Migrating to newer IBM Enterprise Compilers

Marshal Crawford – CEO Marble Computer, Inc.

Stopping the show to migrate to new software takes focus in two directions as follows:

1. One is the focus on what is involved in the migration
2. Two is re-focusing on current tasks to incorporate handling the new migration

When time frames are set to do the conversion, it is important to have all the tools needed to make the conversion run smoothly, so that once again, making the other in-house tasks higher priority can be done quickly.

Migrating to Enterprise 5, 6 or beyond has some significant conversion problems that will be covered here along with suggested ways of handling them.

Six areas of concern:

1. **INVALID DATA** (checking is done by newer (from V5 on) IBM compilers to insure the NUMERIC fields in DISPLAY usage [not BINARY or PACKED] are correct in a way that older compilers do not check)
2. **COBOL PARAMETER inconsistencies** (older COBOL compilers accepted the field length and number of parameters in a Program-Entry or in a CALL statement as correct, and the newer IBM compilers put in code to verify during execution these lengths and number of parameters match each other)
3. **Excessive COBOL compiler time** (the newer compilers do more to speed execution after compiled code is put into production AND unfortunately take up to 5 to 10 more times to do the compile – see discussion later under this sub-heading)
4. **OCCURS/DEPENDING errors** (old compilers allow execution to go beyond the limits specified within the DEPENDING ON name and the newer compilers do not allow this)
5. **NOT INITIALLY ASSIGNING A VALUE** (the newer compilers check more carefully and for fields that are not initially assigned a value)
6. **PACKED and BINARY fields defined too small** (old compilers did not check and did truncation and newer compilers do check)

INVALID DATA

Definition of INVALID DATA for converting to COBOL 5 or 6

The format of NUMERIC fields in COBOL for character or DISPLAY usage consists of two HEX digits making up one NUMERIC character. For instance, the HEX format for the numbers 1, 2 and 3 are F1, F2, and F3.

For unsigned number **123**, the hex format stored in the computer is **F1F2F3**.

In older IBM compilers a check was done on the last or low order digit to see if that HEX format was either Fn,

of Cn, or Dn, where D indicated a negative number for the entire field or was a C or F for a positive number.

+123 = **F1F2C3** (The C indicating a forced positive number)

The other digits are all assumed (in older compilers) to be an F with no implicit check. That allowed for a blank character, hex 40 or other invalid combination to be there. See below example:

'1 3' = 'F140F3' where the middle character is a blank

What problem can this cause

The newer compilers (from 5 on), create different results from the older compilers when this is there. The way to not be surprised is to catch these errors.

How to identify

Without using the solution offered by Control/DCD, each installation must find their own way of identifying these errors.

Control/DCD can help

Control/DCD is a software tool that analyzes COBOL programs and COBOL applications. The core code has been in use since 1975 analyzing over a billion lines of code since its introduction. Control/DCD has grown from analyzing COBOL D & F through now COBOL 6.2.

There are six (6) components that make up Control/DCD.

1. The Digital Documentation Manual (DDM)
2. COBOL Source Editor
3. The Core Kernel formally known as DCD
4. COBOL Application Analyzer
5. Abend Analysis
6. JCL Analysis

The six (6) components of Control/DCD provide analysis on COBOL programs with many features including the following:

1. It is also designed to be used as a pre-compiler in place of extra COBOL compilers to avoid making extra compiles that take much longer.
2. It creates Warnings and Error message like the COBOL compiler.
3. An option in Control/DCD may also be turned on to find INVALID data as error messages in the following ways:
   - If a bad non-numeric literal with an INVALID entry is in a REDEFINES to a numeric DISPLAY entry, this is flagged as such.
   - If a bad non-numeric literal with an INVALID entry is MOVED to REDEFINES to a numeric DISPLAY entry, this is flagged as such.

- If a bad literal exists in a VALUE in the DATA DIVISION and is MOVED to another field that is a REDEFINES to a numeric DISPLAY entry, this also is flagged.
- If a bad literal is MOVED to a one field, that is later MOVED to another REDEFINES of a numeric DISPLAY field, then this is flagged.

Simply run Control/DCD on this program and then go to the Error Message generated and look for possible errors.

An example of these messages generated is shown below:

```
     MSG-NBR    DESCRIPTION
  CSEM6KC3-I  MOVE '12T44' may reach field at (0481) via MOVE at 01346
  CSEM6IB2-I  MOVE HIGH-VALUES may be INVALID for field at 00153
```

Use these errors to correct your COBOL code during conversion to COBOL 6.2 or other newer Enterprise COBOL.

COBOL PARAMETER inconsistencies

One of the new enforcements going to newer ENTERPRISE COBOL is that the passed parameters fields between program must have the same size between CALLING program and CALLED program.

For example, if program-A calls program-B and passes a TABLE with 8,000 characters in the table, both record defined sizes must match, or an error is generated.

Sometimes in older COBOL, it made sense to define a TABLE size that would invariably change over time as the correct 8,000 characters in the main program and then if program-B does not access the table itself, but instead passes the record to program-C that does access the table, then program-B and other in-between programs could have a record length of 1 character or other size, and only program-A and the program that actually uses the record (e.g. program-C) need to be modified when the table size changes.

Not so, for COBOL 5 or 6 and beyond. All record lengths for the same field (usually a record) need to be the same.

One way to handle correction is go through each program and built a list of record sizes, number of arguments or parameters passed and very carefully chart and verify there are no differences. This always takes time and sometimes an error happens in verification and some of the differences are not found.

Note – When correcting the problem, both parameter size and the number of arguments passed need to be looked at.

Control/DCD has a CALL/Parameter report which does this checking on all COBOL programs at once and with one simple report, complete verification is done for each correcting of this problem. We generate an ERROR MESSAGE for inconsistencies. See below.

```
CALL Parameter reporting for Program ( TRENTR1 )

  Procedure Division USING for program TRENTR1                Length
                    1   WS-FIRST-FIELD                          200      @00323
                    2   WS-2ND-PARAMETER                        400      @00296

  From:  "TMCALLN" ENTRY at compile nbr 000646

    ENTRY TRENTR1 USING                                       Length
                    1   WS-MATCHES-FIRST-FIELD                  200      @00243
        ** ERROR **  LENGTH does not match P-D USING #   2
                    2   WS-DOES-NOT-MATCH-2ND-PARAM             402      @00124
        ** ERROR **  PARAMETER # 00003 NOT USED in P-D USING
                    3   WS-EXTRA-PARAMETER                       99      @00075

  From:  "TMCALLN" ENTRY at compile nbr 000857
    ENTRY TRENTR1 USING                                       Length
                    1   WS-MATCHES-FIRST-FIELD                  200      @00582
                    2    - - ( parameter missing ) - -
        ** ERROR **  Parameter MISSING for corresponding USING #   2
```

A **B** **C** D **E** **F** **G**

Report Fields

- **A** - Line with Procedure Division or from "progname"
- **B** - ENTRY or CALL with shown name
- **C** - ** ERROR ** message line
- D - Parameter number
- **E** - Parameter name
- **F** - Length of parameter
- **G** - Sequence of this field

By using these errors to correct the code, this solves the CALL PARAMETER problem for converting to newer ENTERPRISE COBOL.

Excessive COBOL compiler time

Compile Time is longer when using COBOL 5 or 6. Not just longer, but way longer! The address space size used during compile time is also significantly greater. The benefit of COBOL 5 or 6 is significantly faster execution time of the program in production! This is good. The extra Compile time is not good!

There are two considerations here as follows:

1. With extra compile time, a way is needed to do a compile less often. This is a significant issue.

2. It is also possible to speed up the execution time for some programs that have a poor PERFORM structure. See 'Improving Compile Time' below and note reasons why this might be important.

Control/DCD has a **pre-compiler** that may be used rather than continual use of the longer processing COBOL compiler and Control/DCD has **two advantages** as follows:

1. It generates ERROR messages like a COBOL compiler and can be used to remove most of the errors (96%) before doing a final compile
2. It generates all sorts of unique documentation available in PDF format for off the mainframe or in another close to similar format for viewing on the mainframe. This ANALYSIS speeds up analysis when working on a COBOL program.

There is another white paper on the MARBLE COMPUTER website that deals with just this topic.
See 'Why Control DCD should be used as a pre-compiler for new Enterprise COBOL 6.2' on the Marble Computer website.

Improving compile time

Minimum time to compile a program takes 5 to 10 times longer using COBOL 5 or 6, versus the older COBOL compilers before them.

A bad program (defined as a program that exceeds the estimate of 5 to 10 times and can go to 15 times (or greater). The largest reason for the excess time is difficult PERFORM structure. This statement conforms to discussions between Marble with people at IBM in their main facility for COBOL in Toronto. See next paragraph.

When a program takes more time to compile, this may be a red flag to the user that something needs to be done to clean up the program for the following reason:

- Almost always, when compile time takes significantly longer, the PERFORM structure of the program is convoluted in a way that makes maintenance harder on this program.
- Also, there may be hidden PERFORM Errors in the program that are causing much longer compile time. (The danger in PERFORM Errors is simply that the logic is probably incorrect and may be creating hidden errors in the output and may be costing your company extra dollars depending on the error.) To not pay attention to Perform Errors is like running empty in a car gas tank. Marble handles this issue. See below.

As just stated, cleaning up extra compile time, also saves extra money by making the maintenance of the COBOL program clearer and easier to follow. Or in reverse, ignoring the red flag, leads to status quo and continual longer-term maintenance and extra costs beyond longer compile time.

Cleaning up the COBOL program should ideally be done by a technician with good, or better than average experience with COBOL programs.

Also, PERFORM ERRORs mentioned above, are documented clearly by MARLBLE Computer in their Control/DCD software. The error message states where the error is, and the technician evaluates than removes the error.

OCCURS/DEPENDING

Before COBOL 5, the defined limits in the OCCURS clause handled by COBOL compilers could be exceeded by using a larger value in the data-name defined within the DEPENDING clause. The technician or programmer who managed the program, needed to be sure there was extra table space beyond the OCCURS to not upset the apple cart and cause an ABEND or worse have an undetected problem that was permanently there and never fixed.

With COBOL 5, 6 and beyond, the limits defined within the OCCURS are strictly forced. In converting in the process of moving forward, each program needs to be carefully examined to see if the table OCCURS limit is exceeded.

Using Control/DCD saves an enormous amount of time in finding tables that violate this rule, especially when checking a whole system of COBOL programs.

A special OCCURs report exists (providing OCCURS option is on) within the DDM to show relevant documentation for finding possible OCCURS errors. See sample report below:

```
  #OCCURS REPORT      &OCCURS REPORT
  Used to verify legal range in OCCURS when DEPENDING on Data-Name is used

(0026) &T2-NBR-ENTRIES
     In 453-454 of T1-TABLE
     in WORKING-STORAGE
     05  T2-NBR-ENTRIES
                 Pic S9(4)  Value ZERO
                 Usage is COMP
     Used in Depending on T2-NBR-ENTRIES (30)      <--- DEPENDING ON field
     Move 50 to T2-NBR-ENTRIES (721)
     Add 1 to T2-NBR-ENTRIES (911)
     Set T2-INDEX to T2-NBR-ENTRIES (633)

(0029) &T2-ENTRY
     In 1-15 of 01 T2-TABLE
     in WORKING-STORAGE
     05  T2-ENTRY  OCCURS  5 TO 30  TIMES
            DEPENDING  ON  T2-NBR-ENTRIES          <--- OCCURS/DEPENDING
     MOVE ZEROS to T2-ENTRY (905)
```

Notice that the 'Move 50 to T2-NBR-ENTRIES' above, exceeds the maximum OCCURS 30 TIMES just above. The user then sees the problem and is tasked with how to fix this problem.

NOT INITIALLY ASSIGNING A VALUE

There are two ways a VALUE is not assigned initially as follows:

- A field does not have VALUE clause with a value assigned (first choice to look at)
- The field (or the record it belongs to) is initialized by moving another field or record to it, where that field or record is not initialized.

Numeric VALUE in a DISPLAY field are covered by Control/DCD errors messages.

```
       MSG-NBR     DESCRIPTION
 295  CSEM6H01-I  WORKING-STORAGE Numeric USAGE DISPLAY Field has NO VALUE
1386  CSEM6H01-I  WORKING-STORAGE Numeric USAGE DISPLAY Field has NO VALUE
```

IBM also generate errors messages like those generated by Control/DCD for Group fields, to show that no VALUE is present.

It is also possible for PROCEDURE DIVISION activity to initially set fields with literals or non-WORKING-STORAGE fields.

Control/DCD has narrative for each field showing all PROCEDURE DIVISION activity for each field. Marble recommends using our older 'Alternate Compile Facility' feature still available in Control/DCD to find and fix these errors.

Go through each field (and record) in the program and quickly review the PROCEDURE DIVISION Narrative for each field looking for possible discrepancies.

An example this NARRATIVE is given here:

```
     01   AW-RECORD                      > Move SPACES to # (2043, 3148)
                                           If # not = SPACE (2374)
          05  AW-NUM-ACCT   PIC 9(3)     > Move ZEROS to # (5381, 5933)
                                           If # not numeric (4232)
```

PACKED and BINARY fields defined too small (Overpopulated Data Items)

Overpopulated Data Items as defined by IBM COBOL Migration Recommendations are restricted to MOVEs between overpopulated data items and redefined fields with other USAGES.

This document is broken into two parts:

1. **Definition**
2. **Suggestions for Identifying Errors**

Definition

A good understanding of the internal data representation for USAGE DISPLAY, PACKED-DECIMAL, and BINARY or COMP usage may be needed for following some of the following definition.

This document is a recap of the problem as defined by IBM for converting to new compilers and then suggests a two-step process to identify these errors.

Whether a user in moving to COBOL 5 or 6 or not, their current COBOL programs will be better off if they find and remove these errors in their existing code.

Understanding the problem:

```
05  WS-ELEMENTARY-FIELD      PIC X(02).
05  WS-COMP-3 REDEFINES WS-ELEMENTARY-FIELD
       USAGE PACKED-DECIMAL  PIC 9(02).
```

A packed-decimal field is allowed 3 numeric digits within two bytes or two characters in the packed decimal format of digit-digit-digit-sign or '567F' where the letter F, C, or D is the sign and 5, 6 and 7 are the digits.

As the field WS-COMP-3 is a REDEFINES of another field, an original value or a MOVE to the other field, can put a VALUE in WS-COMP-3 of more than two digits, such as the '567F' shown in the previous paragraph.

The previous example showed a packed decimal field. The following example shows the same type of problem created with a binary field:

```
05  WS-SECOND-ELEM-FIELD     PIC X(02).
05  WS-BINARY-FIELD REDEFINES WS-ELEMENTARY-FIELD
       USAGE BINARY          PIC 9(03).
```

Here again, the field WS-BINARY-FIELD is a REDEFINES of another field and this allows a MOVE or original VALUE to affect WS-BINARY-FIELD establishing a VALUE greater than 999.

Suggestions for Identifying Errors

Step 1 is to identify all PACKED or BINARY fields in the program and eliminate those from consideration that either are not REDEFINED and also not affected by group moves that could have literals moved to the group field that would affect selected fields.

For users of Control/DCD, consider option 5 from its main menu to do 'Abend Analysis' which allows a SELECT language and use control statement:

```
SELECT IF USAGE = C3 OR C
```

This selection will pull off a report of all Packed and Binary items in the program and the user will be assured that none are missed. Once done, look for appropriate REDEFINES to look at more closely, unless using Control/DCD software, in which case, looking for REDEFINES may be bypassed by using the 'Digital Documentation Manual' for this program as covered in **Step 2** below.

Step 2 is to look for Packed or Binary MOVEs to the REDEFINED fields to these PACKED or BINARY fields.

The previous step identifies these usually elementary fields. Without Control/DCD, when using the cross-reference map or other method to find each field, look at all references found and see if there are PROCEDURE DIVISION statements that cause a problem.

For users of Control/DCD, it is recommended that the user create a Digital Documentation Manual in PDF format with option INDIRECT REFERENCES set to a **'Y'**, under 'Modify Type Analysis provided' and then simply look at each identified field in the SELECT USAGE step above and then use the INDIRECT REFERENCES line as shown in example below to see where MOVEs are that need to be verified.

```
(0467) WS-COMP-3
     in WORKING-STORAGE
     05  WS-COMP-3   PIC 9(02) USAGE COMP-3
      Indirectly Changed @1792
```

The @1792 identifies a MOVE to a REDEFINED field at line 1792. To check for an error here, go to COBOL Procedure Division line 1792 in the DDM and look at the MOVE or other statement found there. See example below:

```
       MOVE '567C'  TO  WS-ELEMENTARY-FIELD.
```

This statement shows how a 3-digit signed positive field is moved into a two-digit defined PACKED DECIMAL field. This identifies an error that needs to be corrected.

Please contact us for more information on Control/DCD or for setting up a demo at info@marblecomputer.com or visit our website at www.marblecomputer.com