Migrating to Enterprise COBOL V6

COBOL migration to V6 is completely different from the way it used to be!

Tom Ross
SHARE Orlando
Feb 24, 2026

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

Examples of invalid data

Best practices for COBOL V6 Migration

How to prepare for COBOL V6 before you buy

Resources

When should you migrate to COBOL V6?

News

- EOM (End Of Marketing: announced)
  - COBOL V5 EOM Sept 11, 2017 (announced Dec 6, 2016)
  - COBOL V4 EOM March 12, 2018 (announced Dec 7, 2017)

- EOS (End Of Service: announced)
  - COBOL V5: 4/2020 (announced Feb 5, 2019)
  - COBOL V4: 4/2022 (announced May 5, 2020)

NOTE: COBOL applications compiled with old compilers are still supported at run time; they are not using the compiler, they are using LE, which is part of z/OS. If your z/OS is in service, your COBOL applications have service support for running in production!

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 V6.1: 2016, V6.2: 2017, V6.3 2019, V6.4 2022, V6.5 2025
  - Migrating to V6 is the same as migrating to V5, we will only say V6 in this talk

Why

- Exploit the latest hardware
- COBOL performance improvement without source code changes
- Less MSUs, save money!
- To take advantage of COBOL-Java interoperability if WCA4Z is used to refactor/convert parts of an application to Java

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.
- This could mean different generated code sequences for the same COBOL source, which is:
  - **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
  - Both target the latest z Systems
  - Both use the same optimizer and code generator
- Doesn’t require a compiler migration

- 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

Optimization of 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 previous reasons, and your programs were previously compiled with VS COBOL II thru V4.2, use Automatic Binary Optimizer (ABO)

- Some customers migrate by recompiling everything
  - Except maybe some old applications that aren't ever being modified
  - Migrating, cleaning up invalid data, and recompiling gives the best performance and best future compatibility
- Some customers recompile their hot modules and use the new compiler for active development, and leave the rest alone
  - ABO gives better performance for the remaining modules

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

Examples of invalid data

Best practices for COBOL V6 Migration

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
  - 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 black lines

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

Examples of invalid data

Best practices for COBOL V6 Migration

How to prepare for COBOL V6 before you buy

Resources

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 always uses some above the 2GB bar storage, so MEMLIMIT must be set to non zero value

Run time differences

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

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
- Users of TRUNC(OPT) or TRUNC(STD) with overpopulated binary data items (values with more digits than are defined in the data definitions)
- Parameter/argument size mismatch
- Data items that are used before they're assigned a value
- Statements that use a zero address to access memory

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

What, when, and why of COBOL Migration

A brief history of COBOL compilers on z/OS

What is different about COBOL V6 migration

Examples of invalid data

Best practices for COBOL V6 Migration

How to prepare for COBOL V6 before you buy

Resources

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 THEN          *> 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) compiler options to get a message or abend when a USAGE DISPLAY data item is invalid
  - NUMCHECK(ZON)
- NUMCHECK can also check PACKED-DECIMAL or BINARY data
  - PAC or BIN suboptions

Invalid Data in Numeric USAGE DISPLAY data items

How to correct

- Use NUMCHECK(ZON) to find the source of invalid data, and correct at the source
  - Invalid value explicitly set through code (e.g. REDEFINES): 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 INVDATA to cause the compiler to generate V4-compatible code

Invalid Data in Numeric USAGE DISPLAY data items

What INVDATA and NUMPROC options should I use?

IBM recommends that all users ensure that they are using valid data and that they use NOINVDATA. However, If you find that you are using invalid data at run time, and you do not have the resources to fix your programs or systems so that you are using valid data at runtime…

In V4, I Used…		In V6, I Should Use…	
ALL data VALID?	**V4 NUMPROC**	**V6 NUMPROC**	**V6 INVDATA**
Yes	NUMPROC(MIG)	NUMPROC(NOPFD)	NOINVDATA
Yes	NUMPROC(NOPFD)	NUMPROC(NOPFD)	NOINVDATA
Yes	NUMPROC(PFD)	NUMPROC(PFD)	NOINVDATA
No	NUMPROC(MIG)	NUMPROC(NOPFD)	INVDATA(FORCENUMCMP, NOCLEANSIGN)
No	NUMPROC(NOPFD)	NUMPROC(NOPFD)	INVDATA(NOFORCENUMCMP, CLEANSIGN))
No	NUMPROC(PFD)	NUMPROC(PFD)	INVDATA(NOFORCENUMCMP, CLEANSIGN)

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 THEN         *> 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.
```


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

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

PROCEDURE DIVISION.

     COMPUTE C = A2 * B     *> A2 = 65535: 5 digits!
     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)”

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

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

PROCEDURE DIVISION.

     COMPUTE C = A2 * B    *> A2 = 65535: 5 digits!
     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.
```


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.
```


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.
PROCEDURE DIVISION.
   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
  - On V6.1, or on V6.2 before the March 2019 PTF, requires OPT(1) or OPT(2)
  - With the March 2019 V6.2 PTF, enabled 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.
```


Data items that are used before being given a value

What we have learned and Recommendations

- Many use STORAGE(00) in LE so that all WORKING-STORAGE items have initial value of hex ‘00’. This worked for years but causes ABENDs with z14 (ARCH(12)) for uninitialized data items!
- Idea: Set INITCHECK and EXIT(MSGEXIT(MYEXIT)) as default compiler options and change the severity of the messages to ‘S’ (RC=12) to force developers to ALWAYS initialize data items before they are used!
- The latest sample message exit has the suggested change:

```
*****************************************************************
*   Change severity of message 7311(W) to 12 ('S')              *
*   This is the case of INITCHECK messages about                *
*   uninitialized data items                                    *
*****************************************************************
          When(7311)
             Compute EXIT-USER-SEV = 12
```

- In sample library SIGYSAMP(IGYMSGXT) , change name to MYEXIT, put in library in compiler concatenation!

Migration to z/OS 2.5 or later and COBOL programs with uninitialized pointer data items

IBM has received several reports of COBOL programs behaving differently (IE: failing, wrong results, etc) when moving to z/OS 2.5 or later

This happens when programs use pointers that were set to zero (or not set). These result in data items trying to access ‘low core’. There is no ABEND, but the contents of low core are different in z/OS 2.5 from earlier z/OS releases, so programs that do this can get different results.

Use of zero addresses to access memory

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BADPTR.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PTRX USAGE POINTER.
LINKAGE SECTION.
01 STUFF.
   05 OTHER-STUFF  PIC X Y Z etc . . .
   05 CUST-NAME PIC X(30).
PROCEDURE DIVISION.
   Set Address of STUFF To PTRX
   If CUST-NAME = x'00' Then
     Display 'This is z/OS before 2.5'
   Else
     Display 'This is z/OS 2.5 or later'
   END-IF
   GOBACK.
```

- The IF … ELSE path taken can change due to changes in z/OS when accessing 'low core', so that some programs get wrong results when moving to new z/OS levels. We have seen this with customers moving to z/OS 2.5 from previous z/OS levels.

Use of zero addresses to access memory

- How to find and fix uses of zero addresses?
  - INITCHECK could help find the use of uninitialized pointers, but would not find cases where a pointer is explicitly set to zero:

```
Set PTRX To NULL
Set Address of STUFF To PTRX
```

  - INITCHECK would also not find addresses that are passed into a program or that are retrieved from some other source.

- NOTE: This problem when migrating to newer levels of z/OS can happen with any level of COBOL!

Use of zero addresses – the solution!

- How to find and fix uses of zero addresses?
  - One way to find these instances of zero address usage is to set up your test/migration region to use a feature of z/OS called "Zero Address Detection":

    SLIP Zero Address Detection (ZAD) is a z/OS feature that detects and documents execution of an instruction that accesses (stores or fetches) storage by using an operand address that was formed from a general register containing zero. This detection feature allows the owner of an application to identify programming errors where an assembler instruction is inadvertently accessing data within the Prefixed Save Area (PSA) control block, which resides at virtual address zero, due to incorrect register content of zero.

    https://www.ibm.com/support/pages/node/6602085

  - Set up ZAD in your test/migration region and regression test your applications. If zero address usage is found, then correct the programs.

Use of zero addresses – another solution!

- How to find AND PREVENT uses of zero addresses?
  - New COBOL compiler option LSACHECK: Linkage Section Addressability Check
  - This option tells the compiler to initialize BLL cells (Base Locator for Linkage section) to x'7FFFF000' instead of the previously normal x'00000000'. This is a special address in z/OS storage for an area that has no actual backing storage. Any reference, read or write, will get an ABEND 0C4
  - This option has ZERO run time overhead, it does not affect performance in any way
  - You could use this option only in development, or once you are confident that most zero address uses have been fixed, you could use LSACHECK in production

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

Examples of invalid data

Best practices for COBOL V6 Migration

How to prepare for COBOL V6 before you buy

Resources

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 programs are using invalid data, IBM has recommendations for migrating to COBOL V6. The first time that you compile a program with V6:

1. Compile with SSRANGE, NUMCHECK, PARMCHECK, INITCHECK, LSACHECK 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
   - Inspect listings for INITCHECK messages
   - Look at runtime logs for NUMCHECK, etc, error messages
2. Recompile with NOSSRANGE, NONUMCHECK, NOPARMCHECK and OPT(2) for quality assurance test and production
   - NOSSRANGE, NONUMCHECK and NOPARMCHECK are required for good performance. LSACHECK and INITCHECK have no impact
   - OPT(2) is preferred for good performance in production

- Note: You may have to change to a 2-compile development process if you are not using one already
- Note2: We also recommend setting up ZAD(Zero Address Detection) in your test/migration region

Best Practices for COBOL V6 Migration

To help developers write better COBOL programs…

- 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)
  - NOOMITODOMIN (flags ODO clause with no minimum occurrences)
  - NOUNREFALL (flags all items that are not referenced, including COPY)
  - NOUNREFSOURCE (flags only unref'd items in the COBOL source)
- 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
  - ARCH should ONLY be set as installation default, never et at compile time. This way you can have central control for all
- NUMPROC(MIG) is removed, which requires special consideration and extra testing.
  - Usually use NUMPROC(NOPFD)
  - Also look at INVDATA(FORCENUMCMP,NOCLEANSIGN)
  - Using NUMCHECK(ZON,PAC) with NUMPROC(PFD) will tell you if your signs are always preferred, allowing you to migrate to NUMPROC(PFD) and NOINVDATA 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

What, when, and why of COBOL Migration

A brief history of COBOL compilers on z/OS

What is different about COBOL V6 migration

Examples of invalid data

Best practices for COBOL V6 Migration

How to prepare for COBOL V6 before you buy

Resources

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.*)
```

  This command will look for all PTFs needed for COBOL V6.3

- Install indicated PTFs on **all systems** before using the new compiler

COBOL V6: Before you buy

- Install latest maintenance for vendor tools
  - Older versions may not support programs compiled with V6
- If you use third-party code in your application, check with the vendors for updates
  - Some vendors have updated their applications; others may need to
- Convert PDS COBOL load libraries to PDSE datasets
- 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

New IBM tool to assist with COBOL migration!

- COBOL Upgrade Advisor for z/OS (CUAZ)
- CUAZ:
  - Takes inventory of programs in load libraries
  - Discover programs used by an application
  - Helps with using migration options and locating instances of invalid data usage
- The initial release of COBOL Upgrade Advisor for z/OS 1.1 (CUAZ) focuses on addressing the COBOL upgrade challenges associated with **planning and scoping an upgrade project**
- We welcome ideas from our clients to guide the future direction of CUAZ
  - https://ibm-z-software-portal.ideas.ibm.com/ideas/?project=CUAZOS

Parts and packaging for COBOL Upgrade Advisor for z?OS

PID: 5900-BPP

Perpetual License

Part Number	Part Description	Entitles customer to
D10**K2**ZX	CUAZ Virtual Server Perpetual License	Analysis engine (runs on mainframe)
D10**JS**ZX	CUAZ Authorized User Perpetual License	UI (VS Code extension)

Subscription License

Part Number	Part Description	Entitles customer to
D10**K1**ZX	CUAZ Virtual Server Subscription	Analysis engine (runs on mainframe)
D10**JR**ZX	CUAZ Authorized User Subscription	UI (VS Code extension)

Requirements

- Sold on Passport Advantage
- Minimum purchase: 1 Server + 2 Authorized User licenses
- User must purchase **both** server and authorized user licenses to use CUAZ
- Subscription licenses offer Term flexibility from 12 to 36 months with a minimum term of 12 months [link]

COBOL V6: Before you buy

- Use tools to take inventory of your COBOL load libraries
  - Use IBM COBOL Upgrade Advisor for z/OS and/or
    - IBM File Manager
    - Or COBOL Migration Manager by DTS Software
    - Or Edge Portfolio Analyzer! New release, new owner!
    - Or COBANAL Tape #321 at cbttape.org (freeware)
    - Or ‘Library Utility’ of File-AID/MVS (BMC)
    - Or ‘MRS Utility’ of CA SYMDUMP (Broadcom)
  - What compiler was used for each program
    - Especially OS/VS COBOL!

COBOL V6: Before you buy

- Take inventory of your COBOL load libraries
  - What compiler options used
    (esp CMPR2, NUMPROC, TRUNC)
    - Use the same options with COBOL V6 when possible
  - Do not change from NUMPROC(NOPFD) to NUMPROC(PFD) or from TRUNC(BIN) to TRUNC(OPT) without doing research and testing
    Note: You could use NUMCHECK to find out if changing to the better-performing setting is possible
    - Using NUMCHECK(ZON,PAC) with NUMPROC(PFD) will tell you if your signs are always preferred, allowing you to migrate to NUMPROC(PFD) and NOINVDATA for better performance

COBOL V6: Before you buy

- If you have OS/VS COBOL programs in load libraries
  - See if they are being used
    - LE messages IGZ0268 or IGZ0269 (may need to enable this feature of LE, see IGZUOPT)
    - COBOL Migration Manager by DTS Software
    - IBM Tivoli Asset Discovery for z/OS
  - If so, either target them for early migration to V6 or migrate them to V4
  - If not, delete them from Load Libraries!
  - Get rid of the “OS/VS COBOL problem” early
- Use IBM CCCA to convert source
  - https://www.ibm.com/docs/en/cobol-zos/6.4?topic=optac-cobol-cics-command-level-conversion-aid-zos-ccca
  - OS/VS COBOL programs
  - Pre-V3 programs compiled with CMPR2

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

Examples of invalid data

Best practices for COBOL V6 Migration

How to prepare for COBOL V6 before you buy

Resources

COBOL Resources

Enterprise COBOL
- Product Page: https://www.ibm.com/products/cobol-compiler-zos
- Documentation: http://www-01.ibm.com/support/docview.wss?uid=swg27036733

Suggestions for improvements to IBM products: The IBM Ideas Portal
- https://ibm-z-software-portal.ideas.ibm.com/

COBOL Blogs and discussion:
- https://community.ibm.com/community/user/ibmz-and-linuxone/groups/public?CommunityKey=dc94cb0f-7361-47d9-854f-dfcbdbbf04a3

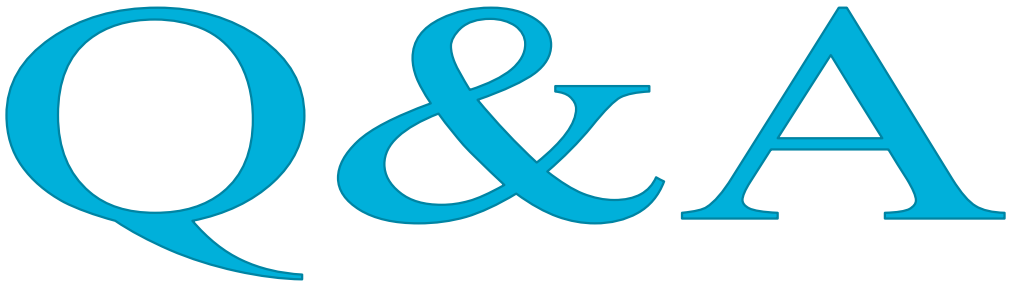
COBOL Performance
- Guide: https://www.ibm.com/docs/en/SS6SG3_6.4.0/pdf/perfguide.pdf

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: https://www.ibm.com/account/reg/signup?formid=urx-49350

Questions?



Thank You!

Experience more with IBM

Visit us at the IBM Booth #113

After a full day of technical sessions, take a break with us!

Connect with our experts, snap a photo with the z17 Plexi or the latest Telum II, and get an up-close look at our Spyre Accelerator.

Come back each day for fresh topics and demos at our expert stations.

Think 2026

Join 5000+ senior business and technology leaders who are seizing the AI revolution to unlock unprecedented growth and productivity at **Think 2026**.

Find out more information using the QR code below.

IBM Digital Asset Haven

IBM Digital Asset Haven is the operational backbone for financial institutions and regulated enterprises entering the digital asset economy.

Find out more information using the QR code below.

AGENDA

- Explain COBOL compiler history and ‘generations’
- Explain why we needed a new code generator for third generation COBOL (hardware exploitation)
- Explain source compatibility vs runtime results
- Describe relative difficulty of different COBOL migrations
- Explain the concept of ‘invalid data’ in COBOL data items at run time and how they can result in failed migration
- Describe differences in using the new compiler
- Describe compiler ‘tools’ (compiler options) to assist in identifying the presence or absence of invalid data in COBOL programs at run time
- Describe our recommended 2-compile and 2-test process for due diligence in having a successful migration
- Explain other requirements for migration:
    - PDS load libs to PDSE, OS/VS COBOL, required service, tools that can help

Invalid Data in Numeric USAGE DISPLAY data items

Side note about invalid data, actual customer example.

In MOVE alphanumeric to numeric the alpha is treated as unsigned numeric external decimal usage display:

```
  77 INS-PURCHASE-UNIT  PIC X(3) VALUE X'00006C'.
  77 INS-MIN-ORDER-QTY REDEFINES INS-PURCHASE-UNIT
                        PIC 9(5) COMP-3.
  77 WS-PURCHASE-UNIT   PIC 9(5) COMP-3.

 PROCEDURE DIVISION.

  MOVE INS-PURCHASE-UNIT TO WS-PURCHASE-UNIT  *> ABEND ARCH(12)
**********************************************************
* Even though the data in INS-PURCHASE-UNIT is valid
* Packed-decimal, the compiler tries to convert external
* decimal to packed decimal.  With PACK it did not ABEND,
* but with Vector Packed Decimal instructions, it started
* ABENDing.  ARCH(11) no ABEND, ARCH(12) S0C7!|
**********************************************************
  MOVE INS-MIN-ORDER-QTY TO WS-PURCHASE-UNIT  *> OK any ARCH
```


Invalid Data in Numeric USAGE DISPLAY data items

Things to look for:

- **Look for REDEFINES where a numeric is redefined by an alphanumeric, then look for nonnumeric data being MOVEd into the alpha item, like MOVE 'AC' to alpharedef**
- **Data coming from a user**
- **Validate any data that might be bad using NUMERIC class test:**

```
IF xyz IS NUMERIC THEN
    Use it
ELSE
    send a message, stop processing, etc
```