Migrating to Enterprise COBOL V6

Mike Chase
June 5th, 2018

Migration to Enterprise COBOL V6

What, when, and why of COBOL Migration

A brief history of COBOL compilers on z/OS

What is different about COBOL V6 migration

Best practices for COBOL V6 Migration

Examples of invalid data

How to prepare for COBOL V6 before you buy

Resources

What, when, and why of COBOL Migration

What

- Enterprise COBOL for z/OS, V5 and Enterprise COBOL for z/OS, V6
- COBOL compilers with new generation code generator and optimizer

When

- COBOL V5.1: 2013, V5.2: 2015
  - COBOL V5 EOM Sept 11, 2017 (announced Dec 6, 2016)
- COBOL V6.1: 2016, V6.2: 2017
  - Migrating to V6 is the same as migrating to V5, we will only say V6 in this talk

Why

- Exploit the latest hardware
- Give performance improvement to COBOL applications without source code changes
- Less MSUs, save money!

What, when, and why of COBOL Migration

How to save MSUs?

- Migrate to (recompile with) Enterprise COBOL for z/OS, V6

What's different than previous migrations over the last 30 years?

- New code generator could produce more optimal code than prior versions of COBOL.
- You can get different generated code sequences for the same COBOL source.
  - **Good**: Save MSUs (MIPS, CPU)
  - **Not so good**: More optimal instructions can process invalid data differently, causing different runtime behavior

Optimization of COBOL programs

Can IBM improve performance of older COBOL applications without recompiling?

- Yes! Automatic Binary Optimizer (ABO) optimizes the executable (either Load Module or Program Object), without using the source
- ABO uses the same technology as Enterprise COBOL V6
- Useful for programs...
  - With missing source code
  - That aren't being actively deployed
  - That must run out of PDS datasets
  - That must call or be called by OS/VS COBOL programs

Which tool to use to optimize your COBOL?

- To get the best performance, recompile from source with Enterprise COBOL V6 or later
- If you cannot use the newer compilers for some of the reasons above, and your programs were previously compiled with VS COBOL II thru V4.2, use Automatic Binary Optimizer (ABO)

Migration to Enterprise COBOL V6

What, when, and why of COBOL Migration

A brief history of COBOL compilers on z/OS

What is different about COBOL V6 migration

Best practices for COBOL V6 Migration

Examples of invalid data

How to prepare for COBOL V6 before you buy

Resources

A brief history of COBOL compilers on z/OS

Compiler Terminology

- Front End
  - Parser and syntax checker
  - Builds dictionary of data items
  - Creates internal representation of COBOL statements
- Back End
  - Optimizer
  - Generates machine code
  - Allocates/manages machine registers
  - Produces object program and DWARF debugging info

A brief history of COBOL compilers on z/OS

Compiler	Front End	Back End
OS/VS COBOL	74 Std	1st Generation
VS COBOL II	85 Std (new)	2nd Generation (new)
COBOL/370	85 Std (same)	2nd Generation (same)
COBOL for OS/390 V2	85 Std (same)	2nd Generation (same)
COBOL for z/OS V3	85 Std (same)	2nd Generation (same)
COBOL for z/OS V4	85 Std (same)	2nd Generation (same)
COBOL for z/OS V5	85 Std (same)	3rd Generation (new)
COBOL for z/OS V6	85 Std (same)	3rd Generation (same)

Extra care is needed when crossing the red lines

COBOL Migration History

OS/VS COBOL to all newer versions

- Most difficult migration
  - Source incompatibilities between 1974 COBOL Standard and 1985 COBOL Standard
    - Convert source with CCCA (Included with IBM Debug Tool)
  - New code generator with more accurate numeric results
  - One COBOL V5 client rated migrating from OS/VS COBOL to a newer version as a difficulty of 100

VS COBOL II thru COBOL V2 with CMPR2 compiler option

- Some source incompatibilities between 1974 COBOL Standard and 1985 COBOL Standard
- Easier than OS/VS COBOL migration, but still need to convert source using CCCA

COBOL Migration History

VS COBOL II or later to Enterprise COBOL V4 or earlier

- Very easy migration
  - Source is compatible
  - Generated code is the same between versions
    - Even programs using invalid data will behave the same
  - One COBOL V5 client rated migrating from COBOL V3 to V4 as a difficulty of 3

COBOL Migration History

VS COBOL II thru Enterprise COBOL V4 to Enterprise COBOL V6

- Medium difficulty migration
  - Source is compatible
    - COBOL V5 and V6 add a few new reserved words
    - Some rarely used language features are unsupported
    - 99.9% of programs will compile with Enterprise COBOL V6 without changes
  - New code generator produces new machine code
    - Same results for valid data
    - When used with invalid data, the new machine code can have different results from previous compilers
  - One COBOL V5 client rated migrating to COBOL V5 as a difficulty of 20

Migration to Enterprise COBOL V6

What's Different About COBOL V6?

Compile time differences

- About 20x more memory required at compile time
- More time required to compile a program
  - 5x to 12x, depending on optimization level
- More compiler work datasets (SYSUTx) required
  - Use new IBM-supplied compile PROCs
- Compiler messages are not in the same part of the listing as before
  - FE messages are in the middle, before pseudo-assembler
  - BE messages are at the end like in COBOL V4 and earlier
  - Previous behavior restored in V6.2!
- Compiler always uses some above the 2GB bar storage, so MEMLIMIT must be set to non zero value
- Compiler required an OMVS segment be defined for the userid doing the compilation. This requirement is removed with APAR PI94326.

What’s Different About COBOL V6?

Run time differences

- Executables must be in PDSE datasets
- COBOL V6 programs cannot call or be called by OS/VS COBOL programs

What’s Different About COBOL V6?

Bind time differences

- Old IGZEBSTs (bootstrap/initialization routines) can cause problems for VS COBOL II programs mixed with COBOL V6
  - Link edit/bind time correction
  - Will need effort to update VS COBOL II load libraries called dynamically if the programs in them aren't being recompiled
- AMODE 24: There used to be problems, but IBM fixed them in March 2014 (V5.1.1). COBOL V5 and V6 support AMODE 24 is mostly the same as COBOL V4 with some bind-time differences:
  - When a program object contains any of the following programs, the binder option RMODE(24) must be specified:
    - An Enterprise COBOL program that is compiled with the RMODE(24) or NORENT compiler options.
    - A VS COBOL II program that is compiled with the NORENT option.
    - An assembler program that contains a CSECT with RMODE 24.
    - COBOL pre-V5 programs that run with AMODE 24 and statically call a COBOL V5+ program.

Invalid Data in COBOL V6

- About 25% of customers migrating to COBOL V6 encounter migration problems as a result of COBOL programs processing invalid data at run time

My program worked before! What changed in COBOL V6?

- Different generated instructions can process invalid data differently from programs produced by previous compilers
  - Not a problem for valid data

Why doesn’t the compiler give error diagnostics for invalid data?

- We will describe several cases, but in general it is data values at run time or inter-program dependencies, neither of which can be found by a compiler

COBOL V6 Migration: How did we get here?

Why didn't IBM enforce rules against invalid data for the past 30 years?

- IBM does not test invalid data in general
  - We had no idea of the level of 'misuse' of COBOL by customers
    - Previous code generator hid many problems
- The COBOL Standard provided solutions for invalid numeric data
  - i.e. IF NUMERIC
- IBM provided solutions for invalid table processing
  - i.e. SSRANGE

COBOL V6 Migration: How did we get here?

The COBOL V6 migration issues caused by invalid data or parameter passing are:

- Invalid data in numeric USAGE DISPLAY data items
- Parameter/argument size mismatch
- Users of TRUNC(OPT) or TRUNC(STD) with overpopulated binary data items (values with more digits than are defined in the data definitions)
- Data items that are used before they're assigned a value

All other known issues with invalid data causing differences in behavior between compilers have been resolved in PTFs

Note: Make sure that all PTFs are applied to your compiler when you first install it, and consider frequent updates via PTF for performance and new features! Only installing z/OS RSU service is also a good way to go.

Migration to Enterprise COBOL V6

Best Practices for COBOL V6 Migration

How much MSU reduction do you get with Enterprise COBOL V6?

- Depends on many factors, the only way to know is to measure performance before and after migration

IBM recommends this process for performance comparison:

- Back up V4 (or earlier) executables before you migrate
- After migrating, set up a test environment with a real, representative workload, and measure performance against that workload with the old V4 executables and again with the new V6 executables

Measuring in production won't be as accurate

- Different workloads at different times

Not best practice to measure V4 before migrating and V6 after

- Hardware, workloads, code, may all be changed during migration

Best Practices for COBOL V6 Migration

To find out if users have invalid data, IBM has recommendations for migrating to COBOL V6. The first time that you compile a program:

1. Compile with SSRANGE, NUMCHECK,PARMCHECK and OPT(0) for initial code changes and unit test
   - To find table misuse, invalid data use and invalid parameter usage
   - OPT(0) programs are easiest to debug, quicker compiles
   - Look at runtime logs for NUMCHECK, etc, error messages
2. Recompile with NOSSRANGE, NONUMCHECK, NOPARMCHECK and OPT(2) plus INITCHECK for quality assurance test and production
   - NOSSRANGE, NONUMCHECK and NOPARMCHECK are required for good performance
   - OPT(2) is preferred for good performance in production
   - Inspect listings for INITCHECK messages

- Note: You may have to change to a 2-compile development process if you are not using one already

Best Practices for COBOL V6 Migration

To help reduce cases of invalid data IBM has these recommendations for COBOL development

- We recommend using the RULES compiler option to give developers information about their programs, things like:
  - NOENDPERIOD (flags conditional statements terminated with period)
  - NOEVENPACK (flags even number of packed decimal digits)
  - NOLAXPERF (flags opportunities for performance improvements)
  - NOSLACKBYTES (flags bytes added by compiler for SYNCHRONIZED data items)
- We recommend always using DIAGTRUNC
  - To find any cases of ‘hidden’ loss of data when statements truncate numeric data items
- Use the **Scanning COBOL Programs for Compatibility** feature of IDz (introduced in RDz 9.5) to check parameters
  - To find parameter mismatches in CALL statements

Best Practices for COBOL V6 Migration

A few things to consider about compiler options

- Be aware of ARCH setting and your hardware. You need to know the lowest level of hardware where your programs will ever be run (Disaster recovery machine? Subsidiary companies?)
  - EG: ARCH(11) programs will abend with an 0C1 on zEC12 (or earlier)
  - If you update your hardware in the future you will want to update ARCH in your COBOL compile steps as well
- NUMPROC(MIG) is removed, which requires special consideration and extra testing.
  - Usually use NUMPROC(NOPFD)
  - Also look at ZONEDATA(MIG)
  - Using NUMCHECK(ZON,PAC) with NUMPROC(PFD) can indicate that your data and signs are always preferred, allowing you to migrate to NUMPROC(PFD) and ZONEDATA(PFD) for better performance
- Set the other options in COBOL V6 to the same values that you used in COBOL V4 and earlier

Migration to Enterprise COBOL V6

Invalid Data in Numeric USAGE DISPLAY data items

```
77 A1 PIC X(4) VALUE '00 0'. *> x'F0F040F0', third byte
                              *> has x'4' for zone bits.
                              *> OK in PIC X, not valid in
77 A2 REDEFINES A1 PIC 9(4). *> PIC 9 USAGE DISPLAY

PROCEDURE DIVISION.
   IF A2 = ZERO               *> Compiler could do character
      DISPLAY 'ZERO'          *> or numeric compare
   ELSE
      DISPLAY 'NOT ZERO'
   END-IF
```

- Whether the program displays 'ZERO' or 'NOT ZERO' depends on the compiler options you use in COBOL V4 and earlier and in COBOL V6
- Character compare would be not equal, numeric compare would remove zone bits and compare equal

Invalid Data in Numeric USAGE DISPLAY data items

How to identify

- Add IF NUMERIC checks to your code
- Compile and test with the NUMCHECK(ZON) or ZONECHECK compiler options, to get a message or abend when a USAGE DISPLAY data item is invalid
  - ZONECHECK introduced in COBOL V6.1 GA, COBOL V5.2 May 2015 PTFs, COBOL V5.1 June 2015 PTFs, and COBOL V4.2 PTF UI32232 (October 2015)
  - NUMCHECK(ZON) is preferred over ZONECHECK;
    - Added in COBOL V6.2 GA
    - Added in COBOL V6.1 February 2017 PTFs
    - Added in COBOL V5.2 May 2017 PTFs
    - ZONECHECK is still tolerated as NUMCHECK(ZON)

Invalid Data in Numeric USAGE DISPLAY data items

How to correct

- Use NUMCHECK(ZON) or ZONECHECK to find the source of invalid data, and correct at the source
  - Invalid value explicitly set through code (e.g. REDEFINE): correct it
  - Incorrect record description for file, use the correct one
  - Group MOVEs, correct mismatch or use MOVE CORRESPONDING
  - Value coming from another source: correct at the source or add IF NUMERIC test to validate before use

How to tolerate bad data if you can't fix it

- Use ZONEDATA to cause the compiler to generate V4-compatible code

Invalid Data in Numeric USAGE DISPLAY data items

What ZONEDATA and NUMPROC options should I use?

VALID data?	V4 NUMPROC	V6 NUMPROC	V6 ZONEDATA
Yes	NUMPROC(MIG)	NUMPROC(NOPFD)	ZONEDATA(PFD)
Yes	NUMPROC(NOPFD)	NUMPROC(NOPFD)	ZONEDATA(PFD)
Yes	NUMPROC(PFD)	NUMPROC(PFD)	ZONEDATA(PFD)
No	NUMPROC(MIG)	NUMPROC(NOPFD)	ZONEDATA(MIG)
No	NUMPROC(NOPFD)	NUMPROC(NOPFD)	ZONEDATA(NOPFD)
No	NUMPROC(PFD)	NUMPROC(PFD)	ZONEDATA(NOPFD)

Invalid Data in Numeric USAGE DISPLAY data items

```
77 A1 PIC X(4) VALUE ’00 0’. *> x’F0F040F0’, third byte
                             *> has x’4’ for zone bits.
                             *> OK in PIC X, not valid in
77 A2 REDEFINES A1 PIC 9(4). *> PIC 9 USAGE DISPLAY

PROCEDURE DIVISION.
   IF A2 = ZERO              *> Compiler could do character
      DISPLAY ’ZERO‘         *> or numeric compare
   ELSE
      DISPLAY ’NOT ZERO‘
   END-IF
```

```
IGZ0279W The value X'F0F040F0' of data item A2 at the
 time of reference by statement number 1 on line 8 in
 program ZONE failed the NUMERIC class test generated
 by the NUMCHECK compiler option.
```

Parameter/Argument Size Mismatch

```
 77   GRP1 PIC X(100).
Procedure Division.
. . .
   Call ‘SUBP’ Using GRP1.

Program-Id. SUBP.
Linkage Section.
   01  GRP2 PIC X(500).
Procedure Division Using GRP2.
   MOVE ‘stuff’ To GRP2(300:20) *> Invalid!
```

Note: caller is passing fewer bytes than the called program uses

Results

- For V2, V3, V4: illegal program didn’t fail
- For V6: file-status in CALLER changed; flow changed, failed
- NOTE: To catch this error, PARMCHECK(*,400) or greater is needed

Parameter/Argument Size Mismatch

How to identify

- Compile with new PARMCHECK compiler option and run regression tests
  - PARMCHECK available in V6.1 in April 2017 PTF and V6.2 GA
- New feature of IBM Developer for z Systems (initially in RDz 9.5)
  - Scanning COBOL programs for compatibility
    - Use the **Scanning COBOL Programs for Compatibility** feature to scan a set of COBOL programs to determine whether the parameters passed between the calling and called programs are compatible
    - **This works for CALL ‘literal’ statements and also for most CALL data-name statements**

How to correct

- Change the source code so the calling program is passing parameters at least as large as the called program expects

Parameter/Argument Size Mismatch

```
PROCESS PARMCHECK(MSG,500)


77   GRP1 PIC X(100).
Procedure Division.
. . .
 Call 'SUBP' Using GRP1.

Program-Id. SUBP.
Linkage Section.
 01  GRP2 PIC X(500).
Procedure Division Using GRP2.
 MOVE 'stuff' To GRP2(300:20) *> Illegal!
```

```
IGZ0318W The CALL statement on line 135 in program TESTRUN
caused corruption of data beyond the end of the WORKING-
STORAGE SECTION.
```

Overpopulated binary data items with values that have more digits than are defined in the data definitions

```
01 A1 PIC X(2).
01 A2 REDEFINES A1 PIC 9(3) BINARY.  *> 3 digits
01 B PIC 9(2) VALUE 2.
01 C PIC 9(3).

MOVE x‘FFFF' TO A1          *> A2 = 65535: 5 digits!
COMPUTE C = A2 * B
DISPLAY C
```

- This is valid for programs compiled with TRUNC(BIN) and invalid for programs compiled with TRUNC(STD) and TRUNC(OPT)
  - Displays 070 with V6 “TRUNC(any)”, V4 “TRUNC(BIN)”
  - Displays 002 with V4 “TRUNC(STD) or TRUNC(OPT)”

- TRUNC(OPT) is different in COBOL V6 than previous compilers
  - No difference if rules followed, i.e.:

    Use the TRUNC(OPT) option only if you are sure that the data being moved into the binary areas will not have a value with larger precision than that defined by the PICTURE clause for the binary item.

Overpopulated binary data items with values that have more digits than are defined in the data definitions

How to identify

- Compile and test with the NUMCHECK(BIN) compiler option, to get a message or abend when a BINARY data item has a value that exceeds its picture clause

How to correct

- Depends on the context
  - Incorrect data item description, increase number of digits or use USAGE COMP-5
  - Invalid value explicitly set through code (e.g. REDEFINES): correct the code
  - Incorrect record description for file: use the correct one
  - Value coming from another source: correct at the source or add code to force a truncation

Overpopulated binary data items with values that have more digits than are defined in the data definitions

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. BIN.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 A1 PIC X(2).
 01 A2 REDEFINES A1 PIC 9(3) BINARY.
 01 B PIC 9(2) VALUE 2.
 01 C PIC 9(3).
 PROCEDURE DIVISION.
MOVE x‘FFFF' TO A1
COMPUTE C = A2 * B
DISPLAY C
```

```
IGZ0316W The value X'FFFF' of data item A2 at the time
 of reference by statement number 1 on line 11 in
 program BIN was invalid.  The value exceeded the
 number of digits in the data definition, and failed
 the SIZE ERROR test generated by the NUMCHECK(BIN)
 compiler option.
```

Data items that are used before being given a value

```
01 X PIC X(100).
01 Y PIC 9(5).
01 Z PIC 9(3) BINARY.
01 W PIC 9(3) BINARY.

DISPLAY "X: " X
IF Y > 100
  COMPUTE W = Z + 1
END-IF
```

- What values do X, Y, and Z have at runtime?
  - Depends on runtime options, how the compiler has laid out memory, where the program was loaded
  - Uninitialized memory isn't guaranteed to have any specific value
  - COBOL V6 cannot guarantee uninitialized memory has the same value as it did in COBOL V4

Data items that are used before being given a value

How to identify

- Compile with INITCHECK compiler option, introduced in Sept. 2016 PTF for V6.1, and V6.2 GA
  - Requires OPT(1) or OPT(2); compiler does advanced analysis not possible at OPT(0)
  - Warnings are given at compile time

How to correct

- Assign a value to the data item (MOVE, INITIALIZE, or use a VALUE clause) before using it as a sender

Data items that are used before being given a value

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. INIT.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 X PIC X(100).
 01 Y PIC 9(5).
 01 Z PIC 9(3) BINARY.
 01 W PIC 9(3) BINARY.
 PROCEDURE DIVISION.
DISPLAY "X: " X
IF Y > 100
 COMPUTE W = Z + 1
END-IF
GOBACK.
```

```
10  IGYCB7311-W   The data item 'X' may be used at this
statement before it is set.

11  IGYCB7311-W   The data item 'Y' may be used at this
 statement before it is set.

12  IGYCB7311-W   The data item 'Z' may be used at this
 statement before it is set.
```

Migration to Enterprise COBOL V6

COBOL V6: Before you buy

Install latest maintenance required for COBOL V6
(on LE, DB2, CICS, Binder, and other products)

- Use the COBOL FIXCAT feature documented here: http://www-01.ibm.com/support/docview.wss?uid=swg21648871
- Run the SMP/E MISSINGFIX command to find required PTFs (LE,DB2,CICS,Binder, etc) for the new compilers:

```
SET BDY(GLOBAL)
REPORT MISSINGFIX ZONES(ZOS13T,ZOS13P)
FIXCAT(IBM.TargetSystem-RequiredService.Enterprise-COBOL.V5R1,
       IBM.TargetSystem-RequiredService.Enterprise-COBOL.V5R2,
       IBM.TargetSystem-RequiredService.Enterprise-COBOL.V6R1,
       IBM.TargetSystem-RequiredService.Enterprise-COBOL.V6R2)
```

  – This command will look for all PTFs needed for COBOL V6
- Install indicated PTFs on **all systems** before using the new compiler

COBOL V6: Before you buy

- Convert PDS COBOL load libraries to PDSE datasets
- Locate all OS/VS COBOL programs and either target them for early migration to V6 or migrate them to V4
  - Get rid of the “OS/VS COBOL problem” early
- Change build processes in the BIND/LINK step to avoid using the old VS COBOL II bootstrap routines
  - REPLACE –IMMED,IGZEBST
  - This will not fix all, but is a no-risk change that could have a good reward
- **Use the same compiler options with COBOL V6 as in earlier compilers when migrating,** except:
  - Options that have been removed, e.g. NUMPROC(MIG)
  - Optimization level and ARCH level
  - Do not change from NUMPROC(NOPFD) to NUMPROC(PFD) or from TRUNC(BIN) to TRUNC(OPT) without doing research and testing

Migration to Enterprise COBOL V6

What, when, and why of COBOL Migration

A brief history of COBOL compilers on z/OS

What is different about COBOL V6 migration

Best practices for COBOL V6 Migration

Examples of invalid data

How to prepare for COBOL V6 before you buy

Resources

IBM Doc Buddy

IBM

IBM Doc Buddy

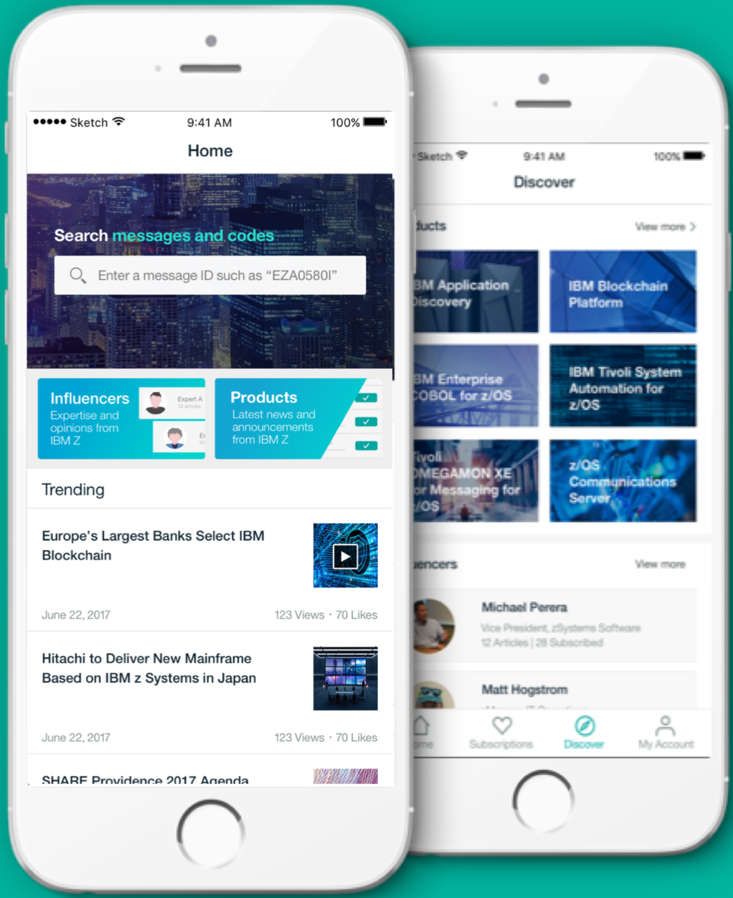
With the IBM Doc Buddy mobile app, you can search messages and codes issued from IBM Z products online and offline. IBM Doc Buddy also aggregates mainframe content including blogs, videos, IBM Knowledge Center topics, and Thought Leader opinions.

iOS

Android

https://ibmdocbuddy.mybluemix.net/

sptast@cn.ibm.com



Sketch
9:41 AM
100%
Home
Search messages and codes
Enter a message ID such as "EZA0580I"
Influencers
Expertise and opinions from IBM Z
Products
Latest news and announcements from IBM Z
Trending
Europe's Largest Banks Select IBM Blockchain
June 22, 2017
123 Views · 70 Likes
Hitachi to Deliver New Mainframe Based on IBM z Systems in Japan
June 22, 2017
123 Views · 70 Likes
SHARE Providence 2017 Agenda
Discover
View more
IBM Application Discovery
IBM Blockchain Platform
IBM Enterprise COBOL for z/OS
IBM Tivoli System Automation for z/OS
Tivoli OMEGAMON XE for Messaging for z/OS
z/OS Communications Server
Michael Perera
Vice President, zSystems Software
12 Articles | 28 Subscribed
Matt Hogstrom
Subscriptions
Discover
My Account

Resources

The COBOL Migration Assistant
https://cobol-migration-assistant.mybluemix.net/

Ease migration to the latest
IBM Enterprise COBOL
for z/OS compiler
Experience
IBM COBOL Migration Assistant for z/OS
Visit
https://cobol-migration-assistant.mybluemix.net
IBM.

COBOL Resources

Enterprise COBOL
- Product Page: http://www-03.ibm.com/software/products/en/entecoboforzos
- Documentation: http://www-01.ibm.com/support/docview.wss?uid=swg27036733
- Trial: http://www-03.ibm.com/software/products/en/enterprise-cobol-developer-trial-for-zos

RFE community: Request For Enhancement
- https://www.ibm.com/developerworks/rfe/?PROD_ID=698

COBOL Café Blogs
- https://www.ibm.com/developerworks/community/blogs/31c890c6-ace1-4eeb-af6b-5950f3a1a5d1/?lang=en

COBOL Café discussion Forum
- https://www.ibm.com/developerworks/community/forums/html/forum?id=11111111-0000-0000-0000-000000002281

COBOL Performance
- Whitepaper: COBOL Applications: Techniques to Make them Efficient
- Performance Tuning Guide

Automatic Binary Optimizer Resources

ABO

- Product Page: : http://www-03.ibm.com/software/products/en/z-compilers-optimizer
- Documentation: http://www-01.ibm.com/support/docview.wss?uid=swg27046990
- Trial: http://www-03.ibm.com/software/products/en/ibm-automatic-binary-optimizer-trial-for-zos

Questions?

Q&A

Thank You!

Backup

Parameter/Argument Size Mismatch

```
 77   GRP1 PIC X(100).
Procedure Division.
. . .
   Call 'SUBP' Using GRP1.

Program-Id. SUBP.
Linkage Section.
   01  GRP2 PIC X(1).
Working-Storage Section.
   01  N PIC 9(5) BINARY.
Procedure Division Using GRP2.
   MOVE 100 TO N.
   MOVE 'stuff' To GRP2(1:N)   *> Illegal by SSRANGE
```

Results

- For V4 and earlier: Moved 100 bytes; did not follow COBOL rules
- For V6: Moved 1 byte
  - Moves 100 bytes after March 2016 V5.2 PTF, April 2016 V5.1 PTF, or June 2016 V6.1 PTF

Modifying data outside the bounds of a table

```
01 MY-TABLE.
   05 TABLE-ROW OCCURS 100 TIMES INDEXED BY MY-INDEX.
      10 MY-ITEM PIC X(1).

   SET MY-INDEX TO 1
   PERFORM UNTIL DONE
     MOVE ‘Z’ To MY-ITEM(MY-INDEX)
     SET MY-INDEX UP BY 1
   END-PERFORM
```

- You may see different results with statements that modify data beyond the end of a table in COBOL V5 compared to previous compilers.
  - In V6, index-names are stored immediately after the table group, rather than being stored elsewhere in memory (TGT in V4)
  - Changed in July 2016 PTF for V5.2 and September 2016 PTF for V6.1
    - Index-names are now at the beginning of their section
  - These types of invalid programs can be detected with the SSRANGE compiler option.

Using tables when the ODO object value is not in legal range

```
01 OBJ PIC 9(5) BINARY.
01 MY-TABLE.
  02 T OCCURS 0 TO 1 TIMES DEPENDING ON OBJ.
    05 MY-FIELD PIC X(1).
01 OFLOW PIC X(500).

MOVE 300 TO OBJ. *> Legal if table is not referenced
MOVE ALL 'M' TO MY-TABLE. *> Illegal, OBJ not in range 0 TO 1
DISPLAY MY-TABLE
DISPLAY OFLOW
```

- Different results in different versions of COBOL
  - V2, V3, V4: Moved 300 bytes of ‘M’
  - V5: Moved 1 byte of ’M’ and 299 bytes of ‘other’
  - Moves 300 bytes of ‘M’ after applying March 2016 V5.2 PTF, April 2016 V5.1 PTF, or June 2016 V6.1 PTF
  - You can use SSRANGE to detect this problem