What's New in Enterprise COBOL Version 4… Release 2!

Tom Ross
IBM

August 2, 2010
Session 7496

Agenda

- New features of Enterprise COBOL V4R2
- Prerequisites for V4R2
- Requirements satisfied by V4R2
- COBOL Performance Tuning Paper updated
  - http://www.ibm.com/software/awdtools/cobol/zos/library/
  - http://www.ibm.com/software/rational/cafe/community/cobol

Briefly

- Enterprise COBOL for z/OS Version 4 Release 2
- General Availability August 28, 2009
- Requires update to Language Environment
  - PK90754
- Full list of prerequisites at end of presentation

New Features

- New in Enterprise COBOL V4R2
  - XML document parsing with validation
  - Performance improvements to parsing XML documents with XMLSS without validation (compared to V4R1 XMLSS)
  - Compiler message severity customization
  - Compiler option BLOCK0 to exploit system-determined block size for QSAM output files
  - COBOL user-defined words can include the underscore (_) character
  - Compiler listings display CICS options in effect
  - More Java SDKs supported: Java 5 and Java 6

New features of V4 Release 2

- How about some details?

XML PARSE with validation

- Validation against XML schema
- No validation against DTD
  - An early requirement for XML PARSE (2001)
  - Good thing we did not do it, schema is better!
- What is validation? An XML schema?

XML PARSE with validation

- What is validation?
  - Validating an XML document determines whether the structure and content of the document conform to a set of rules
  - In Enterprise COBOL, the rules are expressed in an *XML schema*

XML PARSE with validation

- What is an XML schema?
  - A blueprint for a class of documents
  - Expressed in XML
  - Describes and constrains the structure and content of XML documents
  - Using a schema you can control the contents of attributes and elements. You have far more control over what is considered a valid XML document using a schema than with a DTD.

What is an XML schema?

- Consider an XML document that describes an item for stock-keeping purposes:

```
<stockItem itemNumber="453-SR">
<itemName>Stainless steel rope thimbles</itemName>
<quantityOnHand>23</quantityOnHand>
</stockItem>
```

- The example document above is both well formed and valid according to the following schema. (The numbers that precede each line are not part of the schema, but are used in the explanations after the schema.)

What is an XML schema?


```
01. <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
02.   <xsd:element name="stockItem" type="stockItemType"/>
03.   <xsd:complexType name="stockItemType">
04.     <xsd:sequence>
05.       <xsd:element name="itemName" type="xsd:string" minOccurs="0"/>
06.       <xsd:element name="quantityOnHand">
07.         <xsd:simpleType>
08.           <xsd:restriction base="xsd:nonNegativeInteger">
09.             <xsd:maxExclusive value="100"/>    </xsd:restriction>
10.         </xsd:simpleType>
11.       </xsd:element>
12.     </xsd:sequence>
13.     <xsd:attribute name="itemNumber" type="SKU" use="required"/>
14.   </xsd:complexType>
15.   <xsd:simpleType name="SKU">
16.     <xsd:restriction base="xsd:string">
17.       <xsd:pattern value="\d{3}-[A-Z]{2}"/>     </xsd:restriction>
18.   </xsd:simpleType>
19. </xsd:schema>
```

What is an XML schema?

- The schema declares that the root element is stockItem (line 02),

```
01. <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
02.   <xsd:element name="stockItem" type="stockItemType"/>
```

- which has a mandatory itemNumber attribute of type SKU (line 13),

```
13.     <xsd:attribute name="itemNumber" type="SKU" use="required"/>
```

What is an XML schema?

- stockItemType is defined by sequence (lines 04 - 12):
  - optional itemName element of type string (line 05)
  - required quantityOnHand with constrained range of 1 - 99 based on the type nonNegativeInteger (lines 06 - 11)

```
03.  <xsd:complexType name="stockItemType">
04.  <xsd:sequence>
05.     <xsd:element name="itemName" type="xsd:string" minOccurs="0"/>
06.     <xsd:element name="quantityOnHand">
07.        <xsd:simpleType>
08.           <xsd:restriction base="xsd:nonNegativeInteger">
09.             <xsd:maxExclusive value="100"/>   </xsd:restriction>
10.        </xsd:simpleType>
11.     </xsd:element>
12.  </xsd:sequence>
```

What is an XML schema?

- Type declarations can be inline and unnamed, as in lines 07 - 10, which include the maxExclusive facet to specify the legal values for the quantityOnHand element.

```
07.     <xsd:simpleType>
08.       <xsd:restriction base="xsd:nonNegativeInteger">
09.         <xsd:maxExclusive value="100"/>     </xsd:restriction>
10.     </xsd:simpleType>
```

What is an XML schema?

- For the itemNumber attribute, by contrast, the named type SKU is declared separately in lines 15 - 18, which include a pattern facet that uses regular expression syntax to specify that the legal values for that type consist of (in order):
  - 3 digits,
  - a hyphen-minus,
  - then two uppercase letters.

```
15.  <xsd:simpleType name="SKU">
16.    <xsd:restriction base="xsd:string">
17.      <xsd:pattern value="\d{3}-[A-Z]{2}"/>     </xsd:restriction>
18.  </xsd:simpleType>
```

What is an XML schema?

```
<stockItem itemNumber="453-SR">
<itemName>Stainless steel rope thimbles</itemName>
<quantityOnHand>23</quantityOnHand>
</stockItem>
```

- Root element is stockItem with attribute itemNumber
- itemNumber has 3 digits, a hyphen and 2 uppercase letters
- itemName contains a string
- quantityOnHand contains nonnegative integer less than 100

XML PARSE with validation

- How are schemas presented to the parser?
- COBOL validating parse uses XML System Services
- XML System services uses OSR…what?

XML PARSE with validation

- In Enterprise COBOL, a schema used for XML validation must be in a preprocessed format known as *Optimized Schema Representation*, or *OSR*.
- To generate a schema in OSR format from a text-form schema, use the z/OS UNIX command xsdosrg, which invokes the OSR generator provided by z/OS XML System Services.
- Example: to convert the text-form schema in file item.xsd to a schema in preprocessed format in file item.osr, you can use the following z/OS UNIX command:

  xsdosrg -v -o /u/*HLQ*/xml/item.osr /u/*HLQ*/xml/item.xsd

XML PARSE with validation

- Use one of two forms of the VALIDATING phrase, depending on the location of the preprocessed schema:
  - In one form, you use the FILE keyword and specify an XML schema name. In this case, the schema must be in an MVS data set or a z/OS UNIX file.
  - In the other form, you specify the identifier of a data item that contains the schema.

XML PARSE with validation

```
XML PARSE xmldoc
      VALIDATING WITH FILE mySchema
      PROCESSING PROCEDURE p1
END-XML
```

If you use the FILE keyword and specify an XML schema name, the COBOL runtime library automatically retrieves the schema during execution of the XML PARSE statement.

XML PARSE with validation

- To associate an XML schema name with the external file that contains the schema, code the XML-SCHEMA clause in the SPECIAL-NAMES paragraph, specifying either a literal or a user-defined word to identify the file.
- For example, you can associate the XML schema name mySchema with the ddname DDSCHEMA by coding the ddname as a literal in the XML-SCHEMA clause as follows:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    XML-SCHEMA mySchema IS 'DDSCHEMA'.
```

XML PARSE with validation

- For running the program, you can associate ddname DDSCHEMA with the z/OS UNIX file item.osr by coding a DD statement as follows:

  //GO.DDSCHEMA DD PATH='/u/*HLQ*/xml/item.osr‘

- Or you can use an analogous TSO ALLOCATE command.
- Alternatively, DDSCHEMA could be the name of an environment variable that identifies the external file by means of
  - a DSN option that specifies an MVS data set or
  - a PATH option that specifies a z/OS UNIX file

XML PARSE with validation

- VALIDATING WITH identifier
  - Requires schema in memory before XML PARSE statement
  - Use COBOL READ, EXEC CICS READ or SQL SELECT to get schema into memory
  - Supported under CICS
  - Could be efficient to read the schema into memory once and reuse it for subsequent XML documents
  - VALIDATING WITH FILE
    - Reads schema into memory for every parse: convenient, but slow
    - Not supported under CICS

XML PARSE with validation

```
XML PARSE xmldoc
      VALIDATING WITH mySchema-id
      PROCESSING PROCEDURE p1
END-XML
```

If you do not use the FILE keyword and specify an XML schema identifier, the schema must be in the specified data item during execution of the XML PARSE statement.

XML PARSE with validation

```
VParse section.
    Perform Get-schema
    Display "Completed reading schema into memory"
    Perform Validate-XML
    Display "Completed validating PARSE"
    If document-valid
      Perform Process-data-from-XML
    End-if.
```

XML PARSE with validation

```
Data division.
File section.
FD  F2
     block contains 0
     recording mode is F.
   1 R2              Pic x(80).
Local-storage section.
   01 fs2            Pic 99.
   01 DOCL           Pic 9(9) BINARY.
   01 mySchema-id    Pic X(1000000).
```

XML PARSE with validation


```
Get-schema section.
    Open input F2
    If FS2 not = 0 then  Display 'OPEN failed,'
       'FS=' FS2   Goback  End-if
    Set Not-EOF To TRUE
    Compute DocL = 1
    Perform Until EOF
      Read F2
        At End   Set EOF To TRUE  Display "EOF on F2"
        Not At End
           Display "Read:   " R2(1:80)
           Move R2(1:80) To mySchema-id(DocL:80)
           Add 80 To DocL
           End-If
        End-Read
    End-Perform
    Close F2.
```

XML PARSE with validation

```
Validate-XML section.
    Display "Reading and parsing XML ..."
    Open input F1
    Read F1
    Display "Read: " R1(1:L)
    XML parse R1(1:L)
      VALIDATING WITH myschema-id
      with encoding 1047
      processing procedure P
    End-XML
    Close F1.
```

XML PARSE (V4R1 review)

```
*******************************
* XML in file
*******************************
 FD  F1
     record is varying from 1 to 32767
               depending on L
     recording mode is V.
 01 R1 pic x(32767).
 Working-storage section.
   01 L              Pic 9(9) BINARY.
```

XML PARSE (V4R1 review)

```
P. If XML-Code not = zero
      Display "Error: unexpected parse code: "
                    Hex-code
      Perform Display-XML-registers
      Add 1 to ec
   Else
      Perform Display-XML-registers
      Continue
   End-if
   If XML-Event = "END-OF-INPUT"
      Read F1
        At end
          Move zero to XML-Code
        Not at end
          Move 1 to XML-Code
          Display "Next segment: " R1(1:L)
      End-read
   End-if
```

XML PARSE with validation

- Any questions on XML PARSE with validation?

Performance improvements to parsing XML documents without validation

- In Enterprise COBOL V4R1, some users found that XML PARSE was slow with XMLPARSE(XMLSS)
- Typical setup was
  - STORAGE(00)
  - ANYHEAP(FREE)
- Solution
  - STORAGE(NONE) or
  - ANYHEAP(KEEP)
- In Enterprise COBOL V4R2 this problem is solved
  - You get the better performance with any settings

Note: XMLPARSE(COMPAT) is faster, much less function

Compiler message severity customization

- New MSGEXIT suboption of EXIT compiler option
- Exit called for each compiler message
- Sample message exit IGYMSGXT included in SIGYSAMP
  - IGYMSGXT is explained in the Programming Guide
- User can change severity of diagnostic messages, convert FIPS messages to diagnostic, or suppress messages

Compiler message severity customization

- You can change the severity of a compiler message in the following ways:
  - Severity-I and severity-W compiler diagnostic messages, and FIPS messages, can be changed to have any severity from I through S
  - Assigning a severity to a FIPS message converts the FIPS message to a diagnostic message of the assigned severity

Compiler message severity customization

- As examples, you can:
  - Suppress optimizer warnings
  - Disallow REDEFINING a smaller item with a larger item by raising the severity of message 1154
  - Prevent inclusion of TEST information in the object file when the SYSDEBUG file cannot be opened, by raising the severity of message 4073
  - Disallow complex OCCURS DEPENDING ON by changing FIPS message 8235 from a category-E FIPS message to a severity-S compiler diagnostic message.

Compiler message severity customization

- Severity-E messages can be raised to severity S, but not lowered to severity I or W, because an error condition has occurred in the program
- Severity-S and severity-U messages cannot be changed to have a different severity
- You can request suppression of compiler messages as follows:
  - I, W, and FIPS messages can be suppressed
  - E and S messages cannot be suppressed
  - Suppressed messages do not affect compilation return code

Message severity customization examples

```
 Evaluate EXIT-MESSAGE-NUM
*****************************************************************
* Change severity of message 1154(W) to 12 ('S')
* This is the case of redefining a large item
* with a smaller item, IBM Req # MR0904063236
*****************************************************************
   When(1154)
     Compute EXIT-USER-SEV = 12
*****************************************************************
* Suppress all optimizer warning messages:
* 3090, 3091, 3094, 3171 and 3235
* This is IBM Req # MR00049476
*****************************************************************
   When(3090) When(3091) When(3094) When(3171)  When(3235)
     Compute EXIT-USER-SEV = -1
```

Message severity customization examples

```
*************************************************************
* Change severity of messages 3188(W) and 3189(W) to 12 (‘S’)
* This is to force a fix for all SEARCH ALL cases that might
* behave differently between COBOL compilers earlier than
* Enterprise COBOL release V3R4 and later compilers such as
* Enterprise COBOL Version 4 Release 2.
*************************************************************
   When(3188) When(3189)
     Compute EXIT-USER-SEV = 12
*************************************************************
* Change severity of message 4073(W) to 12 ('S')
* Prevent inclusion of TEST information in object
* if SYSDEBUG file cannot be opened.
* This is IBM Req # MR0716082134
*************************************************************
   When(4073)
     Compute EXIT-USER-SEV = 12
```

Message severity customization example

```
Options in effect:
   EXIT(NOINEXIT,NOPRTEXIT,NOLIBEXIT,NOADEXIT,MSGEXIT(MSGXRTN))
   FLAG(S,S)
  .
  .
  .
FIPS Messages      Total      Standard      Nonstandard      Obsolete
                     6            0               6              0
Messages   Total  Informational   Warning   Error   Severe   Terminating
Suppressed:    2           1                  1
Messages suppressed by MSGEXIT: 2
Messages with severity modified by MSGEXIT: 2
* Statistics for COBOL program MSGXT2:
*    Source records = 61
*    Data Division statements = 14
*    Procedure Division statements = 3
End of compilation 1,  program MSGXT2,  highest severity 8.
```

Compiler option BLOCK0 to exploit system-determined block size for QSAM output files

- Prior to this option, the default for files with no BLOCK CONTAINS clause was unblocked
  - One physical write for each logical write!
- BLOCK0 changes the default to blocked with no block size specified
  - BLOCK0 activates an implicit BLOCK CONTAINS 0 clause for all eligible files
- Eligible files are
  - QSAM (physical sequential)
  - No BLOCK CONTAINS clause
  - No RECORDING MODE U

Compiler option BLOCK0 to exploit system-determined block size for QSAM output files

- For INPUT files there might be problems in odd cases. Specifying BLOCK0 for existing programs might result in a change of behavior, and in some cases produce undesirable results for files opened as INPUT. For example:
  - The OPEN INPUT statement fails for files for which no block size can be determined.
  - Programs that continue after handling nonzero FILE STATUS codes for files opened as INPUT might abnormally terminate when executing subsequent I/O statements on those files
- For OUTPUT files, BLOCK0 should only improve performance, no risk

COBOL user-defined words can include the underscore (_) character

- This includes data names and procedure names but also…
- Program names!
- Why?
  - You can use XML GENERATE to create XML with underscore in tag names - no post processing!
  - You can use variables with the same names as column names in DB2/SQL
  - You can share variable names across COBOL, PL/I, C/C++ and assembler
  - You can call C functions with underscores in names without using PGMNAME(LONGMIXED)

COBOL user-defined words can include the underscore (_) character

- COBOL words cannot start with underscore, but can end with underscore
- Literal program names can have underscore anywhere

```
PROGRAM-ID. my_func.
LINKAGE SECTION.
   77 x_ PIC X(8).
   77 y_y PIC 99.
   01 Z_ PIC Z9.
PROCEDURE DIVISION USING x_  y_y  Z_.
    CALL '_cFunc'
```

Compiler listings display CICS options in effect

- Using the integrated CICS translator
- CICS 3.2 or CICS 4.1 with PTFs
- Now you can get DB2 and CICS options in effect in your COBOL listing!

Compiler listings display CICS options in effect


```
PP 5655-S71 IBM Enterprise COBOL for z/OS  4.2.0
Options in effect:
 NOADATA
   ADV
   QUOTE
   ARITH(COMPAT)
 NOAWO
 NOBLOCK0
   BUFSIZE(4096)
   .
   .
   .
CICS Options in effect:
 CICS
 DEBUG
 EDF
 NATLANG(EN)
 APOST
 NOSYSEIB
 NOFEPI
 NOCPSM
 LINKAGE
```

Java interoperability – support for Java 5 and Java 6

- Java 5 was not supported with Enterprise COBOL V4R1
- Java 5 and Java 6 are supported with Enterprise COBOL V4R2
- Enterprise COBOL applications using object-oriented syntax for Java interoperability can now run with Java 5 or Java 6. Java SDK 1.4.2 continues to be supported
- COBOL requires a 31-bit Java SDK. 64-bit Java technology is not currently supported

Requirements satisfied by COBOL V4R2

- MR1216031446 - underscores for var name across COBOL, PL/I, C/C++ and HLASM.
- MR0317083231, MR120804208 - underscore XML GENERATE.
- MR071508264 - underscores in names so COBOL names can match DB2/SQL names
- MR0716082134(SYSDEBUG) MR00023071 (Obsolete FIPS) – satisfied by message severity customization
- MR00049476, MR0227075935, MR042506463, MR0524065843, MR072006118, MR0803072215, MR0904063236, MR110606344 - message severity customization
- MR0318042758, MR1126031815 – CICS options in effect in listing
- MR0529095034, MR1216031447 (SSLNGC03003), MR1107071627 – BLOCK0 compiler option

Prerequisites

- z/OS, V1.9 (5694-A01), or later
- z/OS Language Environment V1.9, V1.10, or V1.11 with PTFs for APAR PK90754 installed
- Enterprise COBOL XML processing with the XMLPARSE(XMLSS) option requires:
  - z/OS XML Systems Services V1.9, V1.10, or V1.11, and PTFs for APAR OA28253
  - When parsing with validation under CICS, the PTFs for APAR OA29675 are also required

Prerequisites

- Including CICS options in effect as part of the COBOL listing requires
  - CICS Transaction Server for z/OS, V4.1 (5655-S97) and PTFs for APAR PK89224, or
  - CICS Transaction Server for z/OS, V3.2 (5655-M15) and PTFs for APAR PK91041
- Enterprise COBOL applications using object-oriented syntax for Java interoperability with Java 6 require
  - SDK for z/OS, Java Technology Edition V6 (5655-R31) and PTFs for APAR PK89762

Prerequisites

- CICS Transaction Server for z/OS, V3
- DB2 Universal Database for z/OS, V8
- IMS (TM & DB), V9
- Debug Tool for z/OS, V7
- IBM Fault Analyzer for z/OS, V7
- IBM Application Performance Analyzer for z/OS, V7