:
For more information on systems consoles, see *z/OS MVS Planning: Operations*.

Messages:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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,
      DATE(20050214)
      REASON('SHOULD ONLY RUN IN PROBLEM DETERMINATION MODE WHEN THERE IS A PROBLEM')
      ACTIVE
      SEVERITY(MED)
      INTERVAL(01:00)
```

Parameters accepted:
No

Reference:
For more information on system consoles, see *z/OS MVS Planning: Operations*.

Messages:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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).

Best practice:
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 up.

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,
      DATE(20050214)
      REASON('TO ENSURE THE SYSTEM CONSOLE IS CONFIGURED TO RECEIVE IMPORTANT MESSAGES')
      ACTIVE
      SEVERITY(LOW)
      INTERVAL(24:00)
```

Parameters accepted:
No

Reference:
For more information on systems consoles, see *z/OS MVS Planning: Operations*.

Messages:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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,
      DATE(20050214)
      REASON('CHECK CONSOLES TASK TABLE')
      ACTIVE
      SEVERITY(LOW)
      INTERVAL(00:15)
```

Parameters accepted:
No

Reference:
N / A

Messages:
See *z/OS MVS System Messages, Vol 4 (CBD-DMO)* .

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMCSV,CSV_APF_EXISTS)
  SEVERITY(LOW) INTERVAL(04:00) DATE(20050720)
  REASON('An entry in the APF list might refer to an obsolete'
  'data set.')
```

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 SDSF CK command display.

Reference:
For more information, see:
- *z/OS MVS Programming: Authorized Assembler Services Guide*
- *z/OS MVS Initialization and Tuning Guide*

Messages:
See *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.

Best practice:
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 up.

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 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(IBMCSV,CSV_LNKLST_NEWEXTENTS)
  PARM('ALL')
  SEVERITY(HIGH) INTERVAL(01:00)   DATE(20050720)
  REASON('When the number of extents in a LNKLST PDS data set'
  'changes, I/O errors might result.')
```

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 SDSF CK command display.

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:
See *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 to see whether all active LNKLST sets on the system for data sets that were created with secondary space defined.

Best practice:
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 up.

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(IBMCSV,CSV_LNKLST_SPACE)
  SEVERITY(LOW) INTERVAL(24:00)   DATE(20050720)
  REASON('PDS data sets in a LNKLST that use only primary space'
  'are protected from problems due to extent changes.')
```

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 SDSF CK command display.

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:
See *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.

Best practice:
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 up.

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(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('Changes in LPA can affect the size of ',
                   'the private area available for ',
                   'applications.')
```

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:
Init & Tuning reference, Init & Tuning guide,

Messages:
CSVHxxxx

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

|
Global Resource Serialization checks (IBMGRS)

GRS_Mode

Description:
Checks the mode of the Global Resource Serialization complex.

Best practice:
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 up.

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(IBMGRS,GRS_MODE)
        SEVERITY(LOW) INTERVAL(ONETIME) PARM('STAR') DATE(20050105)
        REASON('GRS should run in STAR mode to improve performance.')
```

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*, SA22-7600.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
Enabling GRS synchronous reserve processing can prevent deadlock conditions.

z/OS releases the check applies to:
z/OS V1R4 and up.

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(IBMGRS,GRS_SYNCHRES)
       SEVERITY(LOW) INTERVAL(001:00) DATE(20050105)
       REASON('GRS synchronous RESERVE processing should be enabled to
       avoid deadlock conditions.')
```

Parameters accepted:
No

Reference:
For more information on GRS synchronous reserve processing, see *z/OS MVS Planning: Global Resource Serialization*, SA22-7600.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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

Best practice:
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 up.

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(IBMGRS,GRS_CONVERT_RESERVES)
     SEVERITY(LOW) INTERVAL(ONETIME) DATE(20050105)
     REASON('When in STAR mode, converting RESERVEs can help improve
     performance and avoid deadlock.')
```

Parameters accepted:
No

Reference:
For more information on GRS Reserve Conversion, see *z/OS MVS Planning: Global Resource Serialization*, SA22-7600.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMGRS,GRS_EXIT_PERFORMANCE)
      SEVERITY(LOW) INTERVAL(024:00) DATE(20050105)
      REASON('Certain exits may negatively impact system performance.')
```

Reference:
For more information on GRS installation exits, see *z/OS MVS Planning: Global Resource Serialization*, SA22-7600 and*z/OS MVS Installation Exits*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

Type of check:
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(IBMGRS,GRS_GRSQ_SETTING)
SEVERITY(LOW) INTERVAL(ONETIME) DATE(20050202)
REASON('IBM recommends a GRSQ setting of contention.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
For more information on GRSQ, see *z/OS MVS Planning: Global Resource Serialization*, SA22-7600.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

Type of check:
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(IBMGRS,GRS_RNL_IGNORED_CONV)
SEVERITY(LOW) INTERVAL(ONETIME) DATE(20050202)
REASON('If an entry in the RESERVE Conversion RNL is
   superseded by an entry in the SYSTEMX Exclusion RNL,
   it will be ignored.')
```

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*, SA22-7600.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMPDSE,PDSE_SMSPDSE1),
   SEVERITY(LOW),
       INTERVAL(ONETIME),
       DATE('20060301')
```

Debug support:
No.

Verbose support:
No

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for MLS users:
SYSLOW

RACF checks (IBMRACF)

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 39 and Table 40 on page 333. 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 39. **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.
SYSZRACF	DSDTPREP...DSDTPREP The minor name is the string DSDTPREP repeated six times.

Table 39. **Systems** *Level ENQs that RACF_GRS_RNL checks (continued)*

Major Name	Minor Name
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 40. **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
SYSZRAC2	IRRDPI80000
SYSZRAC2	RCVTDPTB000

Table 40. **System** *Level ENQs that RACF_GRS_RNL checks (continued)*

Major Name	Minor Name
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.

Best practice:
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 up.

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(20040703)
REASON('None of the RACF ENQ names should be in RNLs.')
```

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 SDSF CK command display.

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*, SA22-7600 and *z/OS Security Server RACF System Programmer's Guide*.

Messages:
See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security (MLS) 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_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.

Best practice:
The system is critically exposed if these resources are not properly protected.

z/OS releases the check applies to:
z/OS V1R4 and up.

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(20040801)
PARM(GENUSER)
REASON('Testing with GENUSER value')
```

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(20040703)
REASON('Sensitive resources should be protected.')
```

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 SDSF CK command display.

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:
See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security (MLS) 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.

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)
 START TIME: 03/02/2006 18:51:19.175232
 CHECK DATE: 20040703  CHECK SEVERITY: HIGH

                            APF Dataset Report

 S Data Set Name                            Vol    UACC Warn ID*  User
 - -------------------------------------- ------ ---- ---- ---- ----
 E ASM.SASMMOD1                             ZDR18
 V ATC.V2R1M4.AUTHLIB                       DRVPSL
 V CBC.SCBCCMP                              ZDR18
 E CBC.SCCNCMP                              ZDR18
 E CBC.SCLBDLL                              ZDR18
 E CBC.SCLBDLL2                             ZDR18
 E CEE.SCEERUN                              ZDR18
 E CEE.SCEERUN2                             ZDR18
 E CRAIGJ.VTAMLIB                           D94RF2
 E CSF.SCSFMOD0                             ZDR18
 E EOY.SEOYLOAD                             ZDR18
 V EUVF.SEUVFLOD                            ZDR18
 E FARRELL.TEST.TOOLS.LOAD                  D94001
 E FFST.V120ESA.SEPWMOD1                    ZDR18
 E FFST.V120ESA.SEPWMOD2                    ZDR18
 E GAIL.AUTH.LOAD                           D94RF1
 E GDDM.SADMMOD                             ZDR18
 E GIM.SGIMLMD0                             ZDR18
 V GLD.SGLDLOD                              ZDR18
 V GSK.SGSKLOD                              ZDR18
 V ING.SINGMOD1                             ZDR18
 V ING.SINGMOD2                             ZDR18
 E IOE.SIOELMOD                             ZDR18
 E ISF.SISFLINK                             ZDR18
```

```
E ISF.SISFLOAD                          ZDR18
E ISP.SISPLOAD                          ZDR18
E ISP.SISPLPA                           ZDR18
V ISP.SISPSASC                          ZDR18
V ISPF350.ISPLOAD                       PRODAL
V ISPF350.ISPLPA                        PRODAL
V ISPF350.ISRLOAD                       PRODAL
V ISPF350.ISRLPA                        PRODAL
V ISPF350.LPALIB                        PRODAL
V MARUSEK.AUTH.LOAD                     D79PK2
E MSPCT.ZOS16ZTT.LOADLIB                CTTPAK
E MSPCT.ZOS17ZTT.LOADLIB                CTTPAK
E MVSSTORE.SRVLIB.ZOS15.LIBS            DRVPSL
E MVSSTORE.SRVLIB.ZOS15.LPA             DRVPSL
E MVSSTORE.SRVLIB.ZOS15.NUCLEUS         DRVPSL
V NETVIEW.V1R4M0.CNMLINK                ZDR18
E RACFDRVR.ATC.AUTHLIB                  D79PK5
E RACFL2.LINKLIB                        D94RF1
E RACFTEST.ADAU.LOAD                    D94RF2
E RACFTEST.RRSF.LOAD                    D94RF2
V RACF318.ASAP.MIGLIB                   D97107
E RACF318.MIGLIB                        D97107
V RACF318.NEW.MIGLIB                    D97107
  SYS1.CMDLIB                           ZDR18  None No   ****
  SYS1.DFQLLIB                          ZDR18  None No   ****
  SYS1.DGTLLIB                          ZDR18  None No   ****
V SYS1.ISAMLPA                          ZDR18
  SYS1.LINKLIB                          ZDR18  None No   ****
V SYS1.NFSLIB                           ZDR18
  SYS1.RDHARDG.LINKLIB                  D94RF1 None No   ****
  SYS1.SBDTLIB                          ZDR18  None No   ****
  SYS1.SBDTLINK                         ZDR18  None No   ****
  SYS1.SCBDHENU                         ZDR18  None No   ****
V SYS1.SCUNIMG                          ZDR18
  SYS1.SERBLINK                         ZDR18  None No   ****
V SYS1.SHASLINK                         ZDR18
  SYS1.SHASLNKE                         ZDR18  None No   ****
  SYS1.SHASMIG                          ZDR18  None No   ****
  SYS1.SIATLIB                          ZDR18  None No   ****
  SYS1.SIATLINK                         ZDR18  None No   ****
  SYS1.SIATLPA                          ZDR18  None No   ****
  SYS1.SIATMIG                          ZDR18  None No   ****
  SYS1.SICELINK                         ZDR18  None No   ****
  SYS1.SIEALNKE                         ZDR18  None No   ****
  SYS1.SIOALMOD                         ZDR18  None No   ****
  SYS1.SISTCLIB                         ZDR18  None No   ****
  SYS1.SVCLIB                           ZDR18  None No   ****
  SYS1.VTAMLIB                          ZDR18  None No   ****
  TCPIP.SEZADSIL                        ZDR18  None No   ****
V TCPIP.SEZALINK                        ZDR18
  TCPIP.SEZALNK2                        ZDR18  None No   ****
  TCPIP.SEZALOAD                        ZDR18  None No   ****
  TCPIP.SEZATCP                         ZDR18  None No   ****

                           RACF Dataset Report

S Data Set Name                         Vol    UACC Warn ID*  User
- ------------------------------------- ------ ---- ---- ---- ----
E RACFDRVR.RACF318                      RDB318

                          PARMLIB Dataset Report

S Data Set Name                         Vol    UACC Warn ID*  User
- ------------------------------------- ------ ---- ---- ---- ----
E RACFDRVR.PARMLIB                      D94RF4
  RACFDRVR.PARMLIB.R10                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R12                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R13                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R14                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R15                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R16                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R17                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R18                  D94RF4 None No   ****
  RACFDRVR.PARMLIB.R4                   D94RF4 None No   ****
  RACFDRVR.PARMLIB.R6                   D94RF4 None No   ****
  RACFDRVR.PARMLIB.R8                   D94RF4 None No   ****
  SYS1.PARMLIB                                 None No   ****
  SYS1.PARMLIB.INSTALL                  ZDR18  None No   ****
  SYS1.PARMLIB.POK                      ZDR18  None No   ****

                    Current Link List Dataset Report

S Data Set Name                         Vol    UACC Warn ID*  User
- -------------------------------------- ------ ---- ---- ---- ----
E ASM.SASMMOD1                          ZDR18
E ATC.V2R1M4.SATGBMOD                   D94RF1
E CBC.SCLBDLL                           ZDR18
E CEE.SCEERUN                           ZDR18
E COMMON.LOOKFEEL.LINKLIB               ZDR18
E CSF.SCSFMOD0                          ZDR18
E EOY.SEOYLOAD                          ZDR18
E FARRELL.TEST.TOOLS.LOAD               D94001
E FFST.V120ESA.SEPWMOD2                 ZDR18
E GDDM.SADMMOD                          ZDR18
E GIM.SGIMLMD0                          ZDR18
E ISF.SISFLINK                          ZDR18
E ISF.SISFLOAD                          ZDR18
E ISP.SISPLOAD                          ZDR18
E MSPCT.OSR10CTT.LOADLIB                CTTPAK
E MSPCT.OSR12CTT.LOADLIB                CTTPAK
E MSPCT.OSR13CTT.LOADLIB                CTTPAK
E MSPCT.OSR14CTT.LOADLIB                CTTPAK
E MSPCT.OSR15CTT.LOADLIB                CTTPAK
E MSPCT.OSR16CTT.LOADLIB                CTTPAK
E MSPCT.ZOS16ZTT.LOADLIB                CTTPAK
E RACFTEST.LOAD                         D94RF1
E RACF318.LINKLIB                       D97107
E RACF318.MIGLIB                        D97107
  SYS1.CMDLIB                           ZDR18  None No   ****
  SYS1.CSSLIB                           ZDR18  None No   ****
  SYS1.DFQLLIB                          ZDR18  None No   ****
  SYS1.DGTLLIB                          ZDR18  None No   ****
  SYS1.LINKLIB                          ZDR18  None No   ****
  SYS1.MIGLIB                           ZDR18  None No   ****
  SYS1.SCUNIMG                          ZDR18Y None No   ****
  SYS1.SERBLINK                         ZDR18  None No   ****
  SYS1.SHASLNKE                         ZDR18  None No   ****
  SYS1.SHASMIG                          ZDR18  None No   ****
  SYS1.SIATLIB                          ZDR18  None No   ****
  SYS1.SIATLINK                         ZDR18  None No   ****
  SYS1.SIATLPA                          ZDR18  None No   ****
  SYS1.SICELINK                         ZDR18  None No   ****
  SYS1.SIEALNKE                         ZDR18  None No   ****
  SYS1.SIEAMIGE                         ZDR18  None No   ****
  SYS1.SIOALMOD                         ZDR18  None No   ****
  SYS1.SORTLIB                          ZDR18  None No   ****
  SYS1.VTAMLIB                          ZDR18  None No   ****
E SYS2.CSSLIB                           ZDR18
E SYS2.LINKLIB                          ZDR18
E SYS2.MIGLIB                           ZDR18
E SYS2.SIEALNKE                         ZDR18
E SYS2.SIEAMIGE                         ZDR18
  TCPIP.SEZALOAD                        ZDR18  None No   ****


                  System Rexx Dataset Report

S Data Set Name                         Vol    UACC Warn ID*  User
- -------------------------------------- ------ ---- ---- ---- ----
E SYS1.SAXREXEC                         ZDR19

                   Sensitive General Resources Report

S Resource Name                         Class    UACC Warn ID*  User
```

```
- -------------------------------------- -------- ---- ---- ---- ----
  BPX.DAEMON                             FACILITY None No   ****
  BPX.FILEATTR.APF                       FACILITY None No   ****
  BPX.SERVER                             FACILITY None No   ****
  BPX.SUPERUSER                          FACILITY None No   ****
  ICHBLP                                 FACILITY None No   ****
  IRR.PASSWORD.RESET                     FACILITY
  MVS.SET.PROG                           OPERCMDS
  MVS.SETPROG                            OPERCMDS
E ACCT                                   TSOAUTH  Updt No   ****
E CONSOLE                                TSOAUTH  None Yes  ****
E OPER                                   TSOAUTH  None No   Updt
E PARMLIB                                TSOAUTH  None No   Read
E TESTAUTH                               TSOAUTH  None No   Read
  SUPERUSER.FILESYS                      UNIXPRIV
  SUPERUSER.FILESYS.CHANGEPERMS          UNIXPRIV
  SUPERUSER.FILESYS.CHOWN                UNIXPRIV

* 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. Any data set which has an
    "M" in the "S" (Status) field has been migrated.

    The APF_LIBS check provides additional analysis of the non-RACF
    aspects of your APF list.

    If the "S" field contains an "E" or is blank, then blanks 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: 03/02/2006 18:51:39.285499  STATUS: EXCEPTION-HIGH
```

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

Best practice:
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 up.

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(20051111)
REASON('The classname class should be active.')
```

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 SDSF CK command display.

Verbose support:
No.

Reference:
For more information on storage increments, see *z/OS Security Server RACF Security Administrator's Guide* .

Messages:
See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security (MLS) 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 at your installation activate this class and
     define in it the profiles to properly protect your system.

   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 Auditor's Guide and the RACF
    Systems 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:  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.

Best practice:
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 up.

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(20051111)
REASON('IBM recommends that the user ID IBMUSER is revoked.')
```

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 SDSF CK command display.

Verbose support:
No.

Reference:
For more information on storage increments, see *z/OS Security Server RACF Security Administrator's Guide* .

Messages:
See *z/OS Security Server RACF Messages and Codes*.

SECLABEL recommended for multilevel security (MLS) 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
Systems 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
```

RRS checks (IBMRRS)

RRS_RMDataLogDuplexMode

Description:
The duplexing scheme used to protect the RM Data log stream is evaluated.

Best practice:
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 up.

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(IBMRRS,RRS_RMDATALOGDUPLEXMODE)
    SEVERITY(MEDIUM),INTERVAL(8:00),DATE(20050115)
    REASON('RM Data log should use a better duplexing scheme than local
buffer duplexing.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMRRS,RRS_RMDOFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE(20050115)
    REASON('RM Data log offload dataset size should be at least as
large as the space allocated for the log stream in the structure.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMRRS,RRS_DUROFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE(20050115)
    REASON('Delayed UR log offload dataset size should be at least as
large as the space allocated for the log stream in the structure.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMRRS,RRS_MUROFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE(20050115)
    REASON('Main UR log offload dataset size should be at least as
large as the space allocated for the log stream in the structure.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMRRS,RRS_RSTOFFLOADSIZE)
    SEVERITY(LOW),INTERVAL(8:00),DATE(20050115)
    REASON('Restart log offload dataset size should be at least as
large as the space allocated for the log stream in the structure.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

RRS_ArchiveCFStructure

Description:
The check evaluates the coupling facility structure in which the RRS Archive log resides.

Best practice:
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 up.

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(IBMRRS,RRS_ARCHIVECFSTRUCTURE)
    SEVERITY(LOW),INTERVAL(8:00),DATE(20051013)
    REASON('RRS Archive log stream is sharing its coupling facility structure
            with another log stream.  This is not reccomended.')
```

Parameters accepted:
No

Reference:
For more information, see *z/OS MVS Programming: Resource Recovery*.

Debug support:
No

Verbose support:
No

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

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.

Best practice:
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 up in z/Architecture mode only.

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(IBMRSM,RSM_HVSHARE),
INTERVAL(00:15),
SEVERITY(LOW),
PARM('THRESHOLD(80%),SIZE(510T)'),
DATE('20041006')
```

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:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 ensure that the MEMLIMIT setting in SMFPRMxx is not overlooked.

z/OS releases the check applies to:
z/OS V1R4 and up in z/Architecture mode only.

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(IBMRSM,RSM_MEMLIMIT),
    INTERVAL(ONETIME),
    SEVERITY(LOW),
    DATE('20041006')
```

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:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up in z/Architecture mode only.

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(IBMRSM,RSM_MAXCADS),
    INTERVAL(00:15),
    SEVERITY(MED),
    PARM('THRESHOLD(80%)'),
    DATE('20041006')
```

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:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, in both ESA and z/Architecture modes.

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(IBMRSM,RSM_AFQ),
    INTERVAL(ONETIME),
    SEVERITY(HI),
    PARM('AFQLOW(400),AFQOK(600)'),
    DATE('20041006')
```

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*, SA22-7592. For more information on using the Paging Activity report, see *z/OS RMF Report Analysis*, SC33-7991 and the whitepaper, WP100269 "z/OS Performance: Managing Processor Storage in a 64–bit environment".

Messages:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, in both ESA and z/Architecture modes.

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(IBMRSM,RSM_REAL),
    INTERVAL(ONETIME),
    SEVERITY(LOW),
    DATE('20041006')
```

Parameters accepted:
No.

Reference:
For more information on real storage, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

Messages:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, in both ESA and z/Architecture modes.

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(IBMRSM,RSM_RSU),
INTERVAL(ONETIME),
SEVERITY(LOW),
DATE('20041006')
```

Parameters accepted:
No.

Reference:
For more information on reconfigurable storage, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

Messages:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up.

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(IBMSDUMP,SDUMP_AVAILABLE)
SEVERITY(MEDIUM)
INTERVAL(OneTime)
DATE(20050301)
REASON('Example for SCDMP SDUMP checker')
```

Parameters accepted:
No.

Reference:
For more information on SDUMP, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Messages:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
Automatic allocation of SVC dump data sets.

z/OS releases the check applies to:
z/OS V1R4 and up.

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(IBMSDUMP,SDUMP_AUTO_ALLOCATION)
SEVERITY(MEDIUM)
INTERVAL(OneTime)
DATE(20050301)
REASON('Example for SCDMP SDUMP checker')
```

Parameters accepted:
No.

Reference:
For more information, see *z/OS MVS Diagnosis: Tools and Service Aids*, GA22-7589.

Messages:
See *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Supervisor (IBMSUP)

IEA_ASIDS

Description:
This check reports on available ″normal″ and ″replacement″ ASIDs

Best practice:
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 up.

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(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 0-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 0-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 0-99999. 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:
z/OS MVS System Messages, Vol 6 (GOS-IEA).

SECLABEL recommended for multilevel security (MLS) 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)

Best practice:
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 up.

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(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:
z/OS MVS System Messages, Vol 6 (GOS-IEA).

SECLABEL recommended for multilevel security (MLS) 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:
Checks for staging data sets that have encountered full conditions. SMF must be active for system logger to report on staging dataset full conditions.

Best practice:
IBM suggests that tuning actions be taken to avoid future full conditions.

z/OS releases the check applies to:
z/OS V1R7 and up.

Parameters accepted:
No.

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)
       SEVERITY(LOW) INTERVAL(4:00) DATE(20050808)
       REASON('Logger staging dataset full conditions should be investigated to
       determine if application performance is being impacted')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

IXGLOGR_ENTRYTHRESHOLD

Description:
Checks for entry threshold reached conditions. SMF must be active for system logger to report on entry threshold reached conditions.

Best practice:
IBM suggests that tuning actions be taken to avoid future entry full conditions.

z/OS releases the check applies to:
z/OS V1R7 and up.

Parameters accepted:
No.

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)
       SEVERITY(LOW) INTERVAL(4:00) DATE(20050808)
       REASON('Logger entry threshold reached conditions should be investigated
   to determine if applications performance is being impacted')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

IXGLOGR_STRUCTUREFULL

Description:
Checks for structure element full conditions. SMF must be active for system logger to report on structure element full conditions.

Best practice:
IBM suggests that tuning actions be taken to avoid structure element full conditions.

z/OS releases the check applies to:
z/OS V1R7 and up.

Parameters accepted:
No.

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
       SEVERITY(LOW) INTERVAL(4:00) DATE(20050808)
       REASON('Logger structure full conditions should be investigated to
   determine if application performance is being impacted')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

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.

Best practice:
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 up.

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(IBMTSOE,TSOE_USERLOGS)
  SEVERITY(LOW) INTERVAL(24:00) DATE(20060220)
  REASON('User logs should be in use')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No

Reference:
See *z/OS TSO/E Customization*

Messages:
See *z/OS TSO/E Messages*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The outputs for the report that TSOE_USERLOGS produces is shown below:

```
IKJH0201I User Logs are in use


IKJH0202E User logs are not in use
```

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.

Best practice:
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 up.

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(IBMTSOE,TSOE_PARMLIB_ERROR)
  SEVERITY(LOW) INTERVAL(24:00) DATE(20060220)
  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:
See *z/OS TSO/E Messages*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Output: The outputs for the report that TSOE_PARMLIB_ERROR produces is shown below:

```
IKJH0301I All TSO/E PARMLIB settings were initialized successfully.

IKJH0302E TSO/E PARMLIB defaults were used for AUTHCMD due to an error.
```

z/OS UNIX System Services checks (IBMUSS)

USS_AUTOMOUNT_DELAY

Description:
Automount delay configuration values in a sysplex are at least 10 minutes

Best practice:
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 up.

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(IBMUSS,USS_AUTOMOUNT_DELAY)
SEVERITY(MED)
INTERVAL(24:00)
PARM('DELAY=10')
DATE(20050224)
REASON('Sample for USS_AUTOMOUNT_DELAY with default values.')
```

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:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

USS_FILESYS_CONFIG

Description:

Evaluates the file system configuration, which includes:

- AUTOMOVE setup verification
- zFS for a multilevel security (MLS) configuration.
- Mode (either RDWR/READ) of the root, system specific, and version HFSs.

Best practice:

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 up. On a z/OS V1R4 system, the check does not evaluate zFS for an MLS configuration.

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(IBMUSS,USS_FILESYS_CONFIG)
SEVERITY(HI)
INTERVAL(24:00)
PARM('SYSPLEX')
DATE(20050224)
REASON('Sample for USS_FILESYS_CONFIG with default values.')
```

Debug support:

No

Verbose support:

No

Parameters accepted:

Yes:

- PARM('SYSPLEX') - Specifies that the check verify file system configuration. PARM(SYSPLEX) is the default.
- PARM('NOPLEX') - Verifies the MLS configuration.

Use SYSPLEX (the default) if you're running in a sysplex. Change the parameter to NOPLEX if you're running with a MONOPLEX configuration.

Reference:
For more information on file systems, see *z/OS UNIX System Services Planning*, GA22-7800, and APAR II3129.

Messages:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

USS_MAXSOCKETS_MAXFILEPROC

Description:
MAXSOCKETS (AF_INET) and MAXFILEPROC are set high enough

Best practice:
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 up.

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(IBMUSS,USS_MAXSOCKETS_MAXFILEPROC)
SEVERITY(LOW)
INTERVAL(24:00)
PARM('MAXSOCKETS=64000,MAXFILEPROC=64000')
DATE(20050224)
REASON('Sample for USS_MAXSOCKETS_MAXFILEPROC with default values.')
```

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:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) 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. The z/OS UNIX dynamic settings that will be checked are:

AUTOCVT
MAXPROCSYS
MAXPROCUSER
MAXUIDS
MAXFILEPROC
MAXTHREADTASKS
MAXTHREADS
MAXPTYS
MAXFILESIZE
MAXCORESIZE
MAXASSIZE
MAXCPUTIME
MAXMMAPAREA
MAXSHAREPAGES
SHRLIBRGNSIZE
SHRLIBMAXPAGES
PRIORITYGOAL
IPCMSGNIDS
IPCMSGQBYTES
IPCMSGQMNUM
IPCSEMNIDS
IPCSEMNOPS
IPCSEMNSEMS
IPCSHMMPAGES
IPCSHMNIDS
IPCSHMNSEGS
IPCSHMSPAGES
FORKCOPY
SUPERUSER
TTYGROUP
STEPLIBLIST
USERIDALIASTABLE
LIMMSG
PRIORITYPG
FILESYSTYPE
VERSION
ROOT/MOUNT FILESYSTEM
NETWORK
SYSCALL_COUNTS
MAXQUEUEDSIGS
AUTHPGMLIST

- 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
  – PARM subparameter
- For the NETWORK statement, only the MAXSOCKETS value will be checked for AF_INET and AF_INET6.

Best practice:
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 z/OS UNIX is initialized.

z/OS releases the check applies to:
z/OS V1R9 and up.

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(IBMUSS,USS_PARMLIB)
  SEVERITY(LOW) INTERVAL(01:00) DATE(20060112)
  REASON('Reconfiguration settings should be kept in a'
    'permanent location so they are available'
    'the next time z/OS UNIX is initialized.')
```

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:
See *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

SECLABEL recommended for multilevel security (MLS) 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.
```

VSAM checks (IBMVSAM)

VSAMRLS_DIAG_CONTENTION

Description:

Checks for VSAM RLS contention by looking at registered resources. Check displays a contention table if detected.

Best practice:
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 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(IBMVSAMRLS,VSAMRLS_DIAG_CONTENTION)
SEVERITY(HI)
INTERVAL(00:05)
DATE(20060524)
PARMS('ROWS(20)')
REASON('Gives the customer ability to monitor VSAM RLS contention.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
Yes, specify an integer, minimum 1, indicating the maximum number of rows to be displayed in the contention table, (keyword: ROWS).

Reference:
For additional information see VSAM RLS Latch Contention in *z/OS DFSMSdfp Diagnosis*.

Messages:
z/OS MVS System Messages, Vol 9 (IGF-IWM)

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

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.

Best practice:
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.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 permenent 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 up.

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(IBMVSAM,VSAM_INDEX_TRAP)
ACTIVE
SEVERITY(MED) INTERVAL(24:00) DATE(20070122)
REASON('IBM recommends running with the VSAM Index Trap enable.')
```

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:
z/OS MVS System Messages, Vol 6 (GOS-IEA).

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSAMRLS_SINGLE_POINT_FAILURE

Description:
Verifies that the SHCDSs are on unique volumes.

Best practice:
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 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(IBMVSAMRLS,VSAMRLS_SINGLE_POINT_FAILURE)
SEVERITY(HI)
INTERVAL(24:00)
DATE(20060610)
REASON('Prevents a single point of failure due to a lost volumes.')
```

Debug support:
No

Verbose support:
No

Parameters accepted:
No.

Reference:
For additional information see Defining Sharing Control Data Sets in *z/OS DFSMS Storage Administration Reference*

Messages:
z/OS MVS System Messages, Vol 9 (IGF-IWM)

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

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 ALLOWERUSERKEYCSA(YES|NO) DIAGxx option and compares it to the IBM recommended setting of ALLOWUSERKEYCSA(NO). A warning is issued if the setting is YES either by default or through explicit specification as ALLOWERUSERKEYCSA(YES).

Best practice:
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, however, that 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(YES), to maintain compatibility with the behavior of prior z/OS releases.

z/OS releases the check applies to:
z/OS V1R8 and up.

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),
INTERVAL(ONETIME),
SEVERITY(LOW),
DATE('20060201')
```

Debug support:
No

Verbose support:
No

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference* and *z/OS MVS System Codes*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

```
IGVH110E The ALLOWUSERKEYCSA DIAGxx option has not been specified
Explanation: The ALLOWUSERKEYCSA DIAGxx option has not been
specified, and has been defaulted by the system to
ALLOWUSERKEYCSA(YES). Although the default is YES for compatibility,
IBM recommends that you specify ALLOWUSERKEYCSA(NO) in order to
prevent user key CSA from being obtained. User key CSA creates a
security risk because any unauthorized program can modify it.
However, specifying ALLOWUSERKEYCSA(NO) may cause programs obtaining
this storage to fail.

System Action: The system continues processing.

Operator Response: Please report this problem to the system
programmer.

System Programmer Response: Consider coding ALLOWUSERKEYCSA(NO) in
the DIAGxx parmlib member. You may issue the SET DIAG= command to
have your changes take immediate effect.

Problem Determination: n/a

Source: Virtual Storage Manager
Reference Documentation: z/OS MVS Initialization and Tuning Reference

Automation: n/a

Check Reason: Validate the AllowUserKeyCSA DIAGxx Setting
```

VSM_CSA_LIMIT

Description:
The current size of CSA against a minimum suggested value.

Best practice:
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 up, 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 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_LIMIT),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('CSA(512K),ECSA(512K)'),
DATE('20040405')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, 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 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_SQA_LIMIT),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('SQA(512K),ESQA(8M)'),
DATE('20040405')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, 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 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_PVT_LIMIT),
INTERVAL(ONETIME),
SEVERITY(LOW),
PARM('PVT(1M),EPVT(512M)'),
DATE('20040405')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, in both ESA and z/Architecture modes.

Parameters accepted:
- An integer, 0-100, indicating the threshold percent for utilization of CSA below the line, or a size in bytes with an optional suffix (K,M) indicating the minimum amount of unallocated CSA. (keyword: CSA, if a percentage value is given, a percent sign '%' must be included)
- An integer, 0-100, indicating the threshold percent for utilization of CSA above the line, or a size in bytes with an optional suffix (K,M) indicating the minimum amount of unallocated ECSA. (keyword: ECSA, if a percentage value is given, a percent sign '%' must be included)
- Default: CSA(80%),ECSA(80%)

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_THRESHOLD),
INTERVAL(00:05),
SEVERITY(HIGH),
PARM('CSA(80%),ECSA(80%)'),
DATE('20040405')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

VSM_SQA_THRESHOLD

Description:
The current allocation of SQA storage.

Best practice:
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 up, in both ESA and z/Architecture modes.

Parameters accepted:

- An integer, 0-100, indicating the threshold percent for utilization of SQA below the line, or a size in bytes with an optional suffix (K,M) indicating the minimum amount of unallocated SQA. (keyword: SQA, if a percentage value is given, a percent sign '%' must be included)
- An integer, 0-100, indicating the threshold percent for utilization of ESQA above the line, or a size in bytes with an optional suffix (K,M) indicating the minimum amount of unallocated ESQA . (keyword: ESQA , if a percentage value is given, a percent sign '%' must be included)
- Default:SQA(80%),ESQA(80%)

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_SQA_THRESHOLD),
INTERVAL(00:15),
SEVERITY(MED),
PARM('SQA(80%),ESQA(80%)'),
DATE('20050405')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) 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.

Best practice:
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 up, in both ESA and z/Architecture modes.

Parameters accepted:
- Number of bytes with optional suffix (K,M) indicating the maximum acceptable change in CSA or private below the line before an exception is issued. (keyword: BELOW)
- Number of bytes with optional suffix (K,M) indicating the maximum acceptable change in CSA or private above the line before an exception is issued. (keyword: ABOVE)
- Default: BELOW(1M),ABOVE(10M)

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_CHANGE),
INTERVAL(ONETIME),
SEVERITY(HIGH),
PARM('BELOW(1M),ABOVE(10M)'),
DATE('20040405')
```

Reference:
For more information, see *z/OS MVS Initialization and Tuning Reference*.

Messages:
See *z/OS MVS System Messages, Vol 9 (IGF-IWM)*.

SECLABEL recommended for multilevel security (MLS) users:
SYSLOW - see *z/OS Planning for Multilevel Security and the Common Criteria* for information on using SECLABELs.

Cross system coupling facility (XCF) checks (IBMXCF)

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.

Best practice:
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 up.

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_CONNECTIVITY)
        SEVERITY(MED) INTERVAL(004:00) DATE(20050130)
        REASON('Avoid problems with CF Connectivity.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_FDI

Description:
Check that the XCF failure detection interval (FDI) equates to the formula ″multiplier * SPINTIME + increment″. The FDI is the amount of time a system can be stopped before it is considered ’status update missing’.

Best practice:
It is recommended that the FDI be set to 2 times the default spin loop timeout interval (SPINTIME in the EXSPATxx parmlib member) plus 5 seconds to balance the reduction of sympathy sickness with the risk of terminating a system that will recover.

z/OS releases the check applies to:
z/OS V1R4 and up.

Parameters accepted:
1. multiplier - (must be an integer in the range of 0 to 86400)
2. increment - (must be an integer in the range of 0 to 86400)

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_FDI)
         SEVERITY(MED) INTERVAL(004:00) DATE(20050130)
         PARM('2,5')
         REASON('Allow adequate time to recover from spin
                 situation before system is assumed to have
                 failed.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_SFM_ACTIVE

Description:
Check that the status of Sysplex Failure Management (SFM) policy is as recommended.

Best practice:
It is recommended that Sysplex Failure Management (SFM) be active to handle failure conditions in a sysplex with little or no operator involvement.

z/OS releases the check applies to:
z/OS V1R4 and up.

Parameters accepted:
Recommended SFM status (must be either ACTIVE or INACTIVE)

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_SFM_ACTIVE)
         SEVERITY(MED) INTERVAL(004:00) DATE(20050130)
         PARM('ACTIVE')
         REASON('An SFM policy provides better failure
                 management.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
It is recommended that the XCF cleanup time be set to 15 seconds.

z/OS releases the check applies to:
z/OS V1R4 and up.

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 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_CLEANUP_VALUE)
        SEVERITY(MED) INTERVAL(004:00) DATE(20050130)
        PARM('15')
        REASON('Quick removal of a partitioned system from the
               SYSPLEX.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

XCF_CDS_SEPARATION

Description:
Check that sysplex couple data set and function couple data sets are properly isolated with alternates

Best practice:
It is recommended that the SYSPLEX, CFRM and LOGR primary couple data sets reside on different volumes. It is also recommended that each primary couple data set reside on a different volume than its corresponding alternate couple data set.

z/OS releases the check applies to:
z/OS V1R4 and up.

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_CDS_SEPARATION)
        SEVERITY(HI) INTERVAL(001:00) DATE(20050130)
        REASON('Ensure that CDS separation has been maintained.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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 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_SYSPLEX_CDS_CAPACITY)
        SEVERITY(MED) INTERVAL(000:30) DATE(20070425)
        PARM('1,2,5')
        REASON('Check sysplex CDS capacities.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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_TCLASS_HAS_UNDESIG)
        SEVERITY(LOW) INTERVAL(024:00) DATE(20050130)
        REASON('Avoid problems with XCF signalling.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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 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_TCLASS_CONNECTIVITY)
        SEVERITY(MED) INTERVAL(004:00) DATE(20050130)
        PARM('1')
        REASON('Avoid problems with XCF signalling.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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 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_TCLASS_CLASSLEN)
         SEVERITY(MED) INTERVAL(024:00) DATE(20050130)
         PARM('2')
         REASON('Avoid problems with XCF signalling.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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 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_SIG_CONNECTIVITY)
         SEVERITY(MED) INTERVAL(000:30) DATE(20050130)
         PARM('2')
         REASON('Avoid single points of failure problems with XCF signaling.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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 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_DEFAULT_MAXMSG)
         SEVERITY(LOW) INTERVAL(024:00) DATE(20050130)
         PARM('2000')
         REASON('Avoid problems with XCF signalling.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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 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_MAXMSG_NUMBUF_RATIO)
         SEVERITY(MED) INTERVAL(004:00) DATE(20050130)
         PARM('30')
         REASON('Avoid problems with XCF signalling.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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 systems which are connected.

Best practice:
It is recommended that there are no single points of failure for paths.

z/OS releases the check applies to:
z/OS V1R4 and up.

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_SIG_PATH_SEPARATION)
         SEVERITY(MED) INTERVAL(000:30) DATE(20050130)
         REASON('Avoid problems with XCF signalling in CFs.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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 signalling structure entries to support full connectivity in the sysplex.

Best practice:
It is recommended that there be at least 20 available list entries per list to allow full connectivity.

z/OS releases the check applies to:
z/OS V1R4 and up.

Parameters accepted:
The minimum number of entries available for each list. (must be an integer in the range of 1 to 999999)

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_SIG_STR_SIZE)
         SEVERITY(MED) INTERVAL(008:00) DATE(20050130)
         PARM('20')
         REASON('Avoid problems with XCF signalling in CFs.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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 policy.

Best practice:
It is recommended that structure placement is in accordance with the preference list.

z/OS releases the check applies to:
z/OS V1R4 and up.

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_PREFLIST)
         SEVERITY(MED) INTERVAL(008:00) DATE(20050130)
         REASON('Check Structure preference lists.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *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.

Best practice:
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 up.

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_EXCLLIST)
        SEVERITY(MED) INTERVAL(008:00) DATE(20050130)
        REASON('Check Structure exclusion lists.')
```

Reference:
For more information, see *z/OS MVS Setting Up a Sysplex*.

Messages:
See *z/OS MVS System Messages, Vol 10 (IXC-IZP)*.

SECLABEL recommended for MLS users:
SYSLOW

Appendix. Accessibility

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.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`

Notices

This information was developed for products and services offered in the USA.

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
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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.

Programming Interface Information

This book documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS.

Trademarks

Index

Readers' Comments — We'd Like to Hear from You

z/OS
IBM Health Checker for z/OS User's Guide
Version 1 Release 9

Publication No. SA22-7994-05

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrcfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SA22-7994-05

Cut or Fold Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape **Please do not staple** Fold and Tape

SA22-7994-05

Cut or Fold Along Line

Program Number: 5694-A01

Printed in USA

SA22-7994-05