#SHAREorg

SHARE

Z/OMG The Next COBOL Compiler Has Arrived!

Tom Ross

Aug 12, 2013

SHARE in Boston

Standard Legal Disclaimer

Introducing Enterprise COBOL V5

- Announced April 23, GA June 21
- 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
- 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

New Code Generator and Program Optimizer

- 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 and zEC12.

- Common components means more timely exploitation of future zArchitecture advances.

- Uses 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

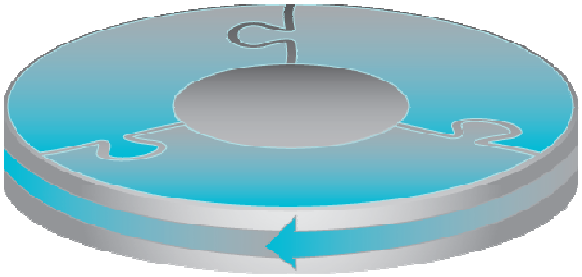
Signature bytes:

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
  - COBOL V3 (COMPAT) XML PARSER
  - Non-reentrant programs above 16MB line
  - OS/VS COBOL Inter-operation
  - AMODE 24



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.

ARCH compiler option details

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

ARCH(6) And Above z990, z890

- Long Displacement Addressing Modes
  - Many instructions that have an addressing mode with 12-bit unsigned displacement now have a corresponding mode with 20-bit signed displacement (can address up to 1048576 bytes)
  - This allows 1 base register to “cover” a great deal more memory
    - Much of the WSA (WORKING-STORAGE) is reachable by 1 reg
  - Unfortunately, SS type instructions such as MVC, which are commonly used in COBOL, are still limited to 12-bit unsigned displacement
    - Secondary interior pointers are computed as needed.

- “Grande” Instructions
  - The ability to do 64-bit computations (binary data of more than 9 digits) in a single 64-bit register

ARCH(7) And Above z9

- Extended Immediate Data
  - Now possible to get immediate values up to 32-bits
  - One common example:
    - CLFI or CLGFI is used to avoid doing MOD operations such as those arising from COBOL Binary data
  - Also used whenever larger immediate values are required in arithmetic, comparison and logical instructions

ARCH(8) And Above z10

- Decimal Floating Point
  - Floating point registers working in base 10 instead of base 2
  - In ARCH(8) we use these for Packed Decimal multiplication and division
  - Note that ZONED and some other data types are converted to Packed for arithmetic, so these operations will also benefit
- Wider Immediate Instructions
  - Many instructions that had a 1-byte immediate form now have 2, 4 and 8 byte versions
  - Initializing constant data is common in COBOL and the wider move immediate instructions are heavily used.

ARCH(9) And Above z196

- Distinct Operand
  - Many arithmetic instructions that update a register in place now allow a third (target) register and leave the contents of the two sources unchanged
  - This allows more efficient register allocation
- Conditional Load
  - A register can be conditionally loaded
    - (avoid surrounding branching logic)

.

ARCH(10) And Above ec12

- More Decimal Floating Point operations
- Conversion between Zoned Decimal and Decimal Floating Point (which is useful for Zoned Decimal arithmetic) is now done with simple hardware instructions.

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, where's the beef?
- How about some code generation examples to show you

LONG DISPLACEMENT INSTRUCTIONS

```
Linkage Section.
  01 DfhCommArea.
     02 DfhStuff      Pic x(32760).
     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

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

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      L0081
         SRDA    R0,32
         DR      R0,R2
L0081:   EQU     *
         ST      R0,152(,R8)
```

Timing (100 million in a loop)

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

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      L0081
LGR     R1,R0
DSGR    R0,R2
STG     R0,152(,R8)
```

Timing (100 million in a loop)

V5 : 0.23 cpu seconds

V4 : 1.92 cpu seconds

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
```

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***

```
ZAP      160(7,R8),152(5,R8)
```

Timing (100 million in a loop)

V5 : 0.31 cpu seconds

V4 : 2.02 cpu seconds

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

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

New 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

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 2 example programs
  - New UTF-8 Intrinsic Functions
  - New XML GENERATE 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 2 example programs to work with
  - New UTF-8 Intrinsic Functions
  - New XML GENERATE features
- XML GENERATE new features
  - 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
  - Show how to improve XML document output without post-processing (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 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>
```

- Debug Tool improvements for COBOL V5

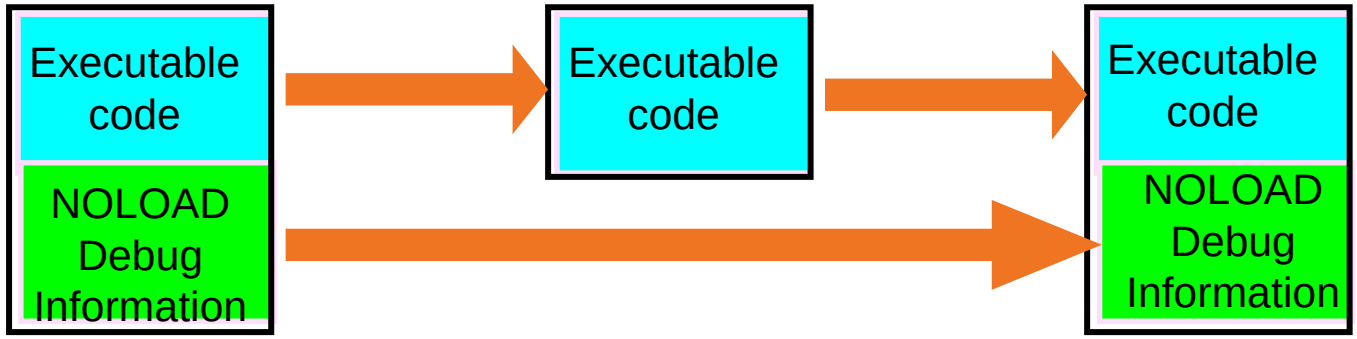
Debug Tool improvements for COBOL V5

- 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 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
```

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

Like IBM Compilers on Facebook

Follow IBM Compilers on Twitter

- Questions?