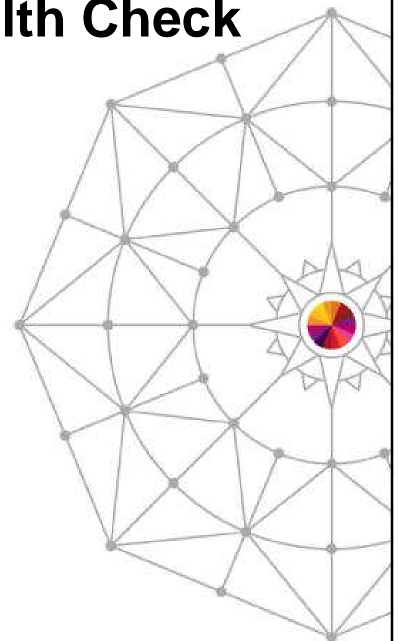
Write a Dynamic Severity Health Check and Avoid Assembler



Ulrich Thiemann
IBM
(presented by Marna Walle)

March 10th, 2014
Session 15291

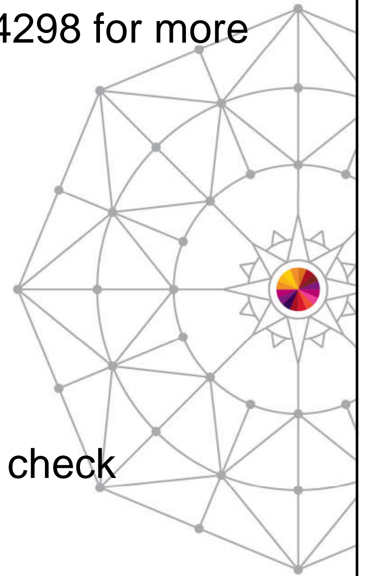
Copyright (c) 2014 by SHARE Inc. Except where otherwise noted, this work is licensed under http://creativecommons.org/licenses/by-nc-sa/3.0/

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.

Agenda

- **Part 1:** Brief introduction (see recent session #14298 for more details)
  - Health Checker and health checks
  - Check exceptions
  - Check exception severity
  - Dynamic severity
  - Check implementation languages
- **Part 2:** Walkthrough for a fully functional sample check
  - for a “resource X”,
  - using dynamic severity,
  - written in METAL C.



Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Health Checker in z/OS

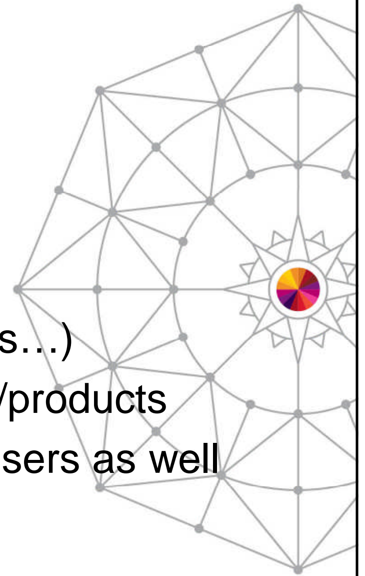
- A component of MVS that identifies potential problems before they impact your system's availability or, in worst cases, cause outages

- Inspects active z/OS and sysplex settings and definitions
  - for deviations from best practices
  - for getting close to critical thresholds
  - for recommended and required migration actions

- Informs the system programmer via detailed messages

Health Checker vs. health checks

- One "Health Checker" framework
  - backed by system address space, HZSPROC
- Many health checks
  - Framework "plug-ins"
  - Do the actual "checking" (inspection of settings…)
  - Owned by separate/independent components/products
  - Not just from IBM (~200), but from ISVs and users as well



 Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Check exceptions

- System programmer gets alerted via check exceptions
  - Summary text via WTO on console

```
*SY40 *HZS0003E CHECK(IBMXCF,XCF_CDS_SPOF):
*IXCH0242E One or more couple data sets have a single point of failure.
```

  - Details in check's message buffer

```
…
IOSPF252I Volumes CPLPKP (0485) and CPLPKA (0487) share the
same physical control unit.
…
* High Severity Exception *

IXCH0242E One or more couple data sets have a single point of failure.

  Explanation:  The couple data set configuration has one or more single
    points of failure.  A failure at one of these points could result in
    loss of a couple data set, system, or even the entire sysplex.
…
```


Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Check exception severity

- Each exception has severity associated with it
  - HIGH, MEDIUM, LOW… based on urgency to fix
  - Default severity defined at ADD CHECK time
- Different severity, different exception message ID
  - HZS0003E – HIGH
  - HZS0002E – MEDIUM
  - HZS0001I – LOW
- Multiline, with automation friendly generic message first

```
*SY40 *HZS0003E CHECK(IBMXCF,XCF_CDS_SPOF):
*IXCH0242E One or more couple data sets have a single point of failure.
```


Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Dynamic severity

- Allows check routine to adjust severity at run time
  - Available in z/OS V1R13 and higher
- For "dynamic" settings, like thresholds, limits, …
  - vs. "fixed", on/off, … type settings
- Check typically supports check parameters to associate different thresholds with different severities

```
CHECK(IBMVSM,VSM_CSA_THRESHOLD)
PARM('CSA_HIGH(95%),CSA_MED(80%),CSA_LOW(60%),ECSA_HIGH(90%)
   ,ECSA_MED(70%)')
```


Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Dynamic severity…

- Needs to be explicitly enabled and exploited…
  - On ADD CHECK via ALLOWDYNSEV(YES)
  - In check routine, via new SEVERITY parameter of message service HZSFMSG:

```
HZSFMSG REQUEST={CHECKMSG|DIRECTMSG} ...

       |-- SEVERITY = SYSTEM
...--|----------------------------------------------
       |-- SEVERITY = NONE
       |-- SEVERITY = LOW
       |-- SEVERITY = MED
       |-- SEVERITY = HI
       |-- SEVERITY = VALUE , SEVERITYVAL = severityval
```

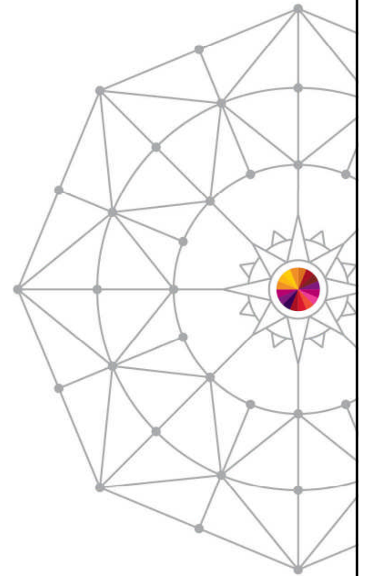
For pre-R13 compatibility and ALLOWDYNSEV(NO)

Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Dynamic severity…

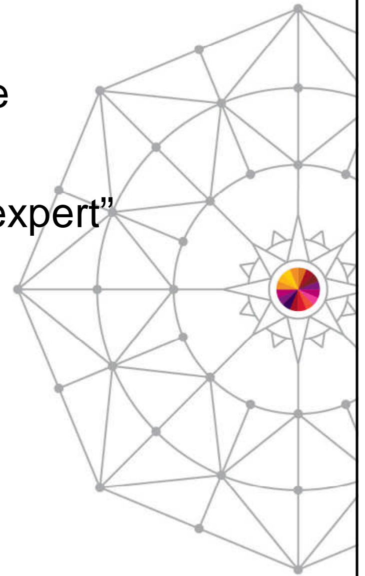
- More IBM health checks are exploiting it
  - For example:
    (IBMASM,ASM_PLPA_COMMON_SIZE)
    (IBMASM,ASM_PLPA_COMMON_USAGE)
    (IBMASM,ASM_LOCAL_SLOT_USAGE)
    (IBMVSM,VSM_CSA_THRESHOLD)
    (IBMVSM,VSM_SQA_THRESHOLD)
    …now optionally support dynamic severity



Check 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

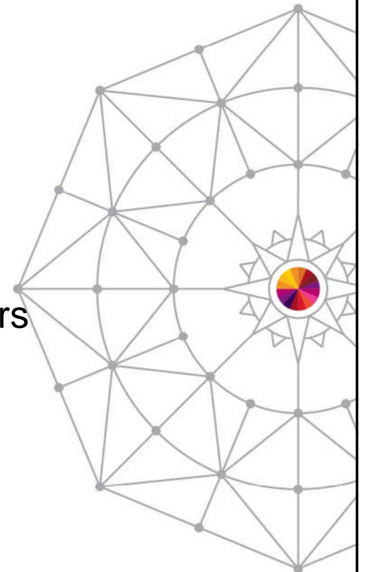


Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

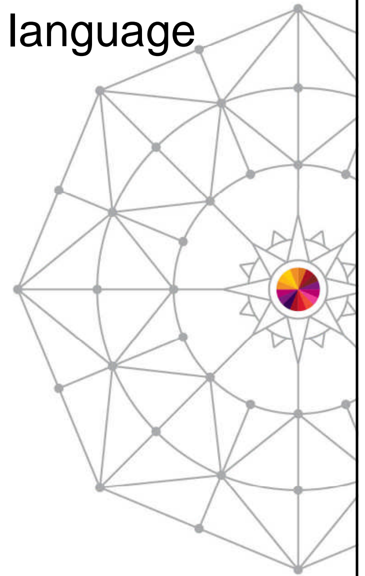
Implementation languages…

- (METAL-) C
  - “real” high level language, less error prone
  - Good performance
  - Health Checker services accessed via embedded Assembler (__asm)
    - 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…

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



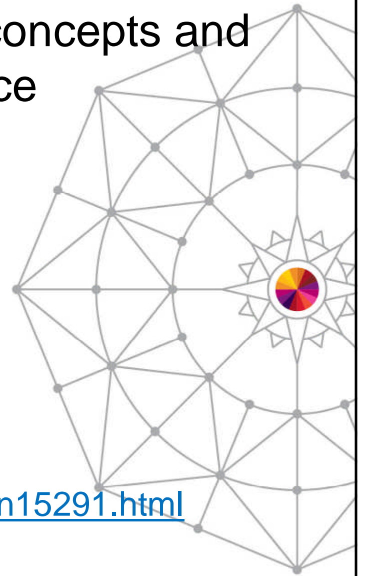
SHARE in Anaheim

Part 2: Sample health check

- Will visit and reiterate general health check concepts and dynamic severity while walking through source

- Full source code available on SHARE site
  - Check routine source
  - JCL script to build check routine
  - HZSPRMxx parmlib member to ADD

  https://share.confex.com/share/122/webprogram/Session15291.html



 Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

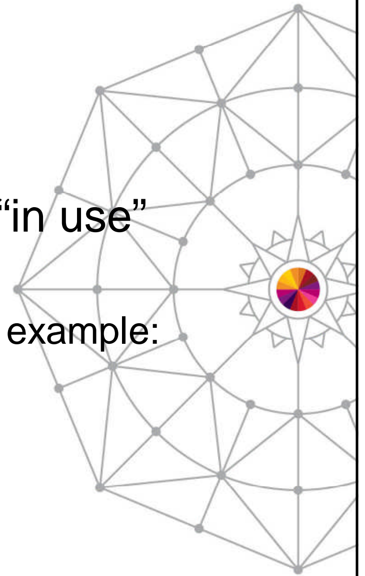
SHARE in Anaheim

Check purpose

- Check is for single system resource (“X”)
  - For example a certain, “rare” memory type

- Check warns about near “exhaustion” / high “in use”
  - With increasing urgency / severity
    - Customizable via THRESHOLD parameters, for example:

```
PARM('THRESHOLD_HIGH(85%)'
      'THRESHOLD_MED(75%)'
      'THRESHOLD_LOW(60%)')
```



SHARE in Anaheim

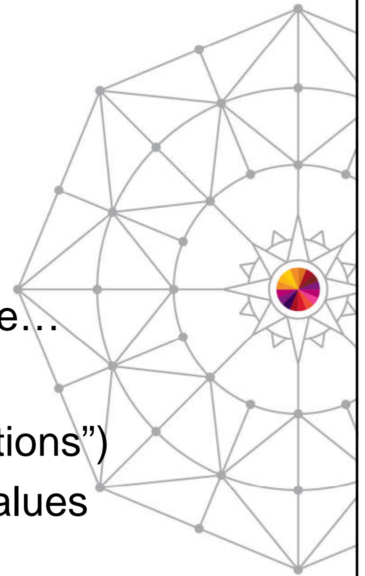
Mainline

```
int main(HZSPQE* pPQE)
{
 tSeverityThresholds *pSevThresholds =
         (tSeverityThresholds *) &(pPQE->ChkWork);

 switch(pPQE->Function_Code)
 {
      case HZSPQE_Function_Code_Init:
         SetDefaultSevThresholds(pSevThresholds);
      break;

      case HZSPQE_Function_Code_Check:
      {
        if (pPQE->LookAtParms)
        {
            if (RC_OK != ParseParmString(pPQE->ParmLen
                                        ,pPQE->ParmArea
                                        ,pSevThresholds))
                goto endMain;
        }

        uint8_t curLevel = getCurrentResourceLevel(pPQE->Check_Count);

        CompareCurrentVsThresholdsAndReport(curLevel,pSevThresholds);
      }
      break;
 }

 endMain:
  /* chance to cleanup temporary resources... */

 return 0; /* Failures should be reported through HZSFMSG */
}
```

Mainline – HZSPQE

```
int main(HZSPQE* pPQE)
{
 tSeverityThresholds *pSevThresholds =
          (tSeverityThresholds *) &(pPQE->ChkWork);
```

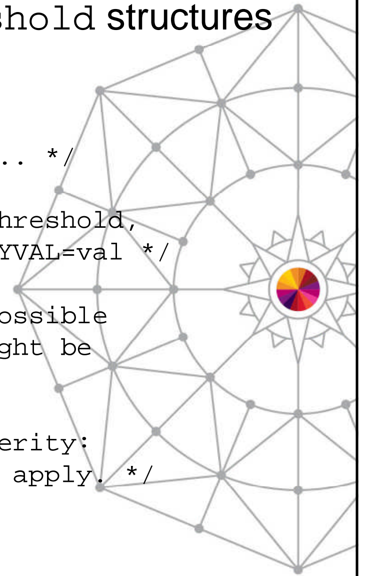
- HZSPQE is the “check control block”
  - Contains all kinds of check attributes and more…
- Includes a 2K work area
  - Preserved between check calls (“runs” / “iterations”)
  - Good spot to save current check parameter values



Mainline – Check parameters

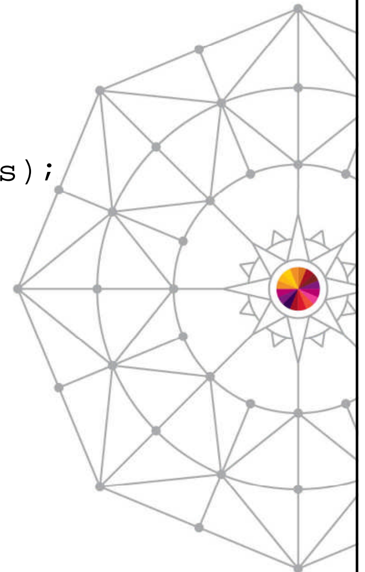
- Mapped as 4-element array of `tOneSeverityThreshold` structures

```
typedef struct
{
    /* const */ char parmName[16]; /* "THRESHOLD_HIGH  ", ... */

    /* const */ uint8_t severityVal; /* Severity for this threshold,
                         ready to use for HZSFMSG ... SEVERITYVAL=val */

    uint8_t isActive; /* We always keep track of all four possible
                         parameters, but not all of them might be
                         "active" / are actually used. */

    uint8_t minPercent; /* The threshold value for this severity:
                           Lower limit for this severity to apply. */

    uint8_t _align;
} tOneSeverityThreshold;

typedef tOneSeverityThreshold tSeverityThresholds[THRESHOLD_COUNT];
```



Mainline – Function code

```
switch(pPQE->Function_Code)
 {
      case HZSPQE_Function_Code_Init:
         SetDefaultSevThresholds(pSevThresholds);
      break;
```

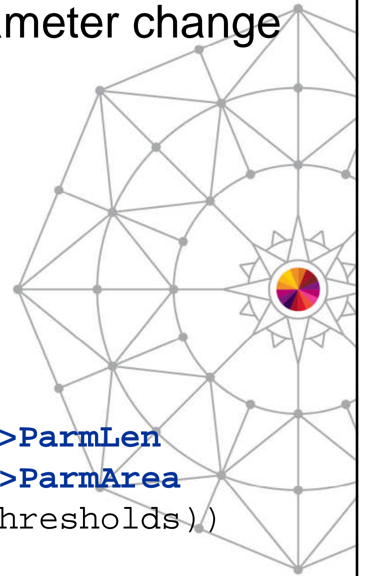


- Tells the check what to do when called by Health Checker framework
  - For example: Chance to initialize, once, before very first "real" check run

SHARE in Anaheim

Mainline – Function code…

- Main, "run" code also gets indication for parameter change / first time…

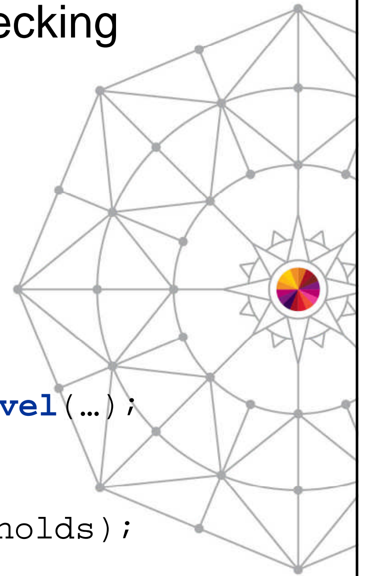


```
switch(pPQE->Function_Code)
{
 …
     case HZSPQE_Function_Code_Check:
     {
       if (pPQE->LookAtParms)
       {
           if (RC_OK != ParseParmString(pPQE->ParmLen
                                        ,pPQE->ParmArea
                                        ,pSevThresholds))
               goto endMain;
       }
```

Mainline – Function code…

- … and of course to actually “run” / do the checking

```
switch(pPQE->Function_Code)
{
  …
     case HZSPQE_Function_Code_Check:
     {
       if (pPQE->LookAtParms)
         …
       uint8_t curLevel = getCurrentResourceLevel(…);

       CompareCurrentVsThresholdsAndReport
                                   (curLevel,pSevThresholds);
     }
```



Complete your session evaluations online at www.SHARE.org/Anaheim-Eval
SHARE in Anaheim

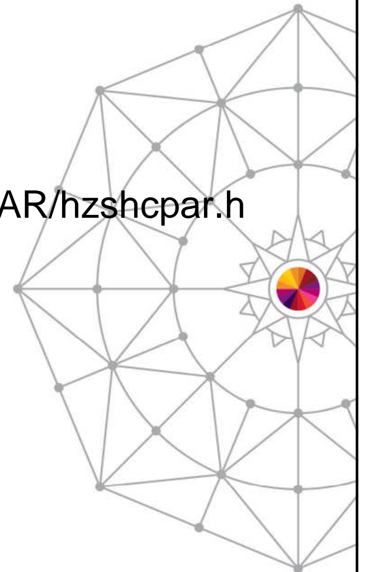
Details – Parameter Parsing

- Check gets reference to PARM string
  - As specified on ADD CHECK or UPDATE CHECK
  - Via HZSPQE fields (PQE_)ParmLen, (PQE_)ParmArea

- Health Checker offers parsing service HZSCPARS
  - Supports keyword+value or positional parameters
  - Supports parameter existence check
  - Supports parameter format check (numeric, percent,…)
  - Supports check for unsupported parameters

Details – Typical use of HZSCPARS

1. HZSCPARS REQUEST=PARSE
   - Tokenize the PARM string
   - Return reference to internal control block
     - Individual sub structures mapped by HZSZCPAR/hzshcpar.h



2. HZSCPARS REQUEST=CHECKPARM
   - Check existence of single parameter
   - Validate key+value vs. positional format
   - Validate number of allowed values
   - Return reference to first value control block

SHARE in Anaheim

Details – Typical use of HZSCPARS…

3. HZSCPARS REQUEST={CHECKDEC|CHECKHEX|CHECKCHAR}
   - For individual values of single parameters
   - Validate valid numeric value/text-length range and format
   - Returns reference to info area with binary numeric value…

4. HZSCPARS REQUEST=FREE
   - Release internal parsing control block from step 1.
     - After done with all parameters and values (steps 2./3.)

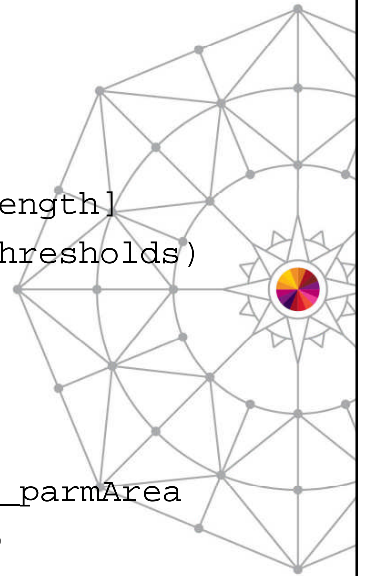
Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Details – Function ParseParmString

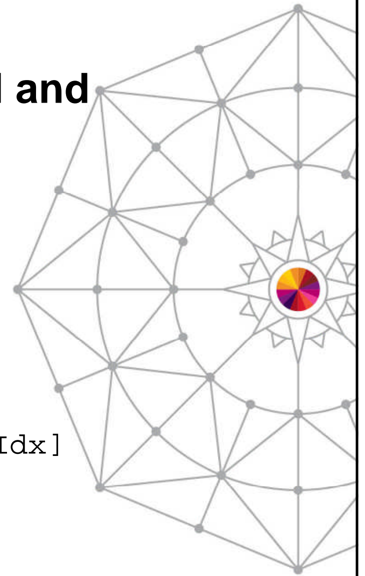
- **Start with HZSCPARS REQUEST=PARSE**

```
static int ParseParmString
                (int16_t in_parmLen
                ,char io_parmArea[HZSPQE_ParmLength]
                ,tSeverityThresholds *io_pSevThresholds)
{
 HzsCParArea* pCParArea = 0;

 /* Initialize parsing */
  switch(IssueHzsCPars_ParseInit(in_parmLen,io_parmArea
                                 ,&pCParArea))
  …<error handling>
```



Details – Function ParseParmString…

- Loop through the possible parameters
  - Uses HZSCPARS REQUEST=CHECKPARM and REQUEST=CHECKDEC under the covers

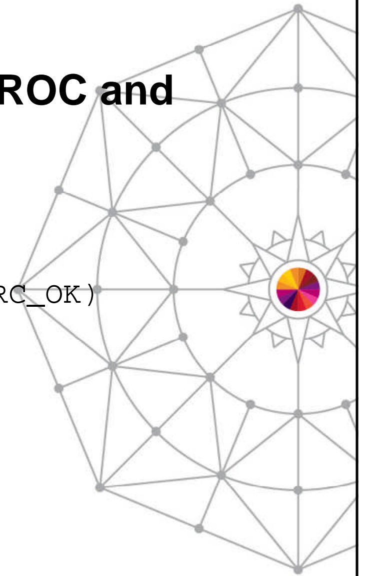


```
for(int thresholdIdx = THRESHOLDIDX_HIGH;
   thresholdIdx <= THRESHOLDIDX_NONE;
   ++thresholdIdx)
{
   switch(FindAndValidateOneThreshold
                 (&(*io_pSevThresholds)[thresholdIdx]
                 ,&prevPercentValue
                 ,pCParArea))
   …<error handling>
}
```

Details – Function ParseParmString…

- **Some cleanup and defaults…**
  - **Uses HZSCPARS REQUEST=CHECKNOTPROC and REQUEST=FREE**

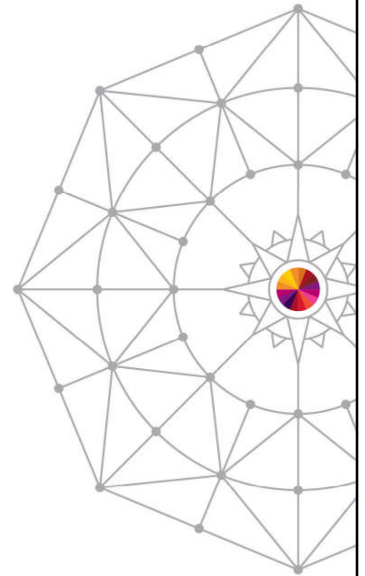
```
/* Any unsupported parameters specified? */
 if (IssueHzsCPars_CheckNotProcessed(pCParArea) != RC_OK)
      …<error handling>

if (!foundAtLeastOneParm)
      SetDefaultSevThresholds(io_pSevThresholds);

if (IssueHzsCPars_Free(&pCParArea) != RC_OK)
     …<error handling>
```



Low-level details

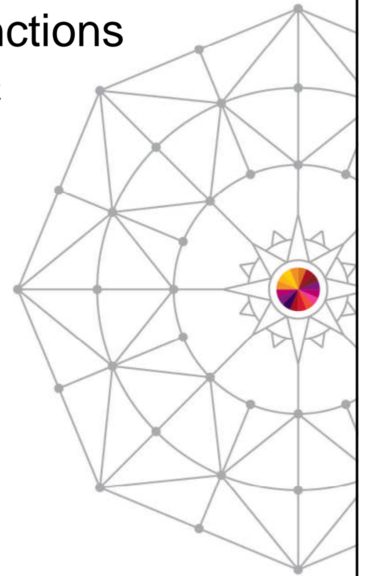
- Encapsulated in “simple” functions

```
static void SetDefaultSevThresholds
static int ParseParmString
static int FindAndValidateOneThreshold
static int IssueHzsCPars_ParseInit
static int IssueHzsCPars_CheckParm_CheckDec
static int IssueHzsCPars_CheckNotProcessed
static int IssueHzsCPars_Free
static uint8_t getCurrentResourceLevel
static void CompareCurrentVsThresholdsAndReport
static void ReportCheckSuccess
static void ReportCheckException
static void ReportParmValueTooLarge
static void IssueStopDueToBadParm
static void IssueStopDueToError
static void AbendWithRsn
```



Low-level details – Assembler use

- A prime example for wrapping in “simple” functions
  - Leaves “high-level” check logic in (METAL-) C
- For example: Use of service HZSCPARS



```
static int IssueHzsCPars_ParseInit
                (uint32_t in_parmLen
                ,char* in_parmArea
                ,HzsCParArea* *out_ppCParArea)
{
    int32_t hzsRC;
    int32_t hzsRsn;

    __asm(" HZSCPARS PLISTVER=MAX MF=L" : "DS"(plHZSCPARS));
```

Low-level details – Assembler use…

- HZSCPARS fills in the "real" parmlist for you
  - Like all HZS services

```
__asm(" HZSCPARS REQUEST=PARSE,"
        "PARM=(%5),"
        "PARMLEN=%4,"
        "TOUPPER=YES,"
        "CPARAREAADDR=%2,"
        "PARMFORMAT=KEYWORD,"
        "RETCODE=%1,"
        "RSNCODE=%0,"
        "MF=(E,(%3))"
      : "=m"(hzsRsn)
       ,"=m"(hzsRC)
       ,"=m"(*out_ppCParArea)
      : "r"(&plHZSCPARS)
       ,"m"(in_parmLen)
       ,"r"(in_parmArea)
      );
```

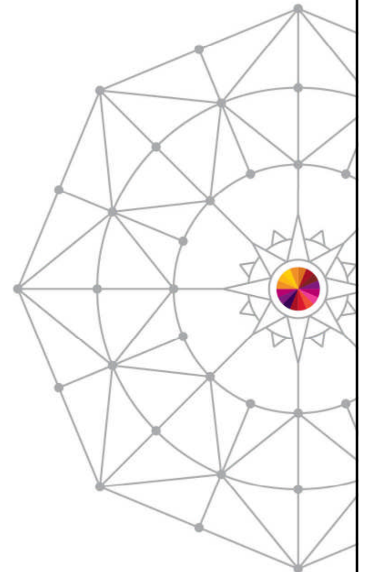
"register" form

"storage" form

"output"

"input"

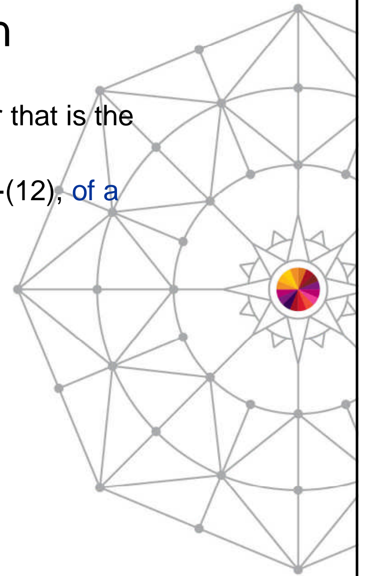
__asm parameters (start with %0)



SHARE Technology • Connections • Results

SHARE in Anaheim

Low-level details – HLASM to __asm



- From the HZSCPARS service documentation

  ,PARMLEN=*parmlen*
  When REQUEST=PARSE is specified, a required input parameter that is the length of the input parameter.
  To code: Specify the **RS-type address**, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

- __asm invocation:

```
uint32_t in_parmLen
...
__asm(" HZSCPARS REQUEST=PARSE,"
          "PARMLEN=%4,"
         : …
         : …
         ,"m"(in_parmLen)
```

Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

Low-level details – HLASM to __asm

- From the HZSCPARS service documentation

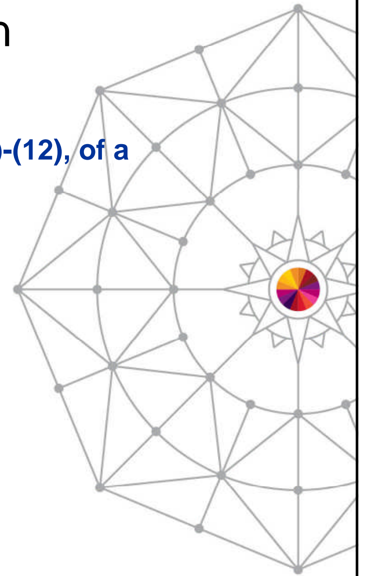
  ,PARM=*parm*

  …

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

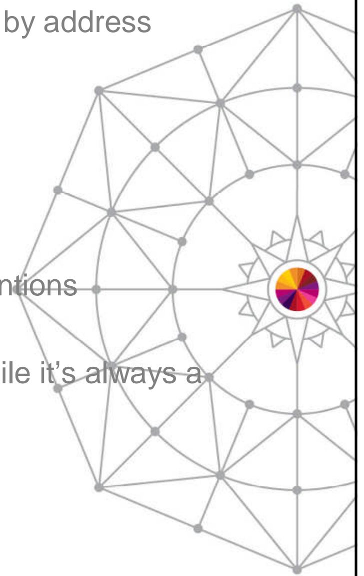
- __asm invocation:

```
char* in_parmArea
…
__asm(" HZSCPARS REQUEST=PARSE,"
          "PARM=(%5),"
         : …
         : …
         ,"r"(in_parmArea)
```



BACKUP – METAL C __asm Notes

- Mark service output parameters as output (“=“), even while passed by address
  - C compiler might “optimize” output away otherwise
  - Alternatively declare C variables used for output as “volatile”
- Be careful when using function parameters in __asm calls
  - OK for data types with size being a multiple of 4
  - uint16_t for example should be copied into uint32_t first
  - Otherwise offset might be off due MVS linkage / parmlist conventions
- Use char* instead of char[ ] in function parameter list
  - Char[ ] makes it tempting to use as “storage operand”, even while it’s always a pointer
- Qualify assembler literals with appropriate length / extension
  - For example =A for 4-byte address, =AD for 8-byte address
- ARMODE is tricky
  - Needs lots of manual setting of ALETs,
  - …including for R3 (literal pool) and R10 (static data)
  - Be "safe" and construct all __far pointers explicitly via __set_far_...



Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Low-level details – Assembler use…

- Error handling
  - C constants available for RC/RSN
  - Check practice: Use HZSFMSG to report and “disable”
- So, concluding our HZSCPARS REQUEST=PARSE example via C function `IssueHzsCPars_ParseInit`:

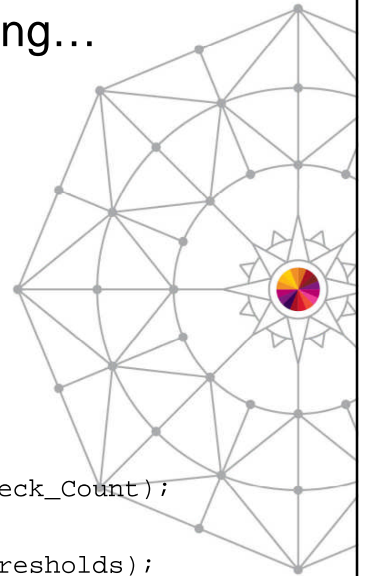
```
if (hzsRC != 0)
{
    if (hzsRsn == HzsCParsRsn_NoParms)
        return INITPARSE_OK_EMPTY;

    IssueStopDueToBadParm();
    return INITPARSE_ERROR;
}

return INITPARSE_OK;
```


SHARE in Anaheim

Back to `main`

- Now that we have “covered” parameter parsing…

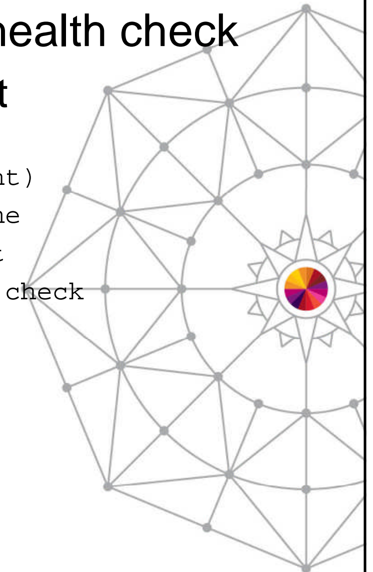
```
case HZSPQE_Function_Code_Check:
     {
       if (pPQE->LookAtParms)
       {
            if (RC_OK != ParseParmString(pPQE->ParmLen
                                        ,pPQE->ParmArea
                                        ,pSevThresholds))
                 goto endMain;
       }

       uint8_t curLevel = getCurrentResourceLevel(pPQE->Check_Count);

       CompareCurrentVsThresholdsAndReport(curLevel,pSevThresholds);
     }
     break;
```



Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

The core of your check

- Deals with the one “unique” feature of your health check
  - The setting/configuration option it is looking at



```
static uint8_t getCurrentResourceLevel(int32_t in_checkCount)
{    /* vary the "current", dummy resource level based on the
      * check iteration number, so that we can see different
      * check exception severities etc. when re-running the check
      */
    switch(in_checkCount%4)
    {
     case 1: return 90;
     case 2: return 80;
     case 3: return 70;
     default: return 50;
    }
}
```

 Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

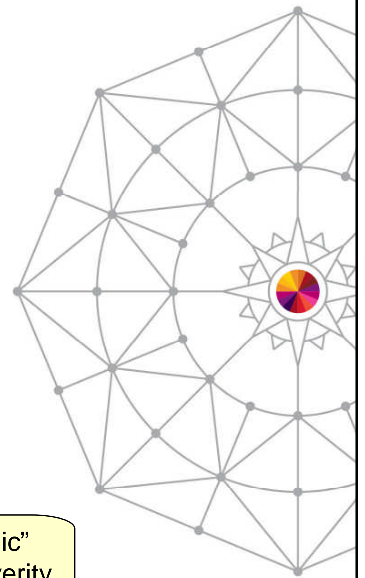
SHARE in Anaheim

More “mechanics”

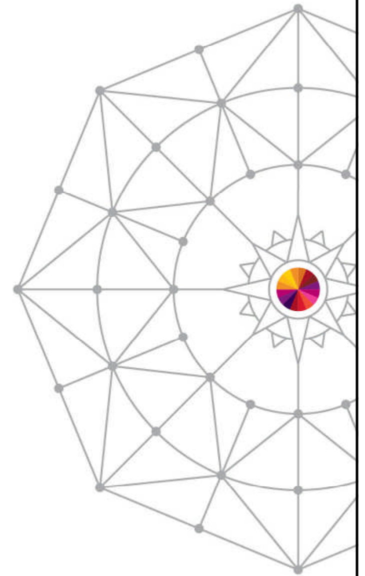
- A check always reports on its findings
  - for every check run (“iteration”)
  - via message service HZSFMSG
- “Success” / “All is well”
  - With brief description of what has been checked
- “Exception” message
  - With details on what was has been found “wrong”
  - With details on how to respond to it / fix it
  - With references to in-depth documentation

```
static void CompareCurrentVsThresholdsAndReport
                (uint8_t in_curPercentage
                ,tSeverityThresholds *in_pSevThresholds)
{
 int foundException = 0;

 for(int thresholdIdx = THRESHOLDIDX_HIGH;
         thresholdIdx <= THRESHOLDIDX_NONE;
         ++thresholdIdx)
 {
     tOneSeverityThreshold* pCurSevThreshold =
             &((*in_pSevThresholds)[thresholdIdx]);

     if (pCurSevThreshold->isActive &&
         in_curPercentage >= pCurSevThreshold->minPercent)
     {
         ReportCheckException(pCurSevThreshold->severityVal
                             ,in_curPercentage);
         foundException = 1;
         break;
     }
 }
 if (!foundException)
     ReportCheckSuccess(in_curPercentage);
}
```

Our “dynamic” exception severity



```
static const char SUCC_TEXT[] = "Resource X is good at %i%%.";

static void ReportCheckSuccess(uint8_t in_curPercentage)
{
    int32_t hzsRC;
    int32_t hzsRsn;
    uint16_t succLen;
    char succBuffer[sizeof(SUCC_TEXT)+10];

    snprintf(succBuffer,sizeof(succBuffer),SUCC_TEXT,in_curPercentage);

    succLen = strlen(succBuffer);

    __asm(" HZSFMSG PLISTVER=MAX MF=L" : "DS"(plHZSFMSG));
    __asm(" HZSFMSG REQUEST=DIRECTMSG,"
               "REASON=CHECKINFO,"
               "ID==CL8'SHRH002I',"
               "IDLEN=8,"
               "TEXT=%4,"
               "TEXTLEN=%3,"
               "RETCODE=%1,"
               "RSNCODE=%0,"
               "MF=(E,(%2),COMPLETE)"
          : "=m"(hzsRsn)
           ,"=m"(hzsRC)
          : "r"(&plHZSFMSG)
          , "m"(succLen)
          , "m"(succBuffer)
          );
    if (hzsRC) IssueStopDueToError(hzsRsn);
}
```



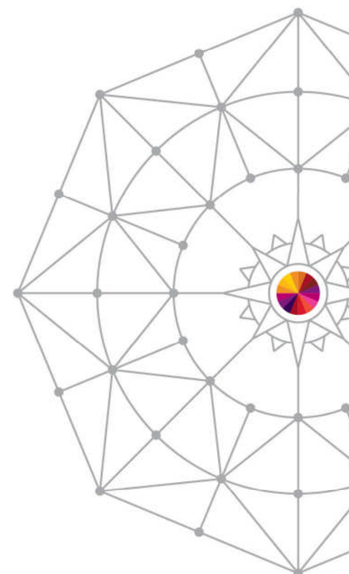
A word on DIRECTMSG and exceptions

- The exception message routine **ReportCheckException** is more involved.
  - Needs to pass more information for that one HZSFMSG call.
- The sample still uses DIRECTMSG
  - Embeds message text directly into check routine source code.
  - Allows “stand-alone” delivery of check
- DIRECTMSG mainly intended for REXX checks though.
- Other checks should (eventually) use a message table
  - Encapsulates message data.
  - Check routine just references via <msgnum> value.

Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

```
static void ReportCheckException(uint8_t in_severityVal
                                ,uint8_t in_curPercentage)
{
…
 fixedLenInput.SeverityVal = in_severityVal;
…
__asm(" HZSFMSG REQUEST=DIRECTMSG,"
                   "REASON=CHECKEXCEPTION,"
                   "SEVERITY=VALUE,"
                   "SEVERITYVAL=16(%1),"
                   "ID==CL8'SHRH001E',"
                   "IDLEN=8,"
                   "TEXT=(%2),"
                   "TEXTLEN=8(%1),"
                   "EXPL=(%3),"
                   "EXPLLEN=10(%1),"
                   "SPRESP=(%4),"
                   "SPRESPLEN=12(%1),"
                   "REFDOC=(%5),"
                   "REFDOCLEN=14(%1),"
                   "RETCODE=0(%1),"
                   "RSNCODE=4(%1),"
                   "MF=(E,(%0),COMPLETE)"
             :
             : "r"(&plHZSFMSG)
             , "r"(&fixedLenInput)
             , "r"(&EXCP_SUMMARY)
             , "r"(&ExplBuffer)
             , "r"(&EXCP_SPRESP)
             , "r"(&EXCP_REFDOC)
             );
```



…and we did not even use the remaining SYSACT, ORESP, PROBD, SOURCE, AUTOMATION parameters and their LEN values…

Build it

- Download JCL sample build script from SHARE
  - See also similar SYS1.SAMPLIB(GTZSHCKJ)
  - See also z/OS Unix: /usr/lpp/bcp/samples/hzssmake.mk

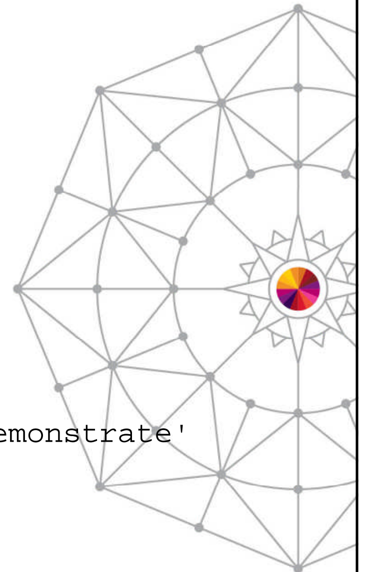
```
//SHRDSVHJ JOB TIME=NOLIMIT,REGION=0K,NOTIFY=&SYSUID
//* Sample JCL for building the sample health check SHRDSVHC.
//* Parameters
// SET CSRCMBR=SHRDSVHC
…
//* METAL C to HLASM
// SET PARM1='/METAL NOARMODE LIST LO NOSE SE(/usr/include/metal)'
//COMPILE EXEC PGM=CCNDRVR,REGION=192M,PARM='&PARM1.&PARM2.'
…
//* HLASM to object module
//ASSEMB    EXEC PGM=ASMA90,
//          PARM='RENT,GOFF,LIST(133),FLAG(PAGE0)',
…
//* Object module to load module
//BIND      EXEC PGM=HEWL,
//          PARM='RENT,AC=1,EP=&CSRCMBR.,AMODE=31',
```

Gives use load module “SHRDSVHC”

Register and Run

- For example via HZSPRMxx:

```
ADDREP CHECK(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD)
                      CHECKROUTINE(SHRDSVHC)
                      MESSAGETABLE(*NONE)
                      ALLOWDYNSEV(YES)
                      PARM('THRESHOLD_HIGH(85%)'
                           'THRESHOLD_MED(75%)'
                           'THRESHOLD_LOW(60%)')
                      DATE(20140204)
                      REASON('A sample check to demonstrate'
                             'dynamic severity')
                      SEVERITY(MEDIUM)
                      INTERVAL(01:00)
```



- MODIFY HZSPROC,ADD,PARMLIB=xx

Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

Register and Run – Alternatives

- MODIFY HZSPROC is only “temporary”
  - Does not survive a restart of system nor Health Checker.
- Make “permanent” by adding HZSPRMxx suffix to system parameter HZS in IEASYSxx (recommended)
  - or to the HZSPRM parameter of procedure HZSPROC.
- Or: Write own HZSADDCHECK dynamic exit routine
  - Need product code to register with exit…
  - See also /usr/lpp/bcp/samples/hzscadd.c (METAL C).
  - See also SYS1.SAMPLIB(HZSSADCK) (HLASM)

SHARE in Anaheim

See it run

- By default a check gets run right away when ADDed:

```
- SY40   f hzsproc,add,parmlib=ud
 *SY40  *HZS0003E CHECK(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD):
 *SHRH001E Resource X is nearing exhaustion
  SY40   HZS0403I ADD PARMLIB PROCESSING HAS BEEN COMPLETED
```

- "Manual" run possible besides INTERVAL scheduling:

```
- SY40   f hzsproc,run,check(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD)
  SY40   HZS0400I CHECK(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD):
  RUN PROCESSING HAS BEEN COMPLETED
  SY40   HZS0002E CHECK(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD):
  SHRH001E Resource X is nearing exhaustion
```

- Note the "dynamic" severity (first HIGH, then MEDIUM)

Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

SHARE in Anaheim

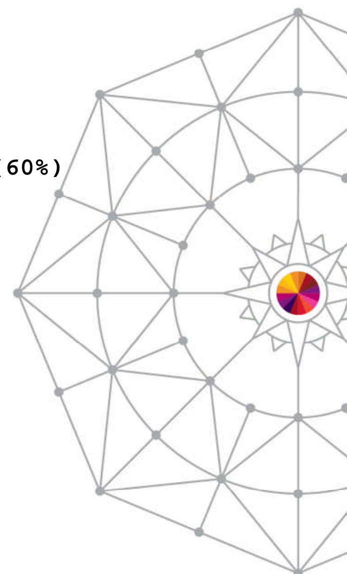
Check iteration details via SDSF CK

```
SDSF HEALTH CHECKER DISPLAY  SY40
COMMAND INPUT ===>
NP NAME                      CheckOwner   State             Status
   RSM_REAL                  IBMRSM       ACTIVE(ENABLED)   SUCCESSFUL
   RSM_RSU                   IBMRSM       ACTIVE(ENABLED)   SUCCESSFUL
   SDUMP_AUTO_ALLOCATION     IBMSDUMP     ACTIVE(ENABLED)   EXCEPTION-MEDIUM
   SDUMP_AVAILABLE           IBMSDUMP     ACTIVE(ENABLED)   SUCCESSFUL
S  SHARE_DYNSEV_THRESHOLD    SHARESAMPLE  ACTIVE(ENABLED)   EXCEPTION-HIGH
   SLIP_PER                  IBMSLIP      ACTIVE(ENABLED)   SUCCESSFUL
   SMS_CDS_REUSE_OPTION      IBMSMS       ACTIVE(ENABLED)   SUCCESSFUL
   SMS_CDS_SEPARATE_VOLUMES  IBMSMS       ACTIVE(ENABLED)   EXCEPTION-MEDIUM
   SUP_LCCA_ABOVE_16M        IBMSUP       ACTIVE(ENABLED)   SUCCESSFUL
```

SHARE in Anaheim

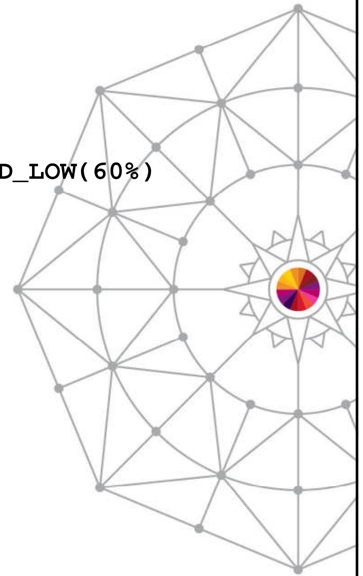
SDSF Message Buffer display – Exception

```
CHECK(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD)
SYSPLEX:    PLEX1     SYSTEM: SY40
START TIME: 02/20/2014 11:19:36.446817
CHECK DATE: 20140204  CHECK SEVERITY: MEDIUM-DYNAMIC
CHECK PARM: THRESHOLD_HIGH(85%),THRESHOLD_MED(75%),THRESHOLD_LOW(60%)

* High Severity Exception *

SHRH001E Resource X is nearing exhaustion

  Explanation:  90% of resource X is in use

  System Programmer Response:  Make more X available

  Reference Documentation:  See book Y about resource X

  Check Reason:  A sample check to demonstrate dynamic severity

END TIME: 02/20/2014 11:19:36.465733  STATUS: EXCEPTION-HIGH
```



SDSF Message Buffer display – Success

```
CHECK(SHARESAMPLE,SHARE_DYNSEV_THRESHOLD)
SYSPLEX:    PLEX1     SYSTEM: SY40
START TIME: 02/20/2014 11:22:10.684570
CHECK DATE: 20140204  CHECK SEVERITY: MEDIUM-DYNAMIC
CHECK PARM: THRESHOLD_HIGH(85%),THRESHOLD_MED(75%),THRESHOLD_LOW(60%)

SHRH002I Resource X is good at 50%.

END TIME: 02/20/2014 11:22:10.684907  STATUS: SUCCESSFUL
```



Complete your session evaluations online at www.SHARE.org/Anaheim-Eval

References

- SHARE Anaheim 2014 – Session 15291
  - Download slides and sample check source / JCL / HZSPRMxx
- SHARE Anaheim 2014 – Session 15114
  - Health Checker V2.1 updates and check writing details and comparisons
- SHARE Boston 2013 – Session 14298
  - Health Checker introduction + REXX check writing
- “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
- Metal C Programming Guide and Reference (SC14-7313)
- Health Checker framework contact and to direct questions about individual health checks: Ulrich Thiemann (thiemanu@us.ibm.com)