#SHAREorg

IBM Health Checker for z/OS - V2R1 Updates - Check writing details and comparisons

Peter Relson
Ulrich Thiemann
IBM

August 13th, 2013
Session 14232

SHARE in Boston

Trademarks

- See URL http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.
- The term Health Checker is used as short form of “IBM Health Checker for z/OS” in this presentation.
- The term “health check” or just “check” is used as short form of “health check for the IBM Health Checker for z/OS” in this presentation.

Session Objectives

- Part 1
    - Learn about new features of Health Checker for z/OS V2R1
        - “Auto”-start and new system & procedure parameters
        - HZSPRINT using PARMDD, …
- Part 2
    - Learn about check writing, check types, and languages
        - Many details not mentioned in the overview session 14298

Why “auto”-start?

- For many releases Health Checker has been helping
  - to ensure system configuration best practices,
  - to prevent system outages,
  - to successfully migrate from one z/OS release to another.

- Health Checker was not “ON” by default though and many opportunities to prevent system problems have been missed still.

The “auto”-start

- System will schedule start of Health Checker address space at IPL time
  - Via started procedure HZSPROC

- No more “START HZSPROC” in COMMNDxx needed

- For this and the topics on the next few pages see also “Convert your existing IBM Health Checker for z/OS set-up for automatic start-up” in the V2.1 z/OS Migration book

Used Health Checker before?

- Might have to use new system parameter HZSPROC
  - If you renamed your HZSPROC to *myhcproc*
    - Set HZSPROC=*myhcproc* in IEASYSxx
  - Could just rename procedure back to HZSPROC
    - But would need to adjust security setup tied to procedure name

Used Health Checker before? - Continued

- Remove “START HZSPROC[,HZSPRM=xx]”
  - From COMMNDxx or automation products
  - Move xx to sysparm HZS and adjust HZSPRM
    - see later slides for details
- Until then, “auto”-started instance will get precedence
  - System will reject START and “remind” you:
    - HZS0101I – “...HEALTH CHECKER... IS ALREADY ACTIVE”
      - *When the IPL-time instance is already up and running*
    - HZS0116I – “...HEALTH CHECKER... START PENDING
      - *When the IPL-time instance is still initializing*
      - *Has emergency 3-strikes rule though*

Have not used Health Checker before?

- See session 14298 for Health Checker introduction
- Be prepared to initially handle a number of check “exceptions” via messages like HZS0001I (low severity), HZS0002E (medium), and HZS0003E (high)
  - Refer to individual check message and details in a check's message buffer on how to “fix” those exceptions
  - Sometimes your installation might follow different “best practices” and you should customize the check behavior/parameters via HZSPRMxx parmlib members to avoid future exception messages

- See also “Managing checks” in the Health Checker User's Guide

Do not want to start using Health Checker?

- That might be OK
  - For a virtual / test system
  - In general for a system that is OK to fail / be re-IPLed often
- Otherwise...
  - Reconsider and do not let an initial “rush” of health check exceptions prevent you from taking advantage of this preventative tool!

- If you really have to
  - Disable “auto”-start via HZSPROC=*NONE in IEASYSxx
  - Can also be used to let automation product start HZSPROC in controlled manner instead

New – More independence for HZSPROC

- Procedure HZSPROC can be made more generic and easier to share between systems via
  - New system parameter HZS for list of HZSPRMxx parmlib member suffixes
  - New HZSPDATA statement in HZSPRMxx for persistent data dataset

New system parameter HZS

- HZS=(aa,…,zz) or just HZS=xx in IEASYSxx
  - Identifies HZSPRMxx parmlib members to use at Health Checker start
- Works in conjunction with new special values for parameter HZSPRM of procedure HZSPROC
  - Allow to specify actual HZSPRMxx suffixes outside of HZSPROC
  - See next pages and also “Tell the system which HZSPRMxx members you want to use” in the Health Checker User's Guide

New special values for HZSPROC parameter HZSPRM

- HZSPRM=SYSPARM
  - Health Checker uses HZSPRMxx as identified by system parameter HZS

- **HZSPRM=PREV**
  - On first start: Same as SYSPARM
  - On restarts (after applying service…):
    - Uses suffixes in effect when previous instance was stopped
  - Is new HZSPRM default in shipped, updated HZSPROC

Procedure parameter HZSPRM – Details

- Old syntax HZSPRM=(aa,…,zz) still valid
  - System will ignore system parameter HZS
  - Old default was HZSPRM=00
    - Remember to add 00 to HZS as needed when switching

- HZSPRM=NONE, for completeness
  - System will ignore system parameter HZS and use no suffixes at all
  - Distinguish from HZSPROC=*NONE – unrelated, but looks similar

HZSPROC independence – HZSPDATA

- Persistent data dataset can now be specified outside of HZSPROC, in HZSPRMxx:
  - **HZSPDATA=*datasetname*[,VOLUME=volser]**
- Or, “activate” in newly shipped HZSPROC
  - commented out to allow “auto”-start w/o customization on new systems
    - But system will nag with “HZS0013A – Specify… HZSPDATA” while allowing HZSPROC to start

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

Also new in V2R1 – “HZSPRINT PARM>100”

- HZSPRINT is the tool to write check message (-buffer) content to a dataset and filter the output by certain criteria
- z/OS V1R10 introduced multiple new filter parameters
  - in particular TIMERANGE
  - total theoretical parameter length grew to ~180 characters
- Can not specify more than 100 characters via JCL PARM!
- See also “Using the HZSPRINT utility” in the Health Checker User’s Guide

HZSPRINT now exploits PARMDD

- PARMDD is mutually exclusive alternative to PARM
  - Allows to pass (long…) parameter string via DD
  - HZSPRINT is just one exploiter of this general new function

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

HZSPRINT PARMDD details

- HZSPRINT
  - Allows up to 256 characters input (> 180 !)
    - Trailing blanks per line do not count
    - PARMDD general max is ~32K
- Do not use leading blanks or anything but comma (‘,’) between parameters

No HZSPRINT PARMDD rollback

- PARMDD in general is a z/OS V2R1-only feature
- To still max out the 100 PARM characters in pre-V2R1:
  - Use wildcards in the check name filter parameter
    - For example use (*,PFA_E*), instead of (IBMPFA,PFA_ENQUEUE_REQUEST_RATE)
  - Continue long PARM in “packed” form, no spaces etc.:

```
//HZSPRINT JOB
// SET PARM1='CHECK(IBMASM,ASM_LOCAL_SLOT_USAGE)'
// SET PARM2=',LOGSTREAM(HZS.HEALTH.CHECKER.LOG)'
// SET PARM3=',EXCEPTIONS'
//HZSPRINT EXEC PGM=HZSPRNT,TIME=1440,REGION=0M,
//    PARM='&PARM1.&PARM2.&PARM3.'
//SYSOUT   DD SYSOUT=A,DCB=(LRECL=256)
```

Also new in V2R1 – More info in Message Buffer header

- System & sysplex name in each check's message buffer
- Easier to associate check output with system the check ran on

```
CHECK(IBMCATALOG,CATALOG_RNLS)
SYSPLEX:     PLEX1     SYSTEM: SY39
START TIME: 02/19/2013 12:16:40.224036
CHECK DATE: 20120827  CHECK SEVERITY: LOW

* Low Severity Exception *
IGGHC111E …
```

Message Buffer header – continued

- For example, when viewing health checks via SDSF CK panel when the SDSF multi-system support is enabled.

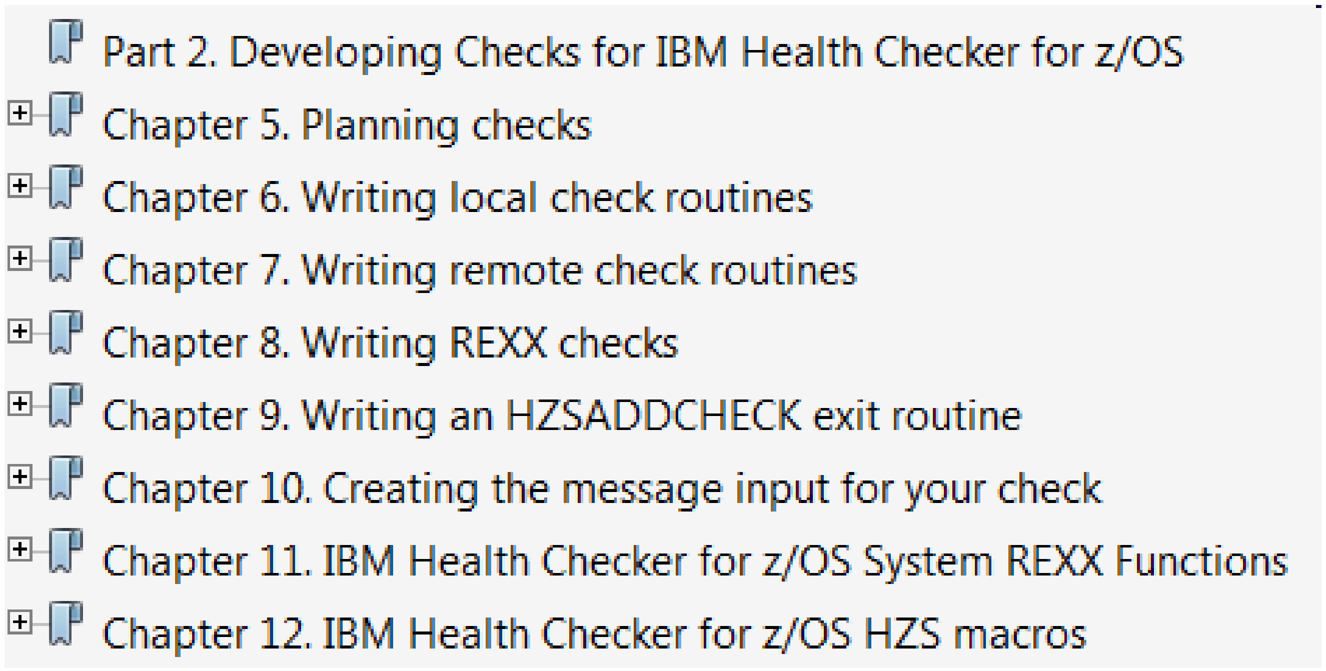
- Note: SDSF display contains these name as well
  - But only on main panel and by default far to the right

Part 2 – Check writing

- Session 14298 showed minimal, but functional, example of writing a health check
- In the following we will describe additional, more advanced considerations when it comes to writing your own health checks

Plenty of references

- Besides the details on the following pages there are multiple chapters on this in the Health Checker User's Guide:



Same basic concepts apply

- You provide the “inspection” code
  - The “check routine”

- You tell Health Checker where to find it
  - “ADD CHECK”

- Health Checker takes care of the rest
  - Runs check on schedule
  - Externalizes check messages

Check routine…

- Every health check needs a check routine
  - Code that checks single setting
    - or set of closely related settings
  - …and reports findings via standard protocol
    - Message service HZSFMSG (HZS**L**FMSG for REXX)

- Each check has its own unique messages
  - At least two: One to report exceptions and one for success
  - Often also “report” messages to build “tables”
    - Illustrate exception conditions and list affected objects etc.

… and message table

- Messages can be “embedded” in the check routine code
  - DIRECTMSG feature, available since z/OS V1R12
- Most checks use separate “Message Table”
  - Separation of code & (message) data is always good
    - Allows for easy translation, …
  - Check routine accesses messages by “xref” reference
- Message table must be provided as separate binary
  - Generated from SGML text message definitions
  - … via REXX exec HZSMSGEN

Message table SGML snippet

```
<msg class=exception>
<msgnum xreftext=001>HZSH0011E</msgnum>
<msgtext>
There are <mv class=hex xreftext="maxlen(20)">n</mv>
 (decimal <mv class=decimal>n</mv>) remaining &hzsnl;
available special &samptxt;. This is below the minimum.
</msgtext>
<msgitem class="EXPLANATION">
<p>The current number of available &samptxt;…
```

Message table SGML – continued

- <msg class=exception> requires multiple <msgitem>'s of class…:
  - "explanation"
  - "sysact" (system action)
  - "oresp" (operator response)
  - "sresp" (system programmer response)
  - "probd" (problem determination"
  - "source" (product/component… name)
  - … "automation", "module", "rcode", "dcode"

- Other <msg> classes:
  - "information", "report", and "debug"
  - No required <msgitem>'s for these

Message table SGML – continued

- Message variables (“inserts”, <mv>) allowed
  - Actual values provided by check routine at run time
- Health Checker provides predefined symbols
  - With, sometimes dynamic, text:
    - &hzs; &hzsproc; &hzssysname; &hzssysplex; &hzsreason; &hzsexitrtn; &hzsparmsource; &hzssev; &hzsparms; &hzsckname; &hzsowner; &hzsdate; &hzsgmttime; &hzslocaltime;
  - As text formatting help:
    - &rbl; &gt; &lt; &amp; &hzsnl; &hzsbl;
- Health Checker User’s Guide has all the details

Registered together

- To “ADD” a check to Health Checker, both check routine and (optional) message table are registered together

```
ADD CHECK(MYPROD,PROD_SAMPLE_CHECK)
    CHECKROUTINE(PRDCKRTN)
    MESSAGETABLE(PRDMSGTB)
     ...
```

- When just using DIRECTMSG

```
ADD CHECK(MYPROD,PROD_SAMPLE_CHECK)
    CHECKROUTINE(PRDCKRTN)
    MESSAGETABLE(*NONE)
     ...
```

- More on adding checks later…

Check routine implementation languages

- High Level Assembler (HLASM)
  - First available check implementation language
  - Best performance, finest control
  - “hardest” language, unless Assembler/MVS “expert”

- PLX (IBM only)
  - “real” high-level language, less error prone
  - Good performance

Implementation languages – continued

- (METAL-) C
  - “real” high level language, less error prone
  - Good performance, can be tweaked further (prolog/epilog)
  - Health Checker services accessed via embedded Assembler
    - But HC data structures fully mapped in C headers

- System REXX
  - “Easiest” language
  - “Simple” protocol with Health Checker
  - Performance can be optimized by compiling the REXX exec

Implementation languages – continued

- Otherwise, basically any language that can access Health Checker services (possibly via wrappers)
  - No special "official" support though
  - Need to provide own HC data structure mappings

Transparent to end-user

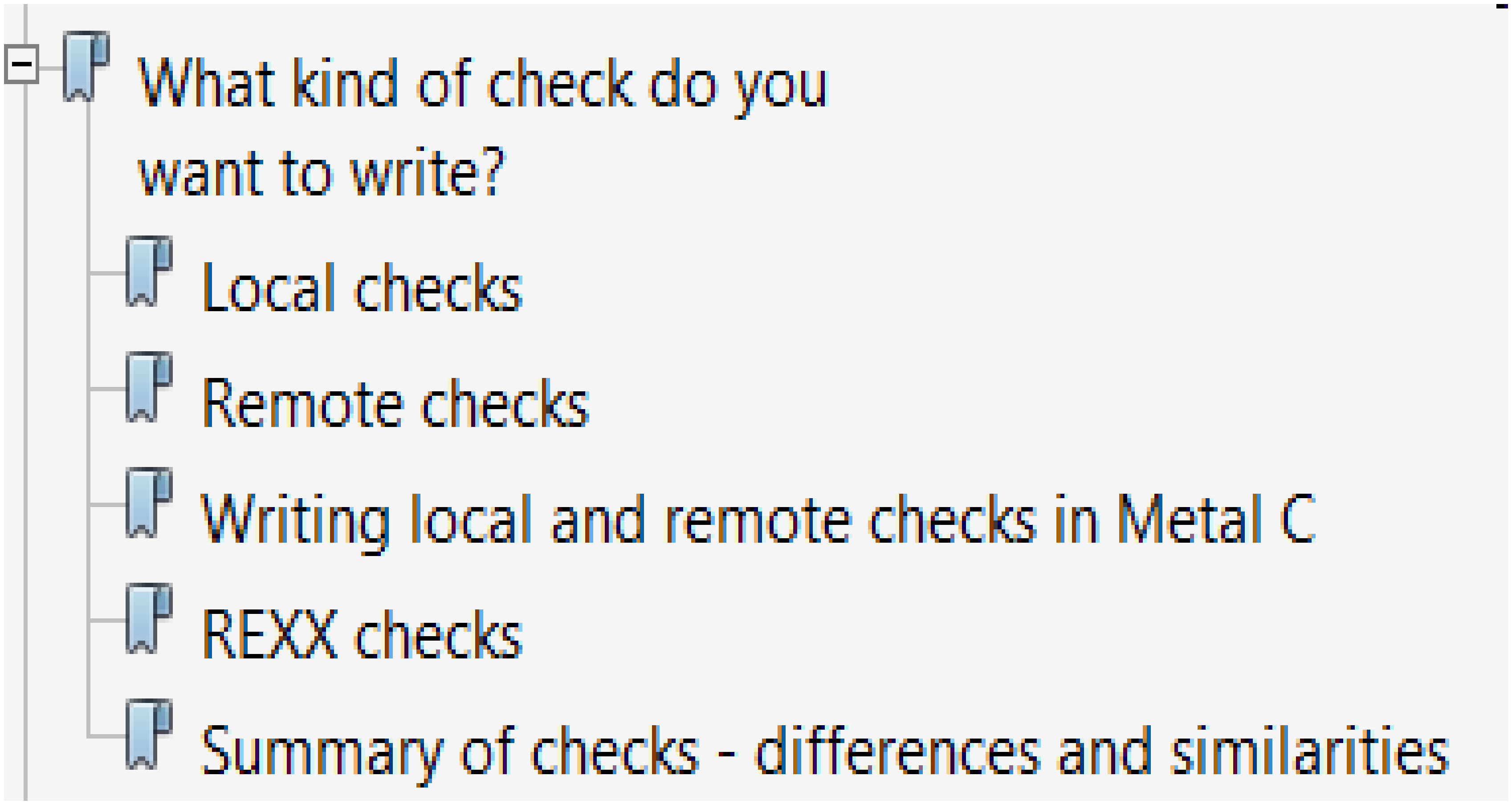
- Regardless of chosen check implementation language
  - No apparent difference for end-user
  - Follow the same “protocol”
  - Use the same services
    - Some services provided via wrappers for convenience (REXX)

Check writers need to distinguish though

- Have to pick appropriate check “locale”
  - “Local” checks
  - “Remote” checks
  - “REXX” checks

- Locales differ in terms of
  - How and where the check routine is provided
  - How check routine is executed
  - How long it takes and what resources are accessed to determine check result

More User's Guide references



- What kind of check do you want to write?
  - Local checks
  - Remote checks
  - Writing local and remote checks in Metal C
  - REXX checks
  - Summary of checks - differences and similarities

Local checks

- Checks are “local” to the Health Checker address space
- Check routine and check message table are loaded into private storage of the Health Checker address space
- Health Checker executes check routine from there and handles recovery etc.
- Good for short running health checks, without extra requirements, like accessing an address space’s private data or having the potential for “wait”
- Runs authorized

Sample local check routine outline

- Receives control with check control block (PQE) reference provided
- Inspects check parameter
  - See `PQE_LOOKATPARMS` and `PQE_PARMAREA`
- Inspects what to do: `PQE_FUNCTIONCODE`
  - `Pqe_Function_Code_`**`Init/Check/Cleanup/Delete`**
- Does the actual “checking“ (inspection of “setting”)
  - For `Pqe_Function_Code_`**`Check`**
  - Other function codes are often optional
  - Reports findings via message service HZSFMSG

Remote checks

- Run check routine in dedicated task of remote address space, “remote” from Health Checker address space
- Can run longer than local check and with ENQs, Waits, IO…
- Allows easy access to product/address space specific data
- Can run authorized or unauthorized
  - When unauthorized, needs additional RACF permissions for some HZS services
- Handshake protocol with Health Checker via
  - Pause Element Token (PET) services
  - Check handle, once task is released after PET PAUSE
- Provides its own recovery

Rough remote check routine outline

More on this later

- Get PAUSE element token (PET) via IEAVAPE service
- Load message table
- Loop: Issue RC=HSZADDCK(PET,…) to add me as a remote check
  - If RC = 'HC not active'
    - IF I'm_running_authorized THEN Wait for HC active ENF 67
    - ELSE /* Unauthorized */ Wait TIME=…
- Until (RC <> 'HC not active')
- Loop: PAUSE via IEAVPSE(PET,…) until Health Checker releases PET
  - Inspect check parameters
  - SWITCH (Function_Code)
    - CASE(Run) HZSCHECK REQUEST(OPSTART) - Establish hand-shake with HC
      CALL InspectionAndReportingCode;
      HZSCHECK REQUEST(OPCOMPLETE) - Finish hand-shake
    - CASE(InitRun) … for first check run – chance to init, too…
    - CASE(Deactivate) …
    - CASE(Delete) …
    - CASE(Delete_Term) …
    - CASE(Delete_Refresh) …
    - CASE(Restart) …

REXX checks

- Check routine is provided as System REXX exec in a System REXX library
- A special type of a remote check, running "remotely" in a System REXX worker task or address space
- Runs authorized
- Can use TSO services

REXX check routine outline

- `RC = HZSLSTRT()`
  - Establish hand-shake with Health Checker
- `IF HZS_PQE_LOOKATPARMS = 1 THEN…`
  - Parse check parameter

-
  ```
  IF HZS_PQE_FUNCTION_CODE = "INITRUN" | ,
     HZS_PQE_FUNCTION_CODE = "RUN" …
  ```
  - Inspect setting
  - Report findings via `HZSLFMSG`
- `RC = HZSLSTOP()`
  - Finish hand-shake with Health Checker

“ADDing” checks

- Multiple ways to register a check with Health Checker
  - ADD CHECK statement in HZSPRMxx parmlib member
    - Easiest
  - HZSADDCHECK dynamic exit routine
    - Most common
  - HZSADDCK service
    - Only option for remote checks
  - MODIFY HZSPROC,ADD,CHECK command
    - Rarely used

Many interfaces, one aim

- Identify the check, providing values such as
  - check owner,
  - check name,
  - check routine name,
  - message table name.
- Specify default values for the check, such as
  - check interval,
  - check parameter,
  - check severity.

ADD CHECK via HZSPRMxx

- Write once and reuse on system / Health Checker restarts
- Specify HZSPRMxx parmlib member suffix(es) via
  - System parameter HZS in IEASYSxx, or
  - Parameter HZSPRM of started procedure HZSPROC, or
  - “Manual” system command
    - MODIFY HZSPROC,ADD,PARMLIB=xx

ADD CHECK via HZSPRMxx – Example

- Some parameters depend on check locale
  - (applies to the other ADD interfaces, too)
  - For examples REXX vs. non-REXX:

```
ADD
CHECK(MYPROD,PROD_LIMIT_CHECK)
    EXEC(PRDLIMCK)
    REXXHLQ(IBMUSER)
    MSGTBL(*NONE)
    PARMS('LIMIT(47)')
    SEVERITY(MEDIUM)
    INTERVAL(0:30)
    DATE(20130630)
    REASON('Check PROD LIMIT')
```

```
ADD
CHECK(MYPROD,PROD_LIMIT_CHECK)
    CHECKROUTINE(PRDLIMCK)

    MSGTBL(*NONE)
    PARMS('LIMIT(47)')
    SEVERITY(MEDIUM)
    INTERVAL(0:30)
    DATE(20130630)
    REASON('Check PROD LIMIT')
```

ADD CHECK via HZSADDCHECK exit

- System comes with dynamic exit HZSADDCHECK
  - Available independent of Health Checker status
- Products can register exit routines with exit
  - Callable service: CSVDYNEX REQUEST=ADD
  - System Command: SETPROG EXIT,ADD (for test)
- Exit routine contains HZSADDCK service call(s)
- Health Checker calls exit for exit routines to be executed
  - At startup or upon request
    - Callable service: HZSCHECK REQUEST=ADDNEW
    - System command: MODIFY HZSPROC,ADDNEW (for test)

ADD CHECK via HZSADDCHECK exit – continued

- So a product would for example
  - Ship exit routine along with check routine and message table
  - Register the exit routine once at product startup
    - CSVDYNEX REQUEST=ADD
  - …and notify Health Checker
    - HZSCHECK REQUEST=ADDNEW
    - Noop if Health Checker is down: Next (re-)start will trigger exit

- If the product does not stay up for the life of the system and health checks depend on product
  - Explicitly delete health checks on product shutdown
  - Code exit routine to only conditionally execute HZSADDCK depending on current product status (started/stopped)
    - Safer than deleting the exit routine from the exit

HZSADDCHECK exit routine samples

```
HZSSADCK CSECT
…
 HZSADDCK CHECKOWNER=CK_Owner,  Owner’s Name of the check *
  CHECKNAME=CK1_Name,      ... the Name of the Check      *
  CHECKROUTINE=CK_Routine, .. Name of the module with chk *
  ACTIVE,                  ... Activate check immediately  *
  LOCAL,                   ... Run check on every system   *
  ENTRYCODE=CK1_Entry_Code, . Entry code for this check    *
  EXITRTN=CK_Xit_Routine,  ... Name of routine adding chk  *
  MSGTBL=CK_MSG_Table,     ... Name of message table       *
  DATE=CK1_Date,           ... Base date for defaults      *
  REASON=CK1_Reason,       ... Reason for this check       *
  REASONLEN=CK1_Rsn_Len,   ... ... and its length          *
  PARMS=Ck1_Parm,          ... Default Parameters          *
  PARMSLEN=Ck1_Parm_Len,   ... ... and their length        *
  SEVERITY=LOW,            ... Severity of check           *
  INTERVAL=ONETIME,        ... A one time check            *
  USS=NO,                  ... Check does not use USS      *
  MF=(E,HZSADDCK_PLIST)    ... Execute form
```

- See also
  - SYS1.SAMPLIB(HZSSADCK)
  - /usr/lpp/bcp/samples/hzscadd.c

ADD CHECK for remote checks

- Can not use HZSPRMxx or HZSADDCHECK exit
  - Needs to exchange “live” information with Health Checker
    - “Handshake” token – PET (Pause Element Token)
    - Check handle for service calls
    - Environment info (implicit, like task handle…)
- Uses HZSADDCK service
  - Just like in HZSADDCHECK exit routine, but “live”

ADD CHECK for remote checks – continued

- So, remote check routines contain extra section at top

```
* Obtain a pause element                                                *
           CALL   IEAVAPE,(IEAVAPE_ReturnCode,                           -
                  theAuthLevel,            ... Has to be IEA_UNAUTHORIZED -
                  thePauseElementToken),                                 -
                  MF=(E,CallList)
...
* Add the check                                                         *
           HZSADDCK CHECKOWNER=CK_Owner, ... the Owner of the check      -
                  CHECKNAME=CK_Name,       ... the Name of the Check      -
                  REMOTE=YES,              ... This is a remote check     -
                  HANDLE=CK_Handle,        ... The handle for the check   -
                  PETOKEN=thePauseElementToken,                          -
...
```

ADD CHECK via command

- Rarely used and if at all, for simple testing only
  - Command line is too short
  - Not a permanent solution
  - Exists only “for completeness” / HZSPRMxx code sharing

Once ADDed, Health Checker takes over

- Takes care of check-run scheduling
  - Based on INTERVAL as specified on ADD
    - Can be modified via UPDATEs
  - First run by default right after ADD
    - SYNCVAL (V1R13) allows more precise control
      - *When to run first; also more predictable consecutive runs*

Health Checker takes care of rest – continued

- Is central processor of check messages
  - And provides service interface for retrieval (HZSQUERY)
    - Used by F HZSPROC,DISPLAY
    - Used by SDSF CK panel and similar “clients”
- Is central processor for check maintenance
  - Process Delete, Update, … requests for checks
    - Via MODIFY HZSPROC commands & HZSCHECK service

References

- SHARE Boston 2013 – Session 14298
  - “IBM Health Checker for z/OS – Intro and next steps”
- “IBM Health Checker for z/OS User's Guide” (SC23-6843)
  - Guide and Reference
  - Includes an inventory of IBM supplied health checks
- “Exploiting the Health Checker for z/OS infrastructure”
  - Health Checker “hands-on” Redpaper 4590
- “V2.1 z/OS Migration” book
  - “Convert your existing IBM Health Checker for z/OS set-up for automatic start-up” section

- Health Checker framework contact and to direct questions about individual health checks:
  - Ulrich Thiemann (thiemanu@us.ibm.com)