December 20, 2019

Control/DCD and COBOL 6.2-6.3

Purpose:

The goal of this White Paper is to inform the reader of the issues with COBOL 6.2-6.3 regarding the efficiency of the executable code generated by these new compilers. We believe that there are two modes of compiler executable code. Namely, Native Mode versus Option Mode. There are several optimization improvements that can be obtaining by using Marble Computer's Control/DCD software.

IBM claims in their presentations that 25% of customers converting to COBOL 6.2. found problems with Invalid Data. Therefore, customers turn on a Compiler option called NUMCHECK which generates If Numeric statements for each numeric data field through the program. The NUMCHECK option adds up to 50% more overhead during execution time. Thus 75% of the customers are experiencing additional run time overhead for no reason. Control/DCD can provide information as to whether or not a program is going to have an Invalid Data issue.

Background:

IBM has provided Enterprise COBOL compilers for its z/OS Mainframe product lines going back to 2004. Since 2004 IBM has introduced a total of 9 COBOL Compilers. IBM Mainframe product line COBOL compilers were first introduced in 1960. Up until June 21, 2013 all the previous compilers were based on COBOL/360 technology. With the introduction of COBOL 5.1 the current in-service Compilers are based on Java technology. Since COBOL 5.1 all the new compilers are for purchase. All previous compilers were on a monthly license charge.

With the introduction of COBOL 5.1 the adoption rate was very slow until COBOL 6.2. COBOL 6.2 has since become the standard compiler for most z/OS mainframe customers. Some customers have still held on to the last of the 360 Technology, COBOL 4.2, which goes out of service September 30, 2021.

Problem:

Prior to COBOL 5.1 previous COBOL compilers were lax in their coding standards. Each new compiler accepted poor coding standards. Over a 40-year period COBOL code became less efficient due to the movement of COBOL programmers from job to job. New code added during each maintenance enhancement was not always done effectively.

The new Enterprise Compilers enforce very strict standards. The stricter standards come with the promise of better object code efficiency. New Compilers come with the end of Monthly Licensee fees in return for a one-time payment. This is important to IBM's mainframe customers for one main reason, better cost control. As transaction rates continually increase so does workload on the z/OS Mainframes. Faster CPU's and other efficiencies make the ROI on staying with IBM more attractive!

Enterprise Compiler Issues:

Since the new z/OS Cobol compilers enforce stricter standards for clean code there is a need for a well thought out migration process to take advantage of the cost savings and efficiencies promised by IBM.

The migration process to the compilers is not that simple. Just recompiling an older program does not always work without manual effort. The manual effort required is reasonably extensive based on the state of the old code. Here at Marble we've identified 6 areas of concern that need to be taken into account when starting a migration project to an Enterprise COBOL compiler.

The 6 Problem Areas of concern when moving to COBOL 6.2 – 6.3:

1. INVALID DATA not flagged in earlier COBOL
    - Literal '1_3' containing 'F140F3'
2. COBOL PARAMETER inconsistencies
    - Parameter Sizes not the same or parameter missing
3. Excessive Compile Time with newer compilers
    - 5 to 10 times for smaller programs, more for larger.
4. OCCURS / DEPENDING where existing code exceeds table size.
5. Not initially assigning a VALUE in Working-Storage.
6. PACKED and BINARY fields defined too small.

Invalid Data is the most egregious of the 6 problem areas. IBM reported at various conferences to potential COBOL 6.2 customers that about 25 percent of customers migrating to Enterprise Compilers reported Invalid Data Issues. These can be serious in their nature as displayed in item #1. The example below was pointed out by IBM on several occasions at multiple customer conferences.

Example of Invalid Data Usage for COBOL V6.2

```
o  77 A1 PIC X(4) VALUE '00 0'.     X'F0F040F0'
o  77 A2 REDEFINES A1 PIC 9(4).
o  PROCEDURE DIVISION.
o  IF A2 = ZERO DISPLAY 'ZERO' ELSE
o  DISPLAY 'NOT ZERO' END-IF
```

The above example passes through the new compilers without error warnings. In production the test using the redefines field, A2, fails and results are unpredictable. While the above may seem extreme take into account the following:

- Worldwide COBOL code currently in production – 250 Billion lines.
- Many COBOL applications have programs with 20-80,00 lines of code.
- Many IBM customers have COBOL programs numbering between 5-120 Million lines of code.
- COBOL Parameter passing is the next biggest issue for migration to the new Compilers.
- Many COBOL applications use a process where a main program calls other smaller programs or sub-routines, i.e. check digit calculation. In COBOL 6.2 the Parameters and the Calling Arguments must be the same size. Former Compilers ignored the size differences.

Compile Time and Storage. The new compilers use enormous amounts of Address Space compared to previous compilers. Compile times can run 5-20 times longer depending on the size of the program. Enterprise compilers require load modules to be stored in PDSE rather than the current PDS format. Add to this factor that COBOL 6.2 compiled code cannot call programs from previous versions of COBOL compilers.

Occurs Depending statements can be a problem. In past IBM compilers non-accurate table sizes would leave room for execution errors. Not so with COBOL 6.2. Table size and arguments must be identical.

Not assigning a value to Working Storage fields is a problem that IBM strongly suggests changing to avoid future problems.

BINARY fields must not exceed the number of digits in the PICTURE clause. Earlier versions allowed this as PIC S9(4) BINARY could hold up to 32,767 positive digits.

Packed (or COMPUTATIONAL-3) fields sometimes used an even number of digits. This is strongly frowned upon in the newer COBOL 6.2 or 6.3. An ODD number of digits is wanted.

IBM's Solution to Its COBOL Enterprise Compilers:

The IBM compiler team in Toronto made a decision to solve these issues by building compile time options. Some of the options are listed below.

- NUMCHECK – NUMCHECK(ZON) options tell the compiler whether to generate extra code to validate data items when they are used as sending data times. For zoned decimal (numeric USAGE DISPLAY) and packed decimal (COMP-3) data items, the compiler generates implicit numeric class tests for each sending field. For binary data items, the compiler generates SIZE ERROR checking to determine whether the data item has more digits than its PICTURE clause allows.

Essentially the Compiler is now adding additional code into the source and compiling the source with new code. According to IBM this can add up to 50% more execution time. In other words, Option Mode negates some of the value of moving to COBOL 6.2 or 6.3. Especially for high volume calculation applications.

- PARMCHECK option tests for subprograms that write beyond the end of WORKING-STORAGE. This option tells the compiler to generate an extra data item following the last item in WORKING-STORAGE that is then used at run time to check whether a called subprogram corrupted data beyond the end of WORKING-STORAGE.
- SSRANGE sub-options allow:
    - MSG/ABD Choose a warning message or an ABEND for SSRANGE conditions.
    - ZLEN allows a reference modification of zero length to proceed without a message or abend.

The above 2 options, PARMCHECK and SSRANGE, have an unknow effect on workload performance. Here IBM is providing a work around a technical issue. In both cases additional code is being added into the source. However, we suggest if you can avoid these options the easier it will be to maintain and modify the programs that by Enterprise Compiler standards are in error.

Why You Can Trust Marble Computer:

Marble Computer was established in 1983 to provide software tools to document COBOL applications. Our Flagship product DCD has dwarfed into the Current version Control/DCD 2.3. Our technical team is headed up by Marshal Crawford, CTO. Mr. Crawford is an expert on COBOL. He has written 5 books on the subject of COBOL in its application as an industrial strength programming language for mission critical applications.

In 2017 Mr. Crawford consulted with the IBM Enterprise Compiler Team in Toronto, Canada after the dismal performance of COBIL 5.1 and 5.2. We think some of his comments assisted in the development of a more reliable Enterprise COBOL 6.1, 6.2 and 6.3.

Control/DCD tackles the 6 major problems as previously defined. As a migration tool Control/DCD will:

1. Point to Programs that have issues with Invalid Data. Each program is analyzed and a Documentation Manual in .pdf format is created with a list of error conditions. The DDM points the programmer to the offending data fields, code and exact location as to where find the errors.
2. Highlight, all parameter, size argument errors between programs and called routines.
3. Be used as a pre-compiler to save CPU time and resources in order to find the errors in programs prior to using the Enterprise Compiler.
4. Identify Occurs/Depending errors in all programs analyzed.
5. Identify Working Storage fields not assigned an initial value.
6. Identify Packed and Binary Fields that do not meet Enterprise Compiler requirements for efficiency.

Additionally, Control/DCD will find:

1. Perform errors (Major & Minor) including Go To statements within a performed routine.
2. Code Not Used.
3. A hierarchical format for understanding the structure and flow of the analyzed program for easier maintenance.
4. Abend analysis after a data error.

All of the above will improve Enterprise Compiler workflow and CPU performance by executing in Native Mode rather than Option Mode.

Enterprise Compiler Release Dates and EOS for Compilers Currently in Service:

4.2 – August 28, 2009 – EOS September 30, 2021
5.1 – June 21, 2013 – EOS April 30,2020
5.2 – Feb 27, 2015 – EOS April 30, 2020
6.1 – March 18, 2016
6.2 - Sept 8, 2017
6.3 – Sept 6, 2019