#SHAREorg

z/OMG The Next COBOL Compiler Has Arrived!

Tom Ross

May 21, 2014

Standard Legal Disclaimer

Introducing Enterprise COBOL V5

- Announced April 23, GA June 21 (2013)
- Introduces advanced optimization technology
  - Designed to optimize applications for current and future System z hardware
  - Initiate delivery of performance improvements seen in C/C++ and Java compilers on System z
- Compiler “back end” is replaced with technology that has long been in use in IBM's Java products. (Back end = part of compiler that does code generation and optimization)
  - Mature, robust compilation technology.
  - New COBOL-specific optimizations have been added.
- Exploits z990, z890, System z9, System z10, zEnterprise 196, zEC12 and zBC12.

New Code Generator and Program Optimizer

- Common components means more timely exploitation of future zArchitecture advances.
- Support modern development tools
  - Tools supplied by ISV's
  - IBM z/OS Problem Determination Tools
  - Rational Development Tools
- Continue to deliver new features
  - to simplify programming and debugging to increase productivity
  - to modernize existing business critical applications
- Use industry standard DWARF, with documented IBM extensions to represent debug information.
  - APIs are available to allow tools to inspect this information.

New Compiler Options for performance

- ***ARCH (6 | 7 | 8 | 9 | 10)***
  - Allows code generator to use instructions found in various levels of z Architecture
- ***OPTIMIZE(0 | 1 | 2)***
  - Levels of optimization
    - Higher levels improve run time performance
    - Highest level has somewhat reduced "debuggability"
- ***STGOPT / NOSTGOPT***
  - Allows compiler to delete unreferenced data items
- ***HGPR (PRESERVE | NOPRESERVE)***
  - Use high word of registers (upper 32 bits of 64-bit registers)
  - Effectively adds 16 more registers to improve optimization
- ***AFP(VOLATILE | NOVOLATILE)***
  - Use full complement of floating point registers.

New Compiler Options for usability

- *DISPSIGN(SEP)*
  - *DISPSIGN controls output formatting for DISPLAY of signed numeric items.*
  - *Can format overpunch sign as separate sign for easier to read output:*

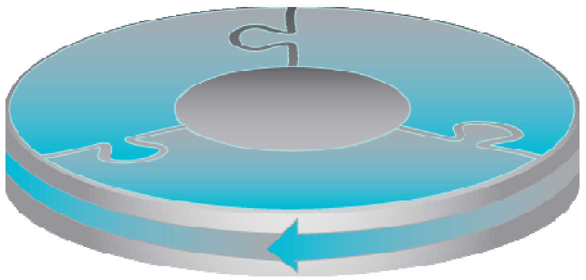
DISPLAY output with	DISPSIGN(COMPAT):	DISPSIGN(SEP):
positive binary	111	+111
negative binary	11J	-111
positive packed-decimal	222	+222
negative packed-decimal	22K	-222

- *LVLINFO* (installation option)
  - Now 8 bytes instead of 4, you can put APAR, PTF, or your own numbers
  - Example: LVLINFO=PN123456
  - Listing header:

```
PP 5655-W32 IBM Enterprise COBOL for z/OS  5.1.0 PN123456
                                        Date 05/20/2013  Time 10:45:03
Signature bytes:
00088E (+40) 00408000              =X'00408000'     INFO. BYTES 24-27
000892 (+44) D7D5F1F2F3F4F5F6      =C'PN123456'      USER LEVEL INFO (LVLINFO)
      Compiler Options and Program Information Section End
```

Compatibility

- Provide Source and binary compatibility
- Most correct COBOL programs will compile and execute without changes and produce the same results
  - “Old” and “new” code can be mixed within an application and communicate with static, dynamic and DLL calls
  - No need to recompile entire applications to take advantage of new V5 features
- Removed some old language extensions and options
  - Millennium Language Extensions
  - Label Declaratives
  - Non-reentrant programs above 16MB line
  - OS/VS COBOL Inter-operation
  - COBOL V3 (COMPAT) XML PARSER
  - Static AMODE 24 CALLs



COBOL language removed

- *Millennium Language Extensions*
- **The removed elements are:**
  - **`DATE FORMAT` clause on data description entries**
  - **`DATEVAL` intrinsic function**
  - **`UNDATE` intrinsic function**
  - **`YEARWINDOW` intrinsic function**
  - **`DATEPROC` compiler option**
  - **`YEARWINDOW` compiler option**

COBOL language removed

- *LABEL DECLARATIVES*

Format 2 declarative syntax:

```
USE ... AFTER ... LABEL PROCEDURE
```

And the syntax:

```
GO TO MORE-LABELS
```

are no longer supported.

  - **Note: GO TO is still supported.**

ARCH compiler option details

Performance Improvements at all ARCH Levels

- The compiler accepts ARCH(6) – ARCH(10) all of which also exploit
  - Relative Instruction
    - Jumps (branches) and nested program calls can be relative to the executing instruction
    - Access to the literal pool can also be relative to the executing instruction
  - Half word immediate instructions
    - Load, Load Logical ANDs, ORs, Add and Subtract logical
  - Twelve additional floating point registers
  - Long Displacement Facility
    - Many load/store instructions that have a 0-4095 displacement now have a “Y” format with a -524,288 → 524,287 displacement reach
    - Now one base register can cover much more working storage and this reduces need for base locators
  - 64 bit “G” form instructions
    - 64 bit computations can be done in single registers vs piecewise in 32 bit registers
    - Particularly useful for improving performance of COBOL BINARY data with more than 9 digits

More Performance at Higher ARCH Levels

z9
ARCH(7)

Extended Immediates

• 32 bit immediates in arithmetic, logical, compare instructions

• Save registers, literal pool space and executes faster

z10
ARCH(8)

Decimal Floating Point

• Larger packed multiply/divide in DFP registers vs library call

• Also benefits other decimal types converted to packed for arithmetic

Wider Immediate Moves

• 2,4,8 byte move immediate instructions

• Benefits applications with lots of VALUE clauses and MOVES

z196/z114
ARCH(9)

Distinct Operands

• Many arithmetic,shift instructions take two source and produce non-destructive result to third register

• Better register allocation, particularly useful in striding through tables

zEC12/zBC12
ARCH(10)

DFP/Zoned Conversions

• More efficient conversions between zoned decimal and DFP

• Enables much greater use of DFP

• One example performs division in 0.22x cost vs staying in zoned at lower arch levels

ARCH quick reference

- ARCH(6)
  - 2084-xxx models (z990)
  - 2086-xxx models (z890)
- ARCH(7)
  - 2094-xxx models (IBM System z9 EC)
  - 2096-xxx models (IBM System z9® BC)
- ARCH(8)
  - 2097-xxx models (IBM System z10 EC)
  - 2098-xxx models (IBM System z10 BC)
- ARCH(9)
  - 2817-xxx models (IBM zEnterprise z196 EC)
  - 2818-xxx models (IBM zEnterprise z114 BC)
- ARCH(10)
  - 2827-xxx models (IBM zEnterprise EC12)
  - 2828-xxx models (IBM zEnterprise BC12)

- That sounds good, how about some code generation examples to show you ?

LONG DISPLACEMENT INSTRUCTIONS

```
Linkage Section.
  01 DfhCommArea.
     02 DfhStuff      Pic x(32757).
     02 DfhName       Pic x(6).
Procedure Division Using
   DfhCommArea.
```

MAP output – V4

```
1  DFHCOMMAREA . . . . . BLL=00001
  2  DFHSTUFF. . . . . . BLL=00001
  2  DFHNAME . . . . . . BLL=00008
```

MAP output – V5

```
1  DFHCOMMAREA . . . . . BLL=00001
  2  DFHSTUFF. . . . . . BLL=00001
  2  DFHNAME . . . . . . BLL=00001
```

V4

- ***Loop to initialize 8 BLL cells***

```
         LA     1,0(0,1)
         ST     1,308(0,9)         BLL=1
         L      8,308(0,9)         BLL=1
         L      15,16(0,10)
         LA     14,308(0,9)        BLL=1
GN=13    EQU    *
         AL     1,12(0,10)
         AH     14,24(0,10)
         ST     1,0(0,14)
         BCT    15,324(0,11)       GN=13
```

V5

- ***Only one BLL***
- ***All ARCH levels***

```
L        R0,0(,R1)
NILH     R0,32767
ST       R0,0(,R8)
```

Timing (100 million in a loop)

```
V5 : 4.44 cpu seconds
V4 : 5.15 cpu seconds
```

V5 is 14% faster

Decimal Divide Where Operands Exceed Packed Decimal Hardware Limits

```
1 z14v2 pic s9(14)v9(2).
1 z13v2 pic s9(13)v9(2).
...
Compute z14v2 = z14v2 / z13v2
```

V4

- ***Calls out to library routine***
- ***Runtime path length is > 100 instructions***

```
PACK  344(9,13),0(16,2)
PACK  360(16,13),16(15,2)
MVC   376(32,13),59(10)
MVC   398(9,13),344(13)
NI    406(13),X'F0'
MVN   407(1,13),352(13)
L     3,92(0,9)
L     15,180(0,3)
LA    1,146(0,10)
BASR  14,15
NI    431(13),X'0F'
ZAP   431(9,13),431(9,13)
UNPK  0(16,2),431(9,13)
```

V5

- ***Inlined with 6 instructions***
- ***CDZT/CZDT are new EC12 instructions to convert between zoned and DFP types***
- ***ARCH (10)***

```
CDZT     FP0,152(16,R8),0x8
CDZT     FP1,168(15,R8),0x8
SLDT     FP0,FP2,2
DDTR     FP0,FP0,FP1
FIDTR    FP0,9,FP0
CZDT     FP0,152(16,R8),0x9
```

Timing (100 million in a loop)

```
V5 : 1.08 cpu seconds
V4 : 4.81 cpu seconds
```

V5 is 78% faster

Binary Arithmetic Conditional Precision Correction

```
1 b6v2a pic s9(6)v9(2) comp.
1 b6v2b pic s9(6)v9(2) comp.
...
Compute b6v2a = b6v2a + b6v2b
```

V4

- ***Divide (D) to correct precision always executed but rarely needed***

```
L      3,8(0,4)
A      3,0(0,4)
LR     2,3
SRDA   2,32(0)
D      2,0(0,12)
```

V5

- ***Divide (DR) to correct precision only executed when actually required***
- ***ARCH(8)***

```
          L        R0,152(,R8)
          A        R0,160(,R8)
          IILF     R2,X'05F5E100'
          LPR      R1,R0
          CLFI     R1,X'05F5E100'
          JL       L081
          SRDA     R0,32
          DR       R0,R2
L081:     EQU      *
          ST       R0,152(,R8)
```

Timing (100 million in a loop)

```
V5 : 0.18 cpu seconds
V4 : 0.52 cpu seconds
```

V5 is 65% faster

Binary Arithmetic Operands Greater Than 9 Digits

```
1 b8v2a pic s9(8)v9(2) comp.
1 b8v2b pic s9(8)v9(2) comp.
...
Compute b8v2a = b8v2a + b8v2b
```

V4

- ***Piecewise arithmetic plus decimal conversions***

```
LM     2,3,0(4)
A      2,8(0,4)
AL     3,12(0,4)
BC     12,126(0,11)
A      2,4(0,12)
D      2,0(0,12)
CVD    3,376(0,13)
MVO    360(6,13),379(5,13)
CVD    2,376(0,13)
TM     365(13),X'10'
MVC    365(5,13),379(13)
BC     8,162(0,11)
OI     369(13),X'01'
MVI    363(13),X'00'
NI     364(13),X'0F'
MVC    376(8,13),103(10)
MVC    379(5,13),365(13)
CVB    2,376(0,13)
MVO    379(5,13),360(5,13)
CVB    7,376(0,13)
M      6,0(0,12)
ALR    7,2
BC     12,210(0,11)
A      6,4(0,12)
LTR    2,2
BC     11,220(0,11)
S      6,4(0,12)
STM   6,7,0(4)
```

V5

- ***Makes use of 'G' format 64 instructions***
- ***Conditional precision correction***
- ***ARCH(6)***

```
      LLIHF    R2,X'00000002'
      IILF     R2,X'540BE400'
      LG       R0,152(,R8)
      AG       R0,160(,R8)
      LPGR     R1,R0
      CLGR     R1,R2
      JL       L082
      LGR      R1,R0
      DSGR     R0,R2
      STG      R0,152(,R8)
L082  EQU      *
```

Timing (100 million in a loop)

```
V5 : 0.23 cpu seconds
V4 : 1.92 cpu seconds
```

V5 is 88% faster

Instruction Scheduling For Performance

```
 1 z7v2a pic s9(7)v9(2).
 1 z7v2b pic s9(7)v9(2).
 1 z7v2c pic s9(7)v9(2).
...
ADD 1 TO z7v2a z7v2b z7v2c
```

V4 – OPTIMIZE

- *Instructions appear in original order and subject to hardware read after write penalties*

```
PACK  344(5,13),0(9,2)
AP    344(5,13),51(2,10)
ZAP   344(5,13),344(5,13)
UNPK  0(9,2),344(5,13)
PACK  344(5,13),16(9,2)
AP    344(5,13),51(2,10)
ZAP   344(5,13),344(5,13)
UNPK  16(9,2),344(5,13)
PACK  344(5,13),32(9,2)
AP    344(5,13),51(2,10)
ZAP   344(5,13),344(5,13)
UNPK  32(9,2),344(5,13)
```

V5 – OPT(2)

- *Independent operations are grouped to reduce read after write hardware penalties*
- *ARCH(8)*

```
PACK    352(5,R13),152(9,R8)
PACK    344(5,R13),168(9,R8)
PACK    336(5,R13),184(9,R8)
AP      352(5,R13),416(2,R3)
AP      344(5,R13),416(2,R3)
AP      336(5,R13),416(2,R3)
ZAP     352(5,R13),352(5,R13)
ZAP     344(5,R13),344(5,R13)
ZAP     336(5,R13),336(5,R13)
UNPK    152(9,R8),352(5,R13)
UNPK    168(9,R8),344(5,R13)
UNPK    184(9,R8),336(5,R13)
```

Timing – (100 million in a loop)

```
V5 : 2.35 cpu seconds
V4 : 2.50 cpu seconds
```

V5 is 6% faster

Optimization of Decimal PICTURE Scaling

```
1 p8v0  pic 9(9) COMP-3.
1 p10v2 pic s9(10)v9(2) COMP-3.
...
COMPUTE p10v2 = p8v0 / 100
```

V4

- ***Explicit instructions for both decimal shift and decimal divide***

```
ZAP    344(8,13),0(5,2)
SRP    346(6,13),2(0),0
DP     344(8,13),42(2,10)
ZAP    8(7,2),344(6,13)
```

V5

- ***The optimizer cancels out the decimal shift and decimal divide***
- ***All ARCH levels***

```
MVC        337(5,R13),152(R8)
MVN        341(1,R13),157(R8)
ZAP        160(7,R8),152(5,R8)
```

Timing (100 million in a loop)

```
V5 : 0.31 cpu seconds
V4 : 2.02 cpu seconds
```

V5 is 85% faster

Optimization of Initialization By Literals

```
01 WS-GROUP.
  05 WS1-COMP3 COMP-3 PIC S9(13)V9(2).
  05 WS2-COMP  COMP   PIC S9(9)V9(2).
  05 WS3-COMP5 COMP-5 PIC S9(5)V9(2).
  05 WS4-COMP1 COMP-1.
  05 WS5-ALPHANUM  PIC X(11).
  05 WS6-DISPLAY   PIC 9(13) DISPLAY.
  05 WS7-COMP2 COMP-2.
```

```
Move +0 to WS3-COMP5
           WS1-COMP3
           WS2-COMP
           WS6-DISPLAY
           WS4-COMP1
           WS7-COMP2
           WS5-ALPHANUM
```

V4

- *Individual initializing stores are generated*
- *34 instruction bytes*

```
LA     2,0(0,0)
L      3,300(0,9)
ST     2,16(0,3)
MVC    0(8,3),188(10)
MVC    8(8,3),177(10)
MVC    35(13,3),163(10)
ST     2,20(0,3)
MVC    48(8,3),177(10)
MVI    24(3),X'F0'
MVC    25(10,3),4(12)
```

V5

- *Entire out of order initializing sequence is collapsed to a single instruction*
- *6 instruction bytes*
- *All ARCH levels*

```
MVC      152(56,R2),920(R3)
```

Timing (100 million in a loop)

V5 : 0.16 cpu seconds

V4 : 0.25 cpu seconds

V5 is 36% faster

New compiler features introduced

- Improved usability
  - Reduced administration overhead with support for z/OS System Management Facilities (SMF) records
  - New NOLOAD debugging segments in program object
    - Debugging data always matches executable
    - No separate debugging files to find or keep track of
    - Executable does not have bigger loaded footprint
  - New pseudo-assembly in program listings

Some New COBOL language features

Some New COBOL language features

- *Floating comment delimiter*
  - **> to end of line is a comment*
- *Raise WORKING-STORAGE section size limit to 2GB*
  - *(from 128MB)*
- *Larger individual data items*
  - *Up to 999,999,999 bytes!*
- *Support for UNBOUNDED tables*
  - *X OCCURS 1 To UNBOUNDED Depending on Y.*
  - *LINKAGE SECTION only*

Some new COBOL language introduced

- New Intrinsic Functions to improve handling of UTF-8 data
- XML GENERATE features for controlling document generation
  - NAME OF phrase
    - User supplied element and attribute names
  - TYPE OF phrase
    - User control of attribute and element generation
  - SUPPRESS phrase
    - Suppression of "empty" attributes and elements
- XML PARSE feature for easier handling of split content:
  - XML-INFORMATION special register

UTF-8 Unicode Built-in Functions

UTF-8 Characters are 1 – 4 bytes in length.

- ***ULENGTH:*** *returns the logical length of a UTF-8 string*
- ***UPOS:*** *returns the byte position in a UTF-8 string of the Nth logical character.*
- ***USBSTR:*** *returns the sub-string of N logical characters starting from a given logical character.*
- ***UVALID:*** *takes an alphanumeric or alpha or national item and returns zero or the index of the first invalid UTF-8 (alphanumeric or alpha) or UTF-16 (national) character.*
- ***UWIDTH:*** *returns the width in bytes of the Nth logical character.*
- ***USUPPLEMENTARY:*** *takes a UTF-8 or UTF-16 string and returns zero or the first UNICODE supplementary character.*

Examples of COBOL new features

- We have 3 example programs
  - New UTF-8 Intrinsic Functions
  - New XML GENERATE features
  - New XML PARSE features
- UTF-8 example
  - Takes an XML document as input in UTF-8
  - There is a bad character (not UTF-8) that causes XML PARSE to fail
  - Use UTF-8 functions to locate and fix bad char

New UTF-8 Intrinsic Functions

```
  PROCESS CODEPAGE(1153)
*-------------------------------------------------------------
* Sample program to illustrate what happens when XML PARSE
* is used with an input UTF-8 document that has been corrupted
*-------------------------------------------------------------
 Identification Division.
   Program-id. UTF8B4.
 Data Division.
  Working-Storage section.
   1 i Comp pic 99.
*-------------------------------------------------------------
* XML document with Czech characters in EBCDIC
*-------------------------------------------------------------
   1 d pic x(99) value
       '<Grp><D1>1324.56</D1><D2>Leoš Janáèek</D2></Grp>'.
   1 u pic x(99).
Procedure Division.
*-------------------------------------------------------------
* Translate XML document from EBCDIC to UTF-8
*-------------------------------------------------------------
     Move Function Display-of( Function National-of(d) 1208 )
                                  to u
```

New UTF-8 Intrinsic Functions

```
*------------------------------------------------------------
* Introduce deliberate invalid UTF-8 character into document
*------------------------------------------------------------
     Move '5' to u(37:1)
*------------------------------------------------------------
* Attempt to Parse the damaged XML document
*------------------------------------------------------------
     Display 'Parsing UTF-8 document:'
     Xml Parse u encoding 1208 processing procedure h
        On Exception Move 16 To Return-Code
                Display ' '
                Display '>> PARSE failed!! <<'
                Display ' '
     End-XML
     Goback.
```

New UTF-8 Intrinsic Functions

OUTPUT:

```
Parsing UTF-8 document:
  XML event name                  XML-CODE  {XML-TEXT}
  -----------------------------------------------
  START-OF-DOCUMENT               000000000 {}
  START-OF-ELEMENT                000000000 {Grp}
  START-OF-ELEMENT                000000000 {D1}
  CONTENT-CHARACTERS              000000000 {1324.56}
  END-OF-ELEMENT                  000000000 {D1}
  START-OF-ELEMENT                000000000 {D2}
  EXCEPTION                       000798768 {<Grp><D1>1324.56</D1><D2>
                                                           <D2>Leo  Jan}}

 >> PARSE failed!! <<
```

New UTF-8 Intrinsic Functions

- How do we avoid the XML PARSE exception?
- There is no IBM provided way to validate UTF-8 data in Enterprise COBOL V4
- You could write a UTF-8 checker, but it would take many LOC in COBOL to do it
  - You would have to maintain that code!
- In comes Enterprise COBOL V5.1 …

New UTF-8 Intrinsic Functions

```
 Process CODEPAGE(1153)
*-----------------------------------------------------------------
* Sample program to illustrate use of the new Unicode
* intrinsic Functions for manipulating UTF-8 character strings
*-----------------------------------------------------------------
 Identification Division.
   Program-id. UTF8CLAS.
 Data Division.
  Working-storage section.
   1  i            Comp pic 99 Value 1.
   88 Valid-UTF-8  Value 0.
*-----------------------------------------------------------------
* XML document with Czech characters in EBCDIC
*-----------------------------------------------------------------
   1 d  pic x(99) value
       '<Grp><D1>1324.56</D1><D2>Leoš Janáèek</D2></Grp>'.
   1 u  pic x(99).
   1 x  Comp pic 99.
   1 y  Comp pic 99.
   1 z  Comp pic 99.
```

New UTF-8 Intrinsic Functions

```
 Procedure Division.
*------------------------------------------------------------------
* Translate XML document from (viewable) EBCDIC to UTF-8
*------------------------------------------------------------------
     Move Function Display-of(Function National-of(d) 1208) to u
*------------------------------------------------------------------
* Introduce deliberate invalid UTF-8 character into document
*------------------------------------------------------------------
     Move '5' to u(37:1)
*------------------------------------------------------------------
* Attempt to parse the damaged XML document
*------------------------------------------------------------------
     Perform Parse
     Perform UTF-8-check
     If Not Valid-UTF-8
       Perform Repair-It
     End-If
*------------------------------------------------------------------
* Re-attempt the XML Parse if document OK now
*------------------------------------------------------------------
     If Valid-UTF-8
       Perform Parse
     End-If
```

New UTF-8 Intrinsic Functions

```
*------------------------------------------------------------------
* Use COBOL XML Parse statement to analyze the XML document:
*------------------------------------------------------------------
   Parse.
     Display 'Parsing UTF-8 document:'
     Xml Parse u encoding 1208 processing procedure h
             On Exception Move 16 To Return-Code
                     Display ' '
                     Display '>> PARSE failed!! <<'
                     Display ' '
             Not On Exception Move 2 To Return-Code
                     Display ' '
                     Display '>> PARSE success!! <<'
                     Display ' '
     End-XML.
```

New UTF-8 Intrinsic Functions

The following code can check your UTF-8 before parse

```
UTF-8-check.
    Compute i = Function UVALID(u)
    If Valid-UTF-8
      Display 'UTF-8 character string is valid.'
    Else
      Display 'Bad UTF-8 character sequence at position ' i ';'
    End-if.
```

New UTF-8 Intrinsic Functions

OUTPUT:

```
Parsing UTF-8 document:
  XML event name                 XML-CODE  {XML-TEXT}
  --------------------------------------------------
  START-OF-DOCUMENT              000000000 {}
  START-OF-ELEMENT               000000000 {Grp}
  START-OF-ELEMENT               000000000 {D1}
  CONTENT-CHARACTERS             000000000 {1324.56}
  END-OF-ELEMENT                 000000000 {D1}
  START-OF-ELEMENT               000000000 {D2}
  EXCEPTION                      000798768 {<Grp><D1>1324.56</D1><D2> }

>> PARSE failed!! <<

Bad UTF-8 character sequence at position 37;
```

New UTF-8 Intrinsic Functions

The following code will better diagnose bad UTF-8

```
UTF-8-check.
    Compute i = Function UVALID(u)
    If Valid-UTF-8
      Display 'UTF-8 character string is valid.'
    Else
      Display 'Bad UTF-8 character sequence at position ' i ';'
      Compute x = Function ULENGTH(u(1:i - 1))
      Compute y = Function UPOS(u x)
      Compute z = Function UWIDTH(u x)
      Display 'The ' x 'th and last valid character starts '
          'at byte ' y ' for ' z ' bytes.'
    End-if.
```

New UTF-8 Intrinsic Functions

OUTPUT:

```
Parsing UTF-8 document:
  XML event name                 XML-CODE  {XML-TEXT}
  --------------------------------------------------
  START-OF-DOCUMENT              000000000 {}
  START-OF-ELEMENT               000000000 {Grp}
  START-OF-ELEMENT               000000000 {D1}
  CONTENT-CHARACTERS             000000000 {1324.56}
  END-OF-ELEMENT                 000000000 {D1}
  START-OF-ELEMENT               000000000 {D2}
  EXCEPTION                      000798768 {<Grp><D1>1324.56</D1><D2>Leo}

>> PARSE failed!! <<

Bad UTF-8 character sequence at position 37;
The 34th and last valid character starts at byte 35 for 02 bytes.
```

New UTF-8 Intrinsic Functions

The following code can ‘repair’ bad UTF-8 data

```
*-------------------------------------------------------------
* Repair the bad UTF-8 character
*-------------------------------------------------------------
   Repair-It.
     Display ' '
     Display 'Repairing bad UTF-8 sequence...'
     Perform Test after until i = 0
*-------------------------------------------------------------
*      x'30' is 0 (zero) in UTF-8
*-------------------------------------------------------------
       Move x'30' to u(i:1)
       Compute i = Function UVALID(u)
     End-perform.
```

New UTF-8 Intrinsic Functions

OUTPUT:

```
Parsing UTF-8 document:
  XML event name                 XML-CODE  {XML-TEXT}
  --------------------------------------------------
  START-OF-DOCUMENT              000000000 {}
  START-OF-ELEMENT               000000000 {Grp}
  START-OF-ELEMENT               000000000 {D1}
  CONTENT-CHARACTERS             000000000 {1324.56}
  END-OF-ELEMENT                 000000000 {D1}
  START-OF-ELEMENT               000000000 {D2}
  EXCEPTION                      000798768 {<Grp><D1>1324.56</D1><D2>Leo}

>> PARSE failed!! <<

Bad UTF-8 character sequence at position 37;
The 34th and last valid character starts at byte 35 for 02 bytes.
```

New UTF-8 Intrinsic Functions

OUTPUT cont.:

```
Repairing bad UTF-8 sequence...
Parsing UTF-8 document:
  XML event name              XML-CODE  {XML-TEXT}
  --------------------------------------------------
  START-OF-DOCUMENT           000000000 {}
  START-OF-ELEMENT            000000000 {Grp}
  START-OF-ELEMENT            000000000 {D1}
  CONTENT-CHARACTERS          000000000 {1324.56}
  END-OF-ELEMENT              000000000 {D1}
  START-OF-ELEMENT            000000000 {D2}
  CONTENT-CHARACTERS          000000000 {Leo00 Jan  ek}
  END-OF-ELEMENT              000000000 {D2}
  END-OF-ELEMENT              000000000 {Grp}
  END-OF-DOCUMENT             000000000 {}

>> PARSE success!! <<
```

Examples of COBOL new features

- We have 3 example programs
  - New UTF-8 Intrinsic Functions
  - New XML GENERATE features
  - New XML PARSE features
- XML GENERATE example
  - Generates an XML document from a group, but we have done post-processing the document to
    - Remove ‘empty’ entries
    - Change tag names:
      - Different from what is in structure
      - Not legal as data item names
      - Use a COBOL reserved word
    - Select which values are ELEMENT and which are ATTRIBUTES
  - Create correct XML document output the first time
    - Post-processing was the only solution in COBOL V4

XML GENERATE features: before

```
 Process DYNAM
*----------------------------------------------------------------
* Demonstrate missing features of XML Generate statement
* in Enterprise COBOL V4.2
*----------------------------------------------------------------
 Identification division.
   Program-Id. XMLGB4.
 Data Division.
  Working-Storage Section.
   77 DOC Pic x(9999).
   01 Inventory.
     05 CBX-764-WSR-LOC Pic x(30).
     05 Product-Count comp Pic 999.
     05 Product Occurs 10 times.
       10 Description Pic x(20).
       10 Quantity comp Pic 999.
       10 Date-Acquired Pic x(10).
```

XML GENERATE features: before

```
 Procedure Division.
*-----------------------------------------------------------------
* Fill data structure, Generate default XML, and "pretty-print" it
*-----------------------------------------------------------------
     Perform Set-Up-Inventory
     Xml Generate DOC from Inventory Count in Tally
     Display "XML GENERATE produced " Tally " bytes of output"
*-----------------------------------------------------------------
* Notice several issues with the default XML:
*   - Unwanted table entries with zero values
*   - Inappropriate or unappealing tag names
*-----------------------------------------------------------------
     Call 'pretty' using DOC Tally
     Goback.
```

XML GENERATE features: before

```
*------------------------------------------------------------------
* Set up data structure with sample values. Notice that, although
* the table has ten entries, only three contain relevant data.
*------------------------------------------------------------------
   Set-Up-Inventory.
     Initialize Inventory
     Move 'Orlando' to CBX-764-WSR-LOC
     Add 1 to Product-Count
     Move 'Carbon filter' to Description(Product-Count)
     Move 34 to Quantity(Product-Count)
     Move '04/12/2012' to Date-Acquired(Product-Count)
     Add 1 to Product-Count
     Move '100'' Hose' to Description(Product-Count)
     Move 20 to Quantity(Product-Count)
     Move '08/25/2012' to Date-Acquired(Product-Count)
     Add 1 to Product-Count
     Move 'Palette' to Description(Product-Count)
     Move 120 to Quantity(Product-Count)
     Move '06/01/2011' to Date-Acquired(Product-Count).
 End program XMLGB4.
```

XML GENERATE features: before

```
Program-Id. PRETTY.
   .  .  .
Procedure Division using doc value len.
   .  .  .

  XML PARSE doc Processing Procedure P
  Goback
  .
p.
  Evaluate xml-event
    When 'VERSION-INFORMATION'
      String '<?xml version="' xml-text '"' delimited by size
          into buffer with pointer posd
      Set xml-declaration to true
    When 'ENCODING-DECLARATION'
      String ' encoding="' xml-text '"' delimited by size
          into buffer with pointer posd
    When 'STANDALONE-DECLARATION'
      String ' standalone="' xml-text '"' delimited by size
          into buffer with pointer posd
```

XML GENERATE subprogram ‘pretty’

```
  When 'START-OF-ELEMENT'
   Evaluate true
     When xml-declaration
       String '?>' delimited by size into buffer
           with pointer posd
       Set unknown to true
       Perform printline
       Move 1 to posd
     When element
       String '>' delimited by size into buffer
           with pointer posd
     When attribute
       String '">' delimited by size into buffer
           with pointer posd
   End-evaluate
 If elementName not = space
   Perform printline
 End-if
 Move xml-text to elementName
 Add 1 to depth
 Move 1 to pose
 Set element to true
```

XML GENERATE features: before

OUTPUT:

```
XML GENERATE produced 01169 bytes of output
<Inventory>
  <CBX-764-WSR-LOC>Orlando</CBX-764-WSR-LOC>
  <Product-Count>3</Product-Count>
  <Product>
    <Description>Carbon filter</Description>
    <Quantity>34</Quantity>
    <Date-Acquired>04/12/2012</Date-Acquired>
  </Product>
  <Product>
    <Description>100' Hose</Description>
    <Quantity>20</Quantity>
    <Date-Acquired>08/25/2012</Date-Acquired>
  </Product>
```

XML GENERATE features: before

OUTPUT (cont.):

```
<Product>
  <Description>Palette</Description>
  <Quantity>120</Quantity>
  <Date-Acquired>06/01/2011</Date-Acquired>
</Product>
<Product>
  <Description> </Description>
  <Quantity>0</Quantity>
  <Date-Acquired> </Date-Acquired>
</Product>
<Product>
  <Description> </Description>
  <Quantity>0</Quantity>
  <Date-Acquired> </Date-Acquired>
</Product>
<Product>
  <Description> </Description>
  <Quantity>0</Quantity>
  <Date-Acquired> </Date-Acquired>
</Product>
```

XML GENERATE features: before

OUTPUT (cont.):

```
  <Product>
    <Description> </Description>
    <Quantity>0</Quantity>
    <Date-Acquired> </Date-Acquired>
  </Product>
  <Product>
    <Description> </Description>
    <Quantity>0</Quantity>
    <Date-Acquired> </Date-Acquired>
  </Product>
  <Product>
    <Description> </Description>
    <Quantity>0</Quantity>
    <Date-Acquired> </Date-Acquired>
  </Product>
  <Product>
    <Description> </Description>
    <Quantity>0</Quantity>
    <Date-Acquired> </Date-Acquired>
  </Product>
</Inventory>
```

XML GENERATE features: after

```
  Process DYNAM
*--------------------------------------------------------------
* Demonstrate features of XML Generate statement added to
* Enterprise COBOL V5.1
*--------------------------------------------------------------
 Identification division.
   Program-Id. XMLGCLAS.
 Data Division.
  Working-Storage Section.
   77 DOC Pic x(9999).
*--------------------------------------------------------------
* Use the same structure for source of XML
*--------------------------------------------------------------
   01 Inventory.
     05 CBX-764-WSR-LOC Pic x(30).
     05 Product-Count comp Pic 999.
     05 Product Occurs 10 times.
       10 Description   Pic x(40).
       10 Quantity comp Pic 9(3).
       10 Date-Acquired Pic x(10).
```

XML GENERATE features: after

Add the following phrases to XML GENERATE :

```
Xml Generate DOC from Inventory Count in tally
    Name of CBX-764-WSR-LOC is 'Warehouse'
            Description is 'Desc'
            Quantity is 'No.'
            Date-Acquired is 'Date'
    Type of  Quantity is Attribute
    Suppress Every Nonnumeric Element When SPACE
             Every Numeric When ZERO
 End-xml
 Display "XML GENERATE produced " Tally " bytes of output"
 Call 'pretty' using DOC tally
 Goback.
```

XML GENERATE features: after

OUTPUT:

```
XML GENERATE produced 00312 bytes of output
<Inventory>
  <Warehouse>Orlando</Warehouse>
  <Product-Count>3</Product-Count>
  <Product No.="34">
    <Desc>Carbon filter</Desc>
    <Date>04/12/2012</Date>
  </Product>
  <Product No.=“20">
    <Desc>100' Hose</Desc>
    <Date>08/25/2012</Date>
  </Product>
  <Product No.=“120">
    <Desc>Palette</Desc>
    <Date>06/01/2011</Date>
  </Product>
</Inventory>
```

Examples of COBOL new features

- We have 3 example programs
  - New UTF-8 Intrinsic Functions
  - New XML GENERATE features
  - New XML PARSE features
- XML PARSE example
  - XMLSS parser can give split content
    - ATTRIBUTE-CHARACTERS
    - CONTENT-CHARACTERS
  - Example shows how to handle possible split content
    - Without XML-INFORMATION (Ugly!)
    - What terminates an attribute value?
      - Almost any event! But no event for '>' (end of tag)
      - Have to buffer attribute value separately from elements
    - With XML-INFORMATION special register

XML PARSE features: before

```
handler.
     evaluate xml-event
       when 'START-OF-DOCUMENT'
         move 0 to attr-bufr-ctr cont-bufr-ctr
         move 1 to attr-bufr-ptr cont-bufr-ptr
       when 'ATTRIBUTE-NAME'
         perform collect-attr-bufr
         move xml-text to attr-name
       when 'ATTRIBUTE-CHARACTERS'
         perform append-attr-bufr
       when 'COMMENT'
       when 'NAMESPACE-DECLARATION'
       when 'PROCESSING-INSTRUCTION-TARGET'
       when 'START-OF-CDATA-SECTION'
         perform collect-attr-bufr
       when 'CONTENT-CHARACTERS'
         perform collect-attr-bufr
         perform append-cont-bufr
```

XML PARSE features: before

```
handler.                             *> continued
       when 'END-OF-ELEMENT'
         perform collect-attr-bufr
         perform collect-cont-bufr
       when 'START-OF-ELEMENT'
         perform collect-attr-bufr
         perform collect-cont-bufr
         move xml-text to elmt-name
       when 'ATTRIBUTE-NATIONAL-CHARACTER'
         perform unsupported-event
       when 'CONTENT-NATIONAL-CHARACTER'
       when 'UNRESOLVED-REFERENCE'
         perform collect-attr-bufr
         perform unsupported-event
       when other
         continue
     end-evaluate.
```

XML PARSE features: before

```
collect-attr-bufr.
  if attr-bufr-ptr > 1
    subtract 1 from attr-bufr-ptr
    if attr-name = 'this'
      move attr-bufr(1:attr-bufr-ptr) to this
    else
      move attr-bufr(1:attr-bufr-ptr) to that
    end-if
    display attr-bufr-ctr ' segments of attribute "' attr-name
            '" of element "' elmt-name '"'
    display '    reassembled, length ' attr-bufr-ptr ':'
    display "  '" attr-bufr(1:13) '...'
            attr-bufr(attr-bufr-ptr - 2:3) "'"
    display ' '
    move 0 to attr-bufr-ctr
    move 1 to attr-bufr-ptr
    move space to attr-name
  end-if.
```

XML PARSE features: before

```
append-attr-bufr.
  string xml-text delimited by size into attr-bufr
      with pointer attr-bufr-ptr
  add 1 to attr-bufr-ctr
  display 'Buffering segment ' attr-bufr-ctr ' of attribute "'
               attr-name '" of element "' elmt-name '"'.
```

New XML PARSE features

- XML PARSE features: before
  - Lots of code ‘just in case’ content gets split
  - Example is minimized, real world example is even worse
- XML PARSE features: after
  - XML-INFORMATION tells us when content is complete
    - Only need 1 buffer since collecting attribute data will not be ended by element content
    - Can do all work within code for ATTRIBUTE-CHARACTERS and CONTENT-CHARACTERS events
      - Not spread all over the program

XML PARSE features: after

```
handler.
     Evaluate xml-event
       When 'START-OF-DOCUMENT'
         move 1 to bufr-ptr       *> Only 1 buffer ptr to init
       When 'ATTRIBUTE-NAME‘        *>  No setup necessary
         Move xml-text to attr-name *> Just save the name
       When 'ATTRIBUTE-CHARACTERS‘  *> Handle attribute value
         Evaluate XML-INFORMATION
           When 1                   *> If content is complete
             Perform get-attr-bufr  *> Get last piece
             If attr-name = 'this'
               Move char-bufr(1:bufr-ptr) to this
             Else
               Move char-bufr(1:bufr-ptr) to that
             end-if
           When 2                   *> If split content
             Perform get-char-bufr  *> Get next piece
           When Other               *> Error condition
             Call ‘CEE3ABND’
         End-Evaluate
```

XML PARSE features: after

```
handler.                                *> Continued
       When 'NAMESPACE-DECLARATION'
       When 'PROCESSING-INSTRUCTION-TARGET'
       When 'START-OF-CDATA-SECTION‘
       When 'COMMENT‘                   *> Nothing to do here for
         Continue                       *> buffer data ‘after’
       When 'CONTENT-CHARACTERS‘        *> Handle element value
         Evaluate XML-INFORMATION
           When 1                       *> If content is complete
             Perform get-attr-bufr      *> Get last piece
             Evaluate element-name      *> Move into data item
               When ‘xyz’
                 Move char-bufr(1:bufr-ptr) to xyz
               etc, etc
               . . .
             End-Evaluate
           When 2                       *> If split content
             Perform get-char-bufr      *> Get next piece
           When Other                   *> Error condition
             Call ‘CEE3ABND’
         End-Evaluate
```

XML PARSE features: after

```
handler.                                    *> Continued
       when 'END-OF-ELEMENT‘                *> Nothing to do here for
         Continue                           *> buffer data ‘after’
       when 'START-OF-ELEMEN                 *> Nothing to do here for
         Continue                           *> buffer data ‘after’
         move xml-text to elmt-name
       when 'ATTRIBUTE-NATIONAL-CHARACTER'
         perform unsupported-event
       when 'CONTENT-NATIONAL-CHARACTER'
       when 'UNRESOLVED-REFERENCE‘          *> Nothing to do here for
         Continue                           *> buffer data ‘after’
         perform unsupported-event
       when other
         continue
     end-evaluate.
```

XML PARSE features: after

```
get-char-bufr.
  string xml-text delimited by size into char-bufr
      with pointer bufr-ptr
  display 'Buffer content so far = ‘
           char-bufr(1:bufr-ptr)
```

- Debug Tool improvements for COBOL V5

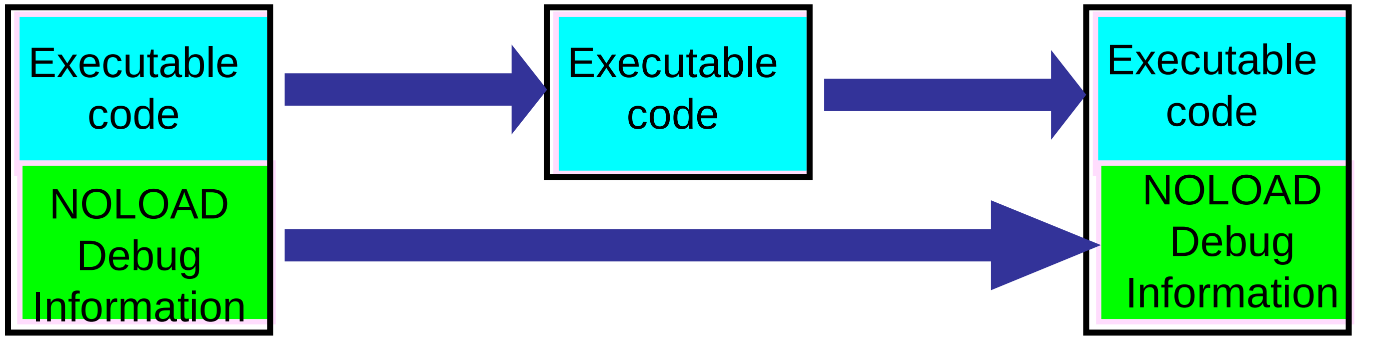
Debug Tool improvements for COBOL V5

- Debug Tool was completely re-instrumented to work with COBOL V5.1:
  - Access to DWARF debug data in NOLOAD classes
  - Change to Debug Tool ‘Level 4 APIs’ from historic level 1
  - New COBOL runtime and COBOL debug support runtime
- As we worked, the question was often posed:

  Do we implement this the old way or this obviously better way?

- A few of the many improvements in the Debug Tool experience with COBOL V5.1:
  - STEP OVER of PERFORM statements
  - Improved presentation of tables (arrays)
  - Improved presentation of data descriptions

Storage used by COBOL V5 program objects compiled w/TEST



Program Object
On disk
(Load Library)

Program Object
In Memory
(Loaded/running,
No Debug Tool)

Program Object
In Memory
(Loaded/debugging
Debug Tool also
running)

Debug Tool improvements for COBOL V5

STEP OVER of PERFORM

```
    When 'START-OF-ELEMENT'
     Evaluate true
       When xml-declaration
         String '?>' delimited by size into buffer
             with pointer posd
         Set unknown to true
         Perform println
         Move 1 to posd
       When element
         String '>' delimited by size into buffer
             with pointer posd
       When attribute
         String '">' delimited by size into buffer
             with pointer posd
     End-evaluate
   If elementName not = space
     Perform println
   End-if
```

Debug Tool improvements for COBOL V5

Improved presentation of tables (arrays)

Debug Tool with COBOL V4:

```
LIST PRODUCT ( 3 ) ;
SUB(3) of 03 XMLGB4:>DESCRIPTION  of 02 XMLGB4:>PRODUCT  =
'Palette              '
SUB(3) of 03 XMLGB4:>QUANTITY  of 02 XMLGB4:>PRODUCT  = 00120
SUB(3) of 03 XMLGB4:>DATE-ACQUIRED  of 02 XMLGB4:>PRODUCT  =
    '06/01/2011'
```

Debug Tool with COBOL V5:

```
 LIST PRODUCT ( 3 ) ;
10 DESCRIPTION of 05 PRODUCT(3) = 'Palette              '
10 QUANTITY of 05 PRODUCT(3) = 00120
10 DATE-ACQUIRED of 05 PRODUCT(3) = '06/01/2011'
```

Debug Tool improvements for COBOL V5

Improved presentation of data descriptions

Debug Tool with COBOL V4:

```
DESCRIBE ATTRIBUTES INVENTORY ;
ATTRIBUTES for INVENTORY
 Its length is 352
 Its address is 0DF7C480
 01 XMLGB4:>INVENTORY
   02 XMLGB4:>CBX-764-WSR-LOC   X(30) DISP
   02 XMLGB4:>PRODUCT-COUNT   999 COMP
   02 XMLGB4:>PRODUCT   AN-GR OCCURS 10
     03 XMLGB4:>DESCRIPTION   X(20)
       SUB(1)  DISP
       SUB(2)  DISP
       SUB(3)  DISP
       SUB(4)  DISP
       SUB(5)  DISP
       SUB(6)  DISP
       SUB(7)  DISP
       SUB(8)  DISP
       SUB(9)  DISP
       SUB(10)  DISP
     03 XMLGB4:>QUANTITY   999 ‘

        etc
        etc
```

Debug Tool improvements for COBOL V5

Debug Tool with COBOL V5:

```
DESCRIBE ATTRIBUTES INVENTORY ;
 ATTRIBUTES for INVENTORY
    Its length is 352
    Its address is  0E010E20
    01 INVENTORY
      05 CBX-764-WSR-LOC  x(30) DISP
      05 PRODUCT-COUNT  999 COMP
      05 PRODUCT  OCCURS 10
          10 DESCRIPTION  x(20) DISP
          10 QUANTITY  9(3) COMP
          10 DATE-ACQUIRED  x(10) DISP
```

Connect With Us

Cafes

C/C++
http://ibm.com/rational/community/cpp

COBOL
http://ibm.com/rational/community/cobol

Fortran
http://ibm.com/rational/community/fortran

PL/I
http://ibm.com/rational/community/pli

Feature Requests

C/C++
http://ibm.com/developerworks/rfe/?PROD_ID=700

COBOL
http://ibm.com/developerworks/rfe/?PROD_ID=698

Fortran
http://ibm.com/developerworks/rfe/?PROD_ID=701

PL/I
http://ibm.com/developerworks/rfe/?PROD_ID=699

Like IBM Compilers on Facebook

Follow IBM Compilers on Twitter

Enterprise COBOL Service: PTF1!

- APARs fixed in the September PTF1 bundle:
  - **COMPILER UK96988/UK96989/UK97247 PTFs**
    PM92585 - COBOL version 5 fixes for problems identified in beta program and Japanese message updates
    PM95418 - CMPL MSGIGYCB7104-U Internal compiler error and RC16 using options offset and test
    PM95906 - Message number 1307 could not be found for facility ID IGY
  - **RUNTIME UK96719/UK96720 PTFs**
    PM93979 - Move static initialization to the heap
    PM95114 - COBOL runtime sort ABENDs in DFSORT
    PM95117 - COBOL performance degradation in procedure pointer call
    PM95118 - COBOL runtime error in handling external files plus error when using procedure pointer
    PM93345 - XML enhancements(z/OS 2.1 only)

Enterprise COBOL Service: PTF2!

- APARs fixed in the October PTF2 bundle:
  - **COMPILER UK98481/UK98482/UK98483/UK98499 PTFs**
    PM92523 - IMS support enhancement SQLIMS
    PM92894 - ABEND322 loop in IGYCDGEN during compile of COBOL program using NOTEST(DWARF)
    PM96176 - IGYWDOPT and IGYWUOPT are missing from SIGYSAMP
    PM97763 - Changing DISPSIGN compiler option default to SEP fails
    PM97939 - Compiler creates invalid special register table
  - **RUNTIME UK98140/UK98141 PTFs**
    PM98032 - The external file I/O verb may use the wrong version of the I/O routines and ABEND

Enterprise COBOL Service: PTF3!

- APARs fixed in the January PTF3 bundle:
  - **COMPILER UI14448 PTF**
    PI05656 - CMPL loop in compile of COBOL/SQL coprocessor program with "REPLACE" and missing "END-EXEC.“
    PI05657 - IGYPS5062-U There was insufficient storage.
    PI05658 - COBOL COPY...REPLACING errors using EXEC to replace partial dataname or paragraph name.
    PI06128 - IGYDS0197-E "11" was a name that started with an underscore
    PI06899 - No compiler error recieved for abbreviated IF statement with confusion about implied subject.
    PI08238 - Compiler generates incorrect code for PERFORM UNTIL statement.
    PM99261 - Expected division-by-zero message is not being printed
  - **RUNTIME UI14246(V1R13) / UI14247(V2R1) PTFs**
    PI09629 - UNSTRING statement can be inefficient if the input string is too long when delimiter is not present in the input string

Enterprise COBOL Service: PTF4! AKA: V5R1M1

- APARs fixed in the March PTF4 bundle:
  - **COMPILER UI16133/UI16134/UI16135 PTFS**
    PM93583 - COBOL 5.1.1 - UPDATE TO ADD AMODE 24 SUPPORT TO ENTERPRISE COBOL VERSION 5.1
    PI07531 - IGYCB7145-U insufficient memory at compile time
    PI11399 - Compiler error when mixing PERFORM & PERFORM w/THRU
    PI11805 - V5.1 batch compilation that specifies DLL may fail with error IGYCB7104-U with "Failed assertion on ./WCode/WCodeDefs.hpp:261“
    PI13222 - COBOL 5.1 compile with OPT(1) returns error IGYCB7104-U - Failed assertion on ./Register.cpp:1034

Enterprise COBOL Service: PTF4! AKA: V5R1M1

- APARs fixed in the March PTF4 bundle:
  - **RUNTIME UI15839(V1R13) / UI15840(V2R1) PTFs**
    PI08326 CEE3201S followed by ABENDU4083 when COBOL program specifies sort parm LOCALE=FR_CA
    PI10522 COBOL version 5 program not entered in last used state when 1st called from a COBOL version 4 program
    PI10647 COBOL V5.1 0C4 ABEND using VSAM file with VSHARE during VSAM EXIT
    PI11295 USUPPLEMENTARY function returns unexpected results for national characters & Language Reference Guide has a USUPPLEMENTARY typo
    PI11389 API routine to query the COBOL working storage area PI12151 COBOL runtime enhancement for AMODE(24)
    PI12928 COBOL V5 runtime event handler does not handle LE Event 31 properly so WORKING-STORAGE address/length unavailable
    PI13285 Wrong conversion of blanks when using codepage 937 DBCS

Enterprise COBOL Developer Trial

- Zero cost evaluation license for 90 days
  - Does not initiate Single Version Charging (SVC)

- Assess the value that could be gained from upgrading to Enterprise COBOL V5.1

- Offer same functionalities as Enterprise COBOL for z/OS V5.1
  - Same pre-requisites (e.g. runs on z/OS V1.13 and z/OS V2.1…)
  - Code compiled with Enterprise COBOL Developer Trial cannot be used for production

- Available as standard offering from IBM through ShopzSeries on Oct 4, 2013
  - Contact your IBM representative for ordering assistance

http://www-03.ibm.com/software/products/ph/en/enterprise-cobol-developer-trial-for-zos

Enterprise COBOL Design Partner Program

Program Mission:

To involve clients early in the design and development process of our products to improve ***quality***, deliver the right ***strategy and features***, increase client ***satisfaction and loyalty***, and secure ***references***.

Benefits to participants

- Direct input on design of new COBOL features
- Visibility into product strategy and roadmaps
- Early experience with pre-release drivers

Nomination:

https://www.ibm.com/software/support/trial/cst/forms/nomination.wss?id=2279

Program contacts:

- Marie Bradford mabrad@us.ibm.com
- Roland Koo rkoo@ca.ibm.com

- Questions?

PDSE requirement for COBOL V5 executables

- COBOL V5 executables are not “load modules”. They are “program objects”. Load modules reside in a PDS dataset. Program objects can only reside in a PDSE dataset (or z/OS UNIX file).
- Therefore, customers using PDS load libraries for COBOL executables must migrate to PDSE load libraries prior to creating COBOL V5 executables. There is no alternative to converting.
- If interested in COBOL V5, start migrating COBOL load libraries to PDSE datasets ASAP!
- Now, why PDSE datasets and why are PDSE datasets better than PDS datasets?

First some history about PDS datasets

- When using PDS datasets for load libraries, customers had problems with :
  - The need for frequent compressions,
  - Loss of data due to the directory being overwritten
  - Performance impact due to a sequential directory search
  - Performance delay if member added to beginning of directory
  - Problems when PDS went into multiple extents

First some history about PDS datasets

- More problems with PDS dataset load libraries:
  - PDS datasets could not share update access to members without an enqueue on the entire data set.
  - The biggest drawback to PDS load libraries was that they had to be taken offline from time to time for:
    - A compression to reclaim member space or
    - Directory reallocation to reclaim directory gas
  - Because of this, applications could not have 24/7/365 access

Introducing PDSE datasets for load libraries!

- PDSEs, which were introduced in 1990, were designed to eliminate or at least reduce these problems
- They have! It's unfortunate that the rollout of PDSEs was so painful (lots and lots of APARs) that many sites have steered clear of them
- OTOH, many sites HAVE moved their COBOL load libraries to PDSEs, it is fairly mechanical

How to migrate from PDS load libraries to PDSE load libraries:

- Assuming the conversion of an entire PDS to a PDSE, the general steps are as follows:
  - Allocate a new PDSE dataset, such as &pds.PDSE, where “&pds” is the PDS dataset name.
  - Use IEBCOPY (or ISPF) to copy the load modules from the PDS into the PDSE.
    - This will automatically convert the load modules to program objects in the PDSE.
  - Rename the PDS. Example: &pds.BACKUP. Retain this dataset (short term) for recovery purposes.
  - Rename the PDSE to &pds, where “&pds” is the original PDS dataset name.

How to migrate from PDS load libraries to PDSE load libraries, some notes:

- Any Load Module in a PDS can be copied into a PDSE
  - It then becomes a Program Object
  - Program Management Binder is called by IEBCOPY or ISPF to do the conversion for you
- Not all Program Objects in PDSEs can be copied back to PDS and Load Module form
- This means that if a Program Object member in a PDSE on a test system is then shipped to production, and the receiving dataset on the production system is a PDS, then there could be a copy problem.
- Convert the downstream library first, i.e. convert the production PDS to a PDSE. Then convert the test system PDS to a PDSE.

Why are PDSE load libraries required with COBOL Version 5?

- First some history about Load Modules
  - z/OS has been moving to solve problems due to limitations of Load Modules for years
  - Program Management BINDER has made many changes to solve these problems
  - Many of these solutions required a new format of executable
  - Program Objects was the answer
  - Program Objects have features that cannot be supported by PDS datasets, so they require PDSE datasets

Load Modules versus Program Objects

- Program Management Binder solves existing problems with Load Modules using new features of Program Objects
  - Example: when customers reached 16M text size limit of load module, our answer was always: "Re-engineer programs to be smaller, re-design" …expensive and not well received!
  - A program object can have a text size of up to 1 gigabyte
  - COBOL can take advantage of this by having more constants for improved MOVE and INITIALIZE performance
    - Makes object size bigger

Why are PDSE load libraries required with COBOL Version 5?

- COBOL V4 required Program Objects and thus PDSE for executable for certain features since 2001:
  - Long program names
  - Object-Oriented COBOL
  - DLLs using the Binder instead of prelinker
- COBOL V5 requires Program Objects and thus PDSE load libraries for all executables
- How about some examples of specific features that COBOL V5 has that can only be supported by Program Objects (PO) and PDSE Load libraries?

Why PDSE for COBOL V5 executables?

- COBOL improving performance using new features that are only available in Program Objects (PO)
  - Improved init/term scheme relies on user-defined classes in object, requiring PO
  - QY-con requires PO
    - That's a performance improvement for RXY (long displacement) instructions.
  - Condition-sequential RLD support requires PO
    - Performance improvement for bootstrap invocation
  - PO can get page mapped 4K at a time for better performance

Why PDSE for COBOL V5 executables?

- Other features requiring Program Objects
  - NOLOAD class DWARF debugging data requires PO
  - Common reentrancy model with C/C++ requires PO
  - XPLINK requires PO and will be used for AMODE 64

What about sharing COBOL load libraries across SYSPLEX systems?

- PDSE datasets cannot be shared across SYSPLEX boundaries
- If PDS load libraries are shared across SYSPLEX boundaries today, in order to move to PDSE load libraries, customers can use a master-copy approach
  - One SYSPLEX can be the writer/owner of master PDSE load library (development SYSPLEX)
  - When PDSE load library is updated, push the new copy out to production SYSPLEX systems with XMIT or FTP
  - The other SYSPLEX systems would then RECEIVE the updated PDSE load library

Can I mix PDS and PDSE load libraries?

- If you convert all load libraries to PDSE first, no worries
  - IE: You will no longer have any PDS load libraries
- If you create a new PDSE dataset and put new code there while keeping existing load modules in PDS load library, you could end up using both PDS and PDSE load libraries in a single application:
  - COBOL V5 in PDSE load library can call COBOL V4 in PDS load library without problems (and vice-versa)
  - DYNAMIC CALL only of course
- If you start with COBOL V4 (or V3, V2) code in a PDS load library and recompile one program of a load module with COBOL V5, and then re-BIND, the result will be a Program Object, and will go into a PDSE
  - STATIC CALL in this case