Got COBOL?
Bet you didn't know you could do THIS in COBOL

SHARE Session 1541
August 27, 2009 Denver
Tom Ross

Stealth features of COBOL

- **Features of COBOL you may not have heard about**
  - Dynamic file allocation without assembler (not CBLQDA)
  - Environment variables: Setting and Querying
    - At least 3 different ways to set
  - COBOL can be C!
  - XML GENERATE
  - Retrieving SYSTEM info using COBOL
  - Displaying HEX data as printable
  - Testing bit settings in COBOL

Dynamic File allocation in COBOL

Dynamic File allocation in COBOL

```
*************************************************************
* Data declarations for dynamic file allocation             *
*************************************************************
  FILE-CONTROL.
    SELECT REPORT-FILE ASSIGN TO DYNFILE.
*************************************************************
* This is the recommended way to code variable-length files
*************************************************************
  FILE SECTION.
    FD  REPORT-FILE
            RECORD IS VARYING IN SIZE FROM 1 TO 10000
            DEPENDING ON REPORT-SIZE.
     01 REPORT-RECORD                PIC X(10000).

  WORKING-STORAGE SECTION.
     77 REPORT-SIZE                  PIC 9(4) COMP-5.
     01 ENV-VARS.
       05 ENV-NAME                   PIC X(9).
       05 ENV-VALUE                  PIC X(100).
       05 ENV-OVERWRITE              PIC S9(8) COMP-5.
     77 DSNAME                       PIC X(64).
```

Dynamic File allocation in COBOL

```
REPORT-CONDITION.
*********************************************************
*  Now we want to create an OUTPUT file with a DSNAME that *
*  contains an HLQ of choice plus CURRENT-DATE and PROGRAM *
*  First, the Env variable name (replacement for DDNAME)   *
*    (Use null-terminated literal for C setenv service)    *
*********************************************************
   MOVE z'DYNFILE' TO ENV-NAME

*********************************************************
* Create our dynamic DSNAME
*********************************************************
   MOVE FUNCTION CURRENT-DATE(1:14) to FAIL-TIME
   STRING 'HLQ4U.'
          FAILING-PROG
          '.D' FAIL-TIME(3:6)
          '.T' FAIL-TIME(9:6) Delimited By Space
          Into DSNAME Pointer DSN-LEN
```

Dynamic File allocation in COBOL

```
REPORT-CONDITION. (continued)
*********************************************************
*  Now create and set the environment variable which      *
*  will contain dataset info for the dynamic allocation   *
*  First, the dataset DCB information                     *
*********************************************************
   STRING 'DSN(' DSNAME(1:DSN-LEN – 1)'),'
          'NEW,CYL,SPACE(5,5),UNIT(SYSDA)'
          z',CATALOG'  Delimited By Size
          Into ENV-VALUE
*********************************************************
* Now create the environment variable with our info
*********************************************************
   MOVE 1 TO ENV-OVERWRITE
   CALL "setenv" USING ENV-NAME, ENV-VALUE, ENV-OVERWRITE

*********************************************************
* Now allocate and create the dataset!                  *
*********************************************************
   OPEN OUTPUT REPORT-FILE
```

Dynamic File allocation in COBOL

```
*************************************************************
* Now we want to open any fixed length record INPUT file    *
* from parameters passed in to this program.                *
*************************************************************
 ID DIVISION.
  PROGRAM-ID. OPENIT.
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
*************************************************************
* First Define the file in a very generic form              *
*************************************************************
  FILE-CONTROL.
     SELECT IN-FILE ASSIGN TO DYNFILE
     File Status is FILE-STATUS-R.

 DATA DIVISION.

  FD IN-FILE
          RECORD CONTAINS 0
          BLOCK CONTAINS 0
          RECORDING MODE F.
  01 IN-REC                  PIC X(1000).
```

Setting environment variables in COBOL

```
 SET-ENVAR.
*********************************************************
*  Set up an environment variable for IN-FILE           *
*    DSNAME and LRECL are input to program              *
*    ENV-NAME is set before PERFORM SET-ENVAR           *
*********************************************************
      COMPUTE DSN-LEN = 0
      INSPECT DSNAME TALLYING DSN-LEN
                     FOR CHARACTERS BEFORE SPACE
      DISPLAY 'DSN-LEN= ' DSN-LEN

      COMPUTE ENV-NAME-LEN = 0
      INSPECT ENV-NAME TALLYING ENV-NAME-LEN
                       FOR CHARACTERS BEFORE SPACE
      DISPLAY 'ENV-NAME=' ENV-NAME(1:ENV-NAME-LEN) '='

      MOVE 1 TO ENV-VALUE-LEN
      STRING 'DSN('  DSNAME(1:DSN-LEN) '),SHR'
             Delimited By Size
             Into ENV-VALUE With Pointer ENV-VALUE-LEN
      COMPUTE ENV-VALUE-LEN = ENV-VALUE-LEN - 1
```

Setting environment variables in COBOL / LE

```
*************************************************************
* Here is an example of how to set an environment variable  *
* using the CEEENV LE callable service.                     *
*                                                           *
* Set the allocation parameters for the input file that     *
* we want to open and read.                                 *
*************************************************************
      COMPUTE ENV-FUNCTION = 5
      SET ENV-VPTR TO ADDRESS OF ENV-VALUE
      CALL 'CEEENV' USING ENV-FUNCTION, ENV-NAME-LEN,
                          ENV-NAME, ENV-VALUE-LEN
                          ENV-VPTR, FC
      IF SEVERITY > 0 THEN
        DISPLAY "CEEENV failed with Severity = " SEVERITY
        DISPLAY "       and message number   = " MSGNO
      END-IF
```

Setting environment variables in LE – options!

```
*************************************************************
* Here is an example of how to set an environment variable  *
* using the ENVAR LE run-time option.                       *
*************************************************************
// PARM.GO='/ENVAR(“DYNFILE=DSN(TMROSS.COBOL.CARDOUT),SHR”)’
```

Querying environment variables in COBOL

```
*************************************************************
* Here is an example of how to query an environment         *
* variable using the CEEENV LE callable service             *
* In this case we will query 'DYNFILE' since that is the    *
* name in ENV-NAME (and length of name in ENV-NAME-LEN)     *
*************************************************************
      COMPUTE ENV-FUNCTION = 1
      CALL 'CEEENV' USING ENV-FUNCTION, ENV-NAME-LEN,
                          ENV-NAME, ENV-VALUE-LEN
                          ENV-VPTR, FC
      SET ADDRESS OF ENV-STRING TO ENV-VPTR
      DISPLAY 'After set, ENV-VALUE='
              ENV-STRING(1:ENV-VALUE-LEN) '='

      IF SEVERITY > 0 THEN
        DISPLAY "CEEENV failed with Severity = " SEVERITY
        DISPLAY "       and message number   = " MSGNO
```

Dynamic File allocation in COBOL

```
*************************************************************
* Now invoke the SET-ENVAR paragraph and OPEN an input      *
* file based on input parameters                            *
*************************************************************
  PROCEDURE DIVISION USING DSNAME LRECL.

 INIT.

    MOVE 'DYNFILE' TO ENV-NAME
    PERFORM SET-ENVAR.

    OPEN INPUT IN-FILE
    IF FILE-STATUS-R NOT = '00' THen
      DISPLAY 'OPEN IN-FILE failed with File Status = '
          File-Status-R
    ELSE
      DISPLAY 'OPEN IN-FILE successful! File Status = '
          File-Status-R
    End-IF

    READ IN-FILE
    DISPLAY IN-REC(1:LRECL)
    CLOSE IN-FILE

    GOBACK
```

Dynamic File allocation in COBOL – input file

```
*************************************************************
* Now CALL the OPENIT routine for different datasets        *
*************************************************************
WORKING-STORAGE SECTION.

  77 DSNAME-80          PIC X(64) VALUE 'TMROSS.COBOL.CARDOUT'.
  77 DSNAME-133         PIC X(64) VALUE 'TMROSS.COBOL.OPNITLST'.
  77 LRECL-80           PIC S9(9) BINARY  VALUE 80.
  77 LRECL-133          PIC S9(9) BINARY  VALUE 133.

PROCEDURE DIVISION.

 INIT.
    DISPLAY '>>>>>>>>>>> Starting CALLOPN <<<<<<<<<<<<<'

    CALL 'OPENIT' USING DSNAME-80  LRECL-80
    CALL 'OPENIT' USING DSNAME-133 LRECL-133

    DISPLAY '>>>>>>>>>>>  Ending  CALLOPN <<<<<<<<<<<<<'
    GOBACK
    .
END PROGRAM CALLOPN.
```

Dynamic File allocation in COBOL: Output…

```
>>>>>>>>>>> Starting CALLOPN <<<<<<<<<<<<<
******* Entered SET-ENVAR *******
DSNAME = TMROSS.COBOL.CARDOUT
LRECL =  000000080
DSN-LEN= 0000000020
ENV-NAME=DYNFILE=
ENV-VALUE-LEN=0000000029
ENV-VALUE=DSN(TMROSS.COBOL.CARDOUT),SHR=
After set, ENV-VALUE=DSN(TMROSS.COBOL.CARDOUT),SHR=
OPEN IN-FILE successful! File Status = 00
999999   THIS IS THE HEADER RECORD FOR THE CBLREP PROGRAM

******* Entered SET-ENVAR *******
DSNAME = TMROSS.COBOL.OPNITLST
LRECL =  000000133
DSN-LEN= 0000000021
ENV-NAME=DYNFILE=
ENV-VALUE-LEN=0000000030
ENV-VALUE=DSN(TMROSS.COBOL.OPNITLST),SHR=
After set, ENV-VALUE=DSN(TMROSS.COBOL.OPNITLST),SHR=
OPEN IN-FILE successful! File Status = 00
>>>>>>>>>>>  Ending  CALLOPN <<<<<<<<<<<<<
```

COBOL can be C!

COBOL can be C

```
C:
-----------------------------
/* Invoke a C function written in COBOL
{ int x;
   x = C_FUNC(x);  }


COBOL:
-----------------------------
 PROGRAM-ID. C_FUNC.
*************************************************************
* This is a C function to increment a variable that was     *
* passed with C convention rather than System z convention *
*************************************************************
 LINKAGE SECTION.

  77 X             PIC S9(9) COMP-5.
  77 Y             PIC S9(9) COMP-5.

 PROCEDURE DIVISION USING BY VALUE X RETURNING Y.

    ADD 1 TO X GIVING Y

    GOBACK
    .
 END PROGRAM C_FUNC.
```

C++ can be C! How to dynamically CALL from COBOL

C++: Must give an entry point name!

```
 extern "C" int C_FUNC (long& A) {
 cout << "Hello from C++" << endl;
 return A + 1;                    }
```

LKED: ENTRY c_func NAME C_FUNC

COBOL:

```
 PROGRAM-ID. CALLCPP.
*************************************************************
* This is a COBOL program invoking a 'C' function that was *
* written in C++.  Note: dynamic call uppercases sub name  *
*************************************************************
 LINKAGE SECTION.
   77 SUBX          PIC X(8)  VALUE 'c_func'.
   77 X             PIC S9(9) COMP-5.

PROCEDURE DIVISION.

    CALL SUBX USING X RETURNING X
    GOBACK
```

C++ can be C! Or…

C++: No need to give entry point name if FETCHABLE

```
extern "C"
#pragma linkage(dsub,FETCHABLE)
int C_FUNC (A) {
cout << "Hello from C++" << endl;
return A + 1;                    }
```

COBOL:

```
 PROCESS PGMNAME(LONGMIXED)
   PROGRAM-ID. CALLCPP.
*************************************************************
* This is a COBOL program invoking a 'C' function that was *
* written in C++.  Note: dynamic call uppercases sub name  *
*************************************************************
 LINKAGE SECTION.
   77 SUBX          PIC X(8)  VALUE 'c_func'.
   77 X             PIC S9(9) COMP-5.

PROCEDURE DIVISION.

    CALL SUBX USING BY VALUE X RETURNING X
    GOBACK
```

Miscellaneous COBOL features

SET to TRUE

```
*************************************************************
* You can use 88 level condition name values to assign to the   *
* conditional variable using SET                                *
*************************************************************
    01  RESULT-CODE              PIC S9(9) BINARY.
    88  RESUME       VALUE +10.
    88  PERCOLATE    VALUE +20.

    01  EOF-INDICATOR            PIC 9.
    88  NOT-EOF      VALUE 0.
    88  EOF          VALUE 1.



*************************************************************
*       Set to true                                         *
*************************************************************
    SET PERCOLATE TO TRUE


*************************************************************
*       Set to true can be used for false, sort of. . .     *
*************************************************************
    SET NOT-EOF TO TRUE
```

COBOL program cleanup trick: OPT(FULL) + MAP

BL = XXXXX means the data item is not used and you can delete it from the source!

```
Data Division Map
Source   Hierarchy and                                          Base
LineID   Data Name                                              Locators

     5  PROGRAM-ID OPENIT--------------------------------------
*
    45   77 ENV-NAME. . . . . . . . . . . . . . . . . . . . . BLW=00000
    46   77 ENV-VALUE . . . . . . . . . . . . . . . . . . . . BLW=00000
    47   77 DSNAME-80 . . . . . . . . . . . . . . . . . . . . BLW=XXXXX
    48   77 DSNAME-133. . . . . . . . . . . . . . . . . . . . BLW=XXXXX
    49   77 LRECL-80. . . . . . . . . . . . . . . . . . . . . BLW=XXXXX
    50   77 LRECL-133 . . . . . . . . . . . . . . . . . . . . BLW=XXXXX
    59   1  CURRENT-CONDITION . . . . . . . . . . . . . . . . BLL=XXXXX
    60     2  FIRST-8-BYTES . . . . . . . . . . . . . . . . . BLL=XXXXX
    61       3  C-SEVERITY. . . . . . . . . . . . . . . . . . BLL=XXXXX
    62       3  C-MSGNO . . . . . . . . . . . . . . . . . . . BLL=XXXXX
    63       3  C-FC-OTHER. . . . . . . . . . . . . . . . . . BLL=XXXXX
    64       3  C-FAC-ID. . . . . . . . . . . . . . . . . . . BLL=XXXXX
    65     2  C-I-S-INFO. . . . . . . . . . . . . . . . . . . BLL=XXXXX
    66   77 DSNAME. . . . . . . . . . . . . . . . . . . . . . BLL=00001
    67   77 LRECL . . . . . . . . . . . . . . . . . . . . . . BLL=00002
    68   77 ENV-STRING. . . . . . . . . . . . . . . . . . . . BLL=00003
```

OPTIMIZE(FULL) trick

- **OPT(FULL)**
  - Don't use for programs with flags in WORKING-STORAGE unless you add references to them
  - The OPTIMIZER will delete a dead reference anyway, but OPT(FULL) will not delete the data item if it is referenced

```
77 BEG-MARKER PIC X(20) VALUE "BEGINNING OF W-S".
  ...
  ...
IF BEG-MARKER NOT EQUAL BEG-MARKER THEN
   DISPLAY BEG-MARKER.
```

COBOL cross reference of COPY libraries

```
PP 5655-S71 IBM Enterprise COBOL for z/OS  4.1.0  DEMOXREF  Date 10/08/2007 Time 17:03:29 Page 15
      COPY/BASIS cross reference of text-names, library names and dataset information

 Text-name Library   File name            Concat   ISPF statistics
 (Member)  (DDNAME)  (Dataset name)       Level    Created        Changed

 ACTIONS   SYSLIB    USER1.COBOL.COPY      0       1992/07/11   1993/03/16 16:16:17
 CUSTOMER  SYSLIB    USER1.COBOL.LIB2PDSE  1       2007/06/07   2007/06/07 11:28:14
 CUSTOMER  ALTDDXXY  USER1.COBOL.LIB3      0       2007/06/01   2007/06/01 17:35:18
 HOUSE     SYSLIB    USER1.COBOL.LIB2PDSE  1
 HOUSE     ALTDDXXY  USER1.COBOL.LIB2      1       2007/06/07   2007/06/07 11:39:02
 IMOTOR    SYSLIB    USER1.COBOL.LIB4X     3       2007/06/07   2007/06/07 11:37:53
 ISOVERFY  SYSLIB    USER1.COBOL.COPY      0
 NSMAP     SYSLIB    USER1.COBOL.LIB3      2
```

XML messages from COBOL

XML messages from COBOL?

```
*REPORT-CONDITION. (continued)
FILE SECTION.
    FD  REPORT-FILE
            RECORD IS VARYING IN SIZE FROM 1 TO 10000
            DEPENDING ON REPORT-SIZE.
     01 REPORT-RECORD          PIC X(10000).

WORKING-STORAGE SECTION.
*****************************************************************
* Here is the Error Report Group (ERG) of data from which we    *
*  will generate an XML message                                 *
*****************************************************************
   01 ERROR-REPORT-GROUP.
     05 PROGRAM-THAT-FAILED    PIC X(20).
     05 OFFSET-OF-FAILURE      PIC X(20).
     05 FAIL-TIME              PIC X(20).
     05 MESSAGE-NUMBER         PIC X(10).
     05 MESSAGE-TEXT           PIC X(200).
   77 REPORT-SIZE              PIC 9(4) COMP-5.
```

XML messages from COBOL?

```
*REPORT-CONDITION. (continued)
*************************************************************
* Here is an example of how to report a condition using fancy   *
* features of COBOL.  First we will create our own Error Report *
* Group (ERG) of data, then we will create an XML message from  *
* that group and write it to a file that does not exist!        *
*************************************************************
   MOVE FAILING-PROG  to PROGRAM-THAT-FAILED
   MOVE FAILING-OFST  to OFFSET-OF-FAILURE
   MOVE ERROR-MSG     to MESSAGE-TEXT

   XML GENERATE REPORT-RECORD
      FROM ERROR-REPORT-GROUP
      COUNT IN REPORT-SIZE
   END-XML

* Note: no need to create XML in temp storage or to separately
* set the record length for the WRITE with this code!

   WRITE REPORT-RECORD
```

How did it work?

MVS1 - [43 x 127]

File Edit View Communication Actions Window Help

File Edit Edit_Settings Menu Utilities Compilers Test Help

```
EDIT       TMROSS.SSRANGE.D070207.T103139
Command ===>
****** ***************************************************** Top of Data **
000001 <ERROR-REPORT-GROUP>
000002   <PROGRAM-THAT-FAILED>SSRANGE</PROGRAM-THAT-FAILED>
000003   <OFFSET-OF-FAILURE>00002232</OFFSET-OF-FAILURE>
000004   <FAIL-TIME>20070207103139</FAIL-TIME>
000005   <MESSAGE-TEXT>
000006     IGZ0006S The reference to table TABLE-1 by verb number 01 on
000007     line 000108 addressed an area outside the region of the table.
000008   </MESSAGE-TEXT>
000009 </ERROR-REPORT-GROUP>
****** ***************************************************** Bottom of Data
```

How to retrieve system info using COBOL

```
Working-Storage Section.
 01 Results.
   05 sys-name Pic x(8).
   05 real-storage-m Pic s9(6) comp.
   05 prodi Pic x(8).
   05 prodn Pic x(8).
   05 mdl Pic 9999.
   05 smf-name Pic x(4).
   05 ipl-jdate Pic 9(7) comp-3.
   05 SU-sec Pic s9(6) comp.
   05 cpu-type Pic x(6).
   05 cpu-model Pic x(3).
   05 sysplex-name  Pic x(8).
   05 os390-release Pic 9(6).
   05 hardware-name Pic x(8).
   05 lpar-name Pic x(8).
   05 VM-userid Pic x(8).
```

How to retrieve system info using COBOL

```
        01 four-bytes.
           05 full-word Pic s9(8) Comp.
           05 ptr4  Redefines full-word Pointer.
           05 pl4   Redefines full-word Pic 9(7) comp-3.
       Linkage Section.
        01 cb1.  05 ptr1 Pointer Occurs 512.
        01 cvt.  05 cvt1 Pointer Occurs 512.

       Procedure Division.
      * First the Prefixed Save Area, with invalid address ☺
PSA       SET ADDRESS OF cb1 TO NULL
      * Then the Communication Vector Table
CVT       SET ADDRESS OF cvt TO ptr1(5)
          MOVE cvt(341:8) TO sys-name
          MOVE cvt(857:4) TO four-bytes
          COMPUTE real-storage-m = (full-word + 1023) / 1024
          DISPLAY ' SYSNAME='  sys-name
          DISPLAY ' STOR='     real-storage-m 'M'
```

How to retrieve system info using COBOL

```
       * Next the CVT Prefix Area
CVTFIX   SET ptr4 to ADDRESS OF cvt
         SUBTRACT 256 FROM full-word
         SET ADDRESS OF cb1 TO ptr4
         MOVE cb1(217:8) TO prodi
         MOVE cb1(225:8) TO prodn
         MOVE ZERO to pl4
         MOVE cb1(251:2) TO four-bytes(1:2)
         COMPUTE mdl = pl4 / 1000
         DISPLAY ' PRODI=' prodi
         DISPLAY ' PRODN=' prodn
         DISPLAY ' MODEL=' mdl
```

How to retrieve system info using COBOL

```
        * Next the System Management Control Area
SMCA      SET ADDRESS OF cb1 TO cvt1(50)
          MOVE cb1(17:4) TO smf-name
          MOVE cb1(341:4) TO four-bytes
          COMPUTE ipl-jdate = pl4 + 1900000
          DISPLAY ' SMFSID='   smf-name
          DISPLAY ' IPL='      ipl-jdate
        * Next the System Resources Manager Control Table
RMCT      SET ADDRESS OF cb1 TO cvt1(152)
          MOVE cb1(65:4) TO four-bytes
          COMPUTE SU-sec = 16000000 / full-word
          DISPLAY ' Speed='   SU-sec ' SU/s'
```

How to retrieve system info using COBOL

```
        * Host ID
HID       SET ADDRESS OF cb1 TO cvt1(268)
          MOVE cb1(27:6) TO cpu-type
          MOVE cb1(33:3) TO cpu-model
          DISPLAY ' CPU='     cpu-type '-' cpu-model
        * Extended CVT
ECVT      SET ADDRESS OF cb1 TO cvt1(36)
          MOVE cb1(9:8) TO sysplex-name
          MOVE cb1(337:8) TO hardware-name
          MOVE cb1(345:8) TO lpar-name
          MOVE cb1(353:8) TO VM-userid
          MOVE cb1(513:6) TO os390-release

          DISPLAY ' SYSPLEX=' sysplex-name
          DISPLAY ' HWNAME='  hardware-name
          DISPLAY ' LPAR='    lpar-name
          DISPLAY ' VM='      VM-userid
          DISPLAY ' OS/390='  os390-release
```

How to retrieve system info using COBOL

```
* What I got on STLMVS1

SYSNAME=STLMVS1
STOR=007168M
PRODI=SP7.0.8
PRODN=HBB7730
MODEL=2064
SMFSID=MVS1
IPL=2007017
Speed=011165 SU/s
CPU=002064-216
SYSPLEX=STLMVS
HWNAME=SYSZ23
LPAR=STLMVS1
VM=
OS/390=010800
```

How to display hex values using COBOL

```
Program-ID. HexView.
* Display a printable and readable address

01  HEXVAL   PIC X(8).
01  HEXSTR   PIC X(16) VALUE "0123456789ABCDEF".
01  DEC      PIC S9(4) COMP.
01  FILLER   REDEFINES DEC.
    02  FILLER PIC X.
    02  DECBYTE PIC X.
01  I   PIC S9(8) COMP.
01  J   PIC S9(8) COMP.
01  Q   PIC S9(8) COMP.
01  R   PIC S9(8) COMP.
01  J1  PIC S9(8) COMP.
01  Q1  PIC S9(8) COMP.
01  R1  PIC S9(8) COMP.
LINKAGE SECTION.
01  HEXIN    PIC X(4).
01  DECNUM   REDEFINES HEXIN PIC S9(8) COMP.
```

How to display hex values using COBOL

```
* Display a printable and readable address

PROCEDURE DIVISION USING HEXIN.
    DISPLAY "Hex         " HEXIN
    DISPLAY "Dec         " DECNUM
    PERFORM CONVERT
    DISPLAY "Printable   " HEXVAL
    GOBACK.
CONVERT.
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
      COMPUTE J = 2 * I - 1
      MOVE HEXIN(I:1) TO DECBYTE
      DIVIDE DEC BY 16 GIVING Q REMAINDER R
      COMPUTE J1 = J + 1
      COMPUTE Q1 = Q + 1
      COMPUTE R1 = R + 1
      MOVE HEXSTR(Q1:1) TO HEXVAL(J:1)
      MOVE HEXSTR(R1:1) TO HEXVAL(J1:1)
    END-PERFORM.
```

How to display hex values using COBOL

```
*********************************************************
* What I got when I called HEXVIEW from SYSTEM          *
* with my CVT address                                   *
*********************************************************

Hex         "ÙJy
Dec         16634280
Printable   00FDD1A8
```

Another way to display hex values using COBOL

```
Identification division.  Program-id. 'TOHEX' is recursive.
Data division.  Working-storage section.
  1 hexv.
    2 pic x(32) value '000102030405060708090A0B0C0D0E0F'.
    2 pic x(32) value '101112131415161718191A1B1C1D1E1F'.
    2 pic x(32) value '202122232425262728292A2B2C2D2E2F'.
    2 pic x(32) value '303132333435363738393A3B3C3D3E3F'.
    2 pic x(32) value '404142434445464748494A4B4C4D4E4F'.
    2 pic x(32) value '505152535455565758595A5B5C5D5E5F'.
    2 pic x(32) value '606162636465666768696A6B6C6D6E6F'.
    2 pic x(32) value '707172737475767778797A7B7C7D7E7F'.
    2 pic x(32) value '808182838485868788898A8B8C8D8E8F'.
    2 pic x(32) value '909192939495969798999A9B9C9D9E9F'.
    2 pic x(32) value 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'.
    2 pic x(32) value 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'.
    2 pic x(32) value 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'.
    2 pic x(32) value 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'.
    2 pic x(32) value 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'.
    2 pic x(32) value 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'.
  1 redefines hexv. 2 hex pic xx occurs 256 times.
```

Another way to display hex values using COBOL

```
Local-storage section.
  1 i pic 9(9) binary.
  1 j pic 9(9) binary value 0.
  1 jx redefines j.
   2 jxl pic x.
   2 pic xx.
   2 jxb pic x.
  1 k pic 9(9) binary value 1.
  1 redefines k.
   2 kl binary pic 9(4).
   2 kb binary pic 9(4).
 Linkage section.
  1 ostr.
   2 ostrv pic xx occurs 1024 times.
  1 istr.
   2 istrv pic x occurs 1024 times.
  1 len pic 9(9) binary.
```

Another way to display hex values using COBOL

```
Procedure division using ostr istr value len.

    perform with test before varying i from 1 by 1
        until i > len
      move 0 to j
      if kl = 0
        move istrv(i) to jxb
      else
        move istrv(i) to jxl
      end-if
      add 1 to j
      move hex(j) to ostrv(i)
    end-perform
    goback
    .
End program tohex.
```

How to test bits in COBOL – hex literals

```
WORKING-STORAGE SECTION.
 01  One-byte          pic X.

*****************************************
* To test all bits at once              *
*****************************************

*****************************************
* All on?                               *
*****************************************
  If One-byte = x'FF' Then

*****************************************
* Bit pattern 00001000 ?                *
*****************************************
  If One-byte = x'08' Then
```

How to test bits in COBOL – use math

```
*********************************************
* Test setting of only one bit              *
*********************************************

WORKING-STORAGE SECTION.
 01  Half-word          pic s9(4) binary value zero.
 01  filler redefines half-word.
     05  filler         pic x.
     05  packed-bits    pic x.
 01  unpacked-bits.
     05  byte-switch    pic x occurs 8.

LINKAGE SECTION.
*********************************************
* Byte with bits to test                    *
*********************************************
 77  Byte-2-check       Pic x.
```

How to test bits in COBOL – use math

```
Move Byte-2-check to packed-bits
Move zero to unpacked-bits

If half-word > 127
  subtract 128 from half-word
  move '1' to byte-switch (1)
end-if
If half-word > 63
  subtract 64 from half-word
  move '1' to byte-switch (2)
end-if
If half-word > 31
  subtract 32 from half-word
  move '1' to byte-switch (3)
end-if
```

How to test bits in COBOL – use math

```
If half-word > 15
  subtract 16 from half-word
  move '1' to byte-switch (4)
end-if
If half-word > 7
  subtract 8 from half-word
  move '1' to byte-switch (5)
end-if
If half-word > 3
  subtract 4 from half-word
  move '1' to byte-switch (6)
end-if
If half-word > 1
  subtract 2 from half-word
  move '1' to byte-switch (7)
end-if
If half-word = 1
  move '1' to byte-switch (8)
end-if
```

How to test bits in COBOL – use math refined

```
*************************************************
* Check only the fifth bit from the right       *
*         111x1111                              *
*            |                                  *
*************************************************

Move Byte-2-check to packed-bits
Move zero to unpacked-bits

If half-word > 127
  subtract 128 from half-word
end-if
If half-word > 63
  subtract 64 from half-word
end-if
If half-word > 31
  subtract 32 from half-word
  Display 'Fifth bit is on!'
end-if
```

How to test bits in COBOL – use LE

```
*************************************************
* This is the officially recommended way to do it… *
* The LE services work on full-words, so we are    *
* going to cheat by passing our byte and adjusting *
* the bit number to make it work                   *
*************************************************
* First set parms to point to the fifth bit from   *
* the right of the byte:                           *
*         111x1111 11111111  111111111111111       *
*            |               |               |     *
*           28              15              0      *
*************************************************
Move 28 to Bit-Number

CALL 'CEESITST' USING Byte-2-check Bit-number FC Result

If Result = 1 Then
  Display 'Fifth bit is on!'
```

How to set bits in COBOL

```
*************************************************
* This is the officially recommended way to do it… *
* The LE services work on full-words, so we are    *
* going to cheat by passing our byte and adjusting *
* the bit number to make it work                   *
*************************************************
* First set parms to point to the fifth bit from   *
* the right of the byte:                           *
*         111x1111 11111111  1111111111111111      *
*            |               |               |     *
*           28              15              0      *
*************************************************
Move 28 to Bit-Number

CALL 'CEESISET' USING Byte-2-set Bit-number FC Byte-2-set

Display 'Fifth bit is on!'
```