TOPS–10/TOPS–20 COBOL Conversion Utility Guide

AA–M586A–TK

July 1982

This manual reflects the software of Version 12B of 68274.EXE, the COBOL–68 to COBOL–74 Conversion Utility.

This is a new manual.

OPERATING SYSTEM:	TOPS–10 V7.01 or later TOPS–20 V4.0 or later
SOFTWARE:	68274 V12B

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region

Digital Equipment Corporation
PO Box CS2008
Nashua, New Hampshire 03061
Telephone:(603)884–6660

Central Region

Digital Equipment Corporation
Accessories and Supplies Center
1050 East Remington Road
Schaumburg, Illinois 60195
Telephone:(312)640–5612

Western Region

Digital Equipment Corporation
Accessories and Supplies Center
632 Caribbean Drive
Sunnyvale, California 94086
Telephone:(408)734–4915

digital equipment corporation • marlboro, massachusetts

First Printing, July 1982

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The following are trademarks of Digital Equipment Corporation:

DEC	DECnet	IAS
DECUS	DECsystem–10	MASSBUS
DECSYSTEM–20	PDT	PDP
DECwriter	RSTS	UNIBUS
DIBOL	RSX	VAX
EduSystem	VMS	VT
digital	RT	

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

Contents

Appendix A Differences Between COBOL–68 and COBOL–74

Notes

Preface

This manual describes the COBOL–68 to COBOL–74 converter utility program, 68274, as it has been implemented on the TOPS–10 and TOPS–20 operating systems.

It is assumed that the reader has a knowledge of the COBOL–68 and COBOL–74 languages. This manual is intended as a guide and as a reference manual for COBOL programmers who wish to convert COBOL–68 programs to COBOL–74 programs.

Chapter 1 contains an introduction to the converter and defines the basic characteristics of the program. Chapter 2 describes how to build and use the converter program, 68274, and provides COPY library and reserved word considerations. Chapter 3 describes the conversions that take place within the four divisions of a COBOL–68 program. Chapter 4 describes the messages that may be generated within the four divisions of a COBOL–68 program. Appendix A describes the differences between COBOL–68 code and COBOL–74 code. (A copy of this appendix can also be found in the *TOPS–10/TOPS–20 COBOL–74 Language Manual.*)

If you wish to reference the latest version of either COBOL–68 or COBOL–74, you should refer to the following manuals:

- *TOPS–10/TOPS–20 COBOL–68 Language Manual*, Order No. AA–5057B–TK
- *TOPS–10/TOPS–20 COBOL–74 Language Manual*, Order No. AA–5059B–TK

If you wish to learn the COBOL language, refer to the books listed in the FORWARD of both the COBOL–68 and COBOL–74 Language Manuals.

Chapter 1
The COBOL–68 to COBOL–74 Converter

1.1 Introduction to 68274

The COBOL–68 to COBOL–74 converter utility program, 68274, provides a tool to assist you in the conversion of a valid COBOL–68 coded program (Version 12B) to a valid COBOL–74 coded program (Version 12B). The converter utility is named 68274.EXE; however, your installation can give this program any other name.

NOTE

There is no guarantee from Digital Equipment Corporation that all programs can be converted without your help. In addition, correcting all differences found between COBOL–68 and COBOL–74 may not provide the efficiency needed for your programs.

The 68274 program does the following:

1. Tries to convert a COBOL–68 program to a COBOL–74 program.
2. Provides a listing with messages of all statements that either could not be converted or may have been converted incorrectly.

The program takes, as input, your COBOL–68 source program including COPY libraries and produces a converted source output file (with a file extension of .CVT) and a listing file (with a file extension of .LST). This utility does not produce a binary file (file extension .REL).

All COBOL–68 switches are valid. However, the only useful switches are /S, for card sequenced files, and /N, for no printing of errors on the terminal.

The converted source output file is in the same format as the COBOL–68 input file with as few source changes as possible, so that a FILCOM listing shows minimal differences.

1.2 Messages and Conversions

As described in the previous section, 68274 produces a .LST file and a .CVT file. The .LST file displays all warning messages that occurred during the conversion. These warning messages indicate COBOL–68 statements

1. that are illegal in their format for COBOL–74.
2. Contain data–names that are reserved words (see Section 2.4) in COBOL–74, but are not reserved words in COBOL–68.

Fatal messages may also appear if your COBOL–68 program does not compile correctly due to errors in the source code.

The .CVT file contains your COBOL–68 program as it can appear in COBOL–74 format. All COBOL–68 statements that can be converted to COBOL–74 statements are done so directly without any messages appearing in the listing (.LST) file. If you attempt to compile and execute this format using CBL74.EXE, you may receive new fatal and warning messages. Thus, the converted COBOL–74 program may need additional editing to compile correctly. To do any editing on the .CVT file, use the listing file for reference.

Using the example shown in Section 1.3, your directory may appear as follows after a conversion:

TOPS–20

```
@DIRECTORY(RET)

CBL20:<BROWN>
COBOL1.CBL.1
.CVT.1
.LST.2

Total of 3 files
@
```

TOPS–10

```
.DIRECT(RET)

COBOL1  CBL     2  <055>   13-Jan-82    DSKB:    [27,5107]
COBOL1  CVT     2  <055>   13-Jan-82
COBOL1  LST     2  <055>   13-Jan-82
Total of 6 blocks in 3 files on DSKB: [27,5107]

.
```

1.3 Line–Sequenced File Considerations

The 68274 program works with most forms of COBOL–68 source input. Input from your terminal presents no problems, with the exception of source files that contain line–sequence numbers (programs produced by EDIT or SOS). A source program that contains line–sequence numbers forces the converter to create a line number 1 greater than the previous line number if it does an insertion, but it does not correct subsequent line numbers. For example, a COBOL–68 source input file could be:

```
00001  IDENTIFICATION DIVISION.
00002  ENVIRONMENT DIVISION.
00003  INPUT–OUTPUT SECTION.
00004  FILE–CONTROL.
00005          SELECT OUTFIL ASSIGN TO DSK.
00006  DATA DIVISION.
00007  FILE SECTION.
00008  FD      OUTFIL LABEL RECORDS ARE STANDARD
00009          VALUE OF IDENTIFICATION IS "OUTFILDAT"
00010          DATA RECORD IS OUTREC
00011          BLOCK CONTAINS 20 RECORDS.
00012  01      OUTREC  PIC X(80).
00013  WORKING–STORAGE SECTION.
00014  77      A       PIC 9999 USAGE IS COMP.
00015  PROCEDURE DIVISION.
00016  START.
00017          OPEN OUTPUT OUTFIL.
00018          MOVE ZEROS TO OUTREC.
00019          MOVE 1000 TO A.
00020  LOOP.
00021          WRITE OUTREC.
00022          SUBTRACT 1 FROM A.
00023          IF A IS GREATER THAN ZERO GO TO LOOP.
00024          CLOSE OUTFIL.
00025          GOBACK.
```

and your converted COBOL–74 source output file would appear as:

```
00001  IDENTIFICATION DIVISION.
00002  ENVIRONMENT DIVISION.
00003  INPUT–OUTPUT SECTION.
00004  FILE–CONTROL.
00005          SELECT OUTFIL ASSIGN TO DSK.
00006  DATA DIVISION.
00007  FILE SECTION.
00008  FD      OUTFIL LABEL RECORDS ARE STANDARD
00009          VALUE OF IDENTIFICATION IS "OUTFILDAT"
00010          DATA RECORD IS OUTREC
00011          BLOCK CONTAINS 20 RECORDS.
00012  01      OUTREC  PIC X(80).
00013  WORKING–STORAGE SECTION.
00014  01      TALLY     PIC S9(5).
00014  77      A         PIC 9999 USAGE IS COMP.
00015  PROCEDURE DIVISION.
00016  START.
00017          OPEN OUTPUT OUTFIL.
00018          MOVE ZEROS TO OUTREC.
00019          MOVE 1000 TO A.
00020  LOOP.
00021          WRITE OUTREC.
00022          SUBTRACT 1 FROM A.
00023          IF A IS GREATER THAN ZERO GO TO LOOP.
00024          CLOSE OUTFIL.
00025          EXIT PROGRAM.
```

NOTE

The TALLY clause is generated automatically for your COBOL–74 program, whether your program needs this clause or not. (Refer to Section 3.3.)

Thus, no effort is made to check that line number 00014, as shown in the example above, already exists. Therefore, it is recommended that all line–sequenced COBOL–68 source files be resequenced (with increments greater than 1) before they are used as input to 68274.

If the source input file is card image (you would use the /S switch to compile the program), then each new line created gets the same line number as the previous line number and "68274" is entered in the comment field (columns 73 through 80). Thus, if line 00013 appeared as:

```
000013 WORKING–STORAGE SECTION.
```

then the next line, for TALLY, would appear as:

```
000013  01    TALLY    PIC S9(5).          68274
```

Chapter 2
How to Use the COBOL Converter Utility

2.1 Building the Converter

The 68274 converter utility is a variation of the COBOL–68, Version 12B compiler. This variation is produced by setting the feature test switch FT68274 to 1 (it is normally 0). To do this, you must add FT68274==1 to COBASM.MAC and rebuild the compiler in the normal way by using the control file COBOL.CTL (on TOPS–20) or by using COBOL.CTM (on TOPS–10). When you have rebuilt the compiler as 68274.EXE, copy (or move) the .EXE files to SYS:.

For example,

```
@EDIT COBASM.MAC(RET)
Edit: COBASM.MAC.1
*I2700(RET)
02750          FT68274==1(RET)
          ↑
         (TAB)
*EUN(RET)
[COBASM.MAC.2]
@
```

then:

```
.SUBMIT COBOL.CTM/TIME:01:00:00(RET)        (for TOPS–10)
```

or

```
@SUBMIT COBOL.CTL /TIME:01:00:00(RET)       (for TOPS–20)
```

2.2 Using the Converter

To use the converter utility, 68274, type the following on your terminal:

.R 68274 ⏎ (for TOPS–10)

or

@68274 ⏎ (for TOPS–20)

The general form of the converter command string is as follows:

*cvtfil,lstfil= libfil/l,src1,src2,...

where:

cvtfil — is the file that is to hold the converted source code. If no converted source code is desired, replace the file specification for cvtfil with a hyphen.

Example:

–,lstfil=src1,src2...

lstfil — is the file that is to hold the generated listing. If no listing is desired, replace the file specification for lstfil with a hyphen.

Example:

cvtfil,–=src1,src2,...

libfil — is the optional library file referenced by COPY verbs in the source files. If this file is not specified, the LIBRARY.LIB file is assumed.

src1,src2 — are one or more source files required to form one input program.

Each file specification has the following form:

device:file.ext [project,programmer]/switch/switch

where:

device — is the name of a physical or logical device. The name is composed of 6 or fewer letters and/or digits.

file — is the name of a file. The name is composed of 6 or fewer letters and/or digits.

ext — is the filename extension. It is composed of 3 or fewer letters and/or digits.

project — is a user's project number (TOPS–10).

programmer — is a user's programmer number (TOPS–10).

switch — is any of the switches shown in Table 6–1 of the COBOL–68 Language Reference Manual.

2.3 Copy Library Considerations

You must consider the following when text is copied from a copy library with the COPY verb statement in your COBOL–68 program:

1. The .LST file contains the COPY statement and the text as it is copied into your source program.
2. The .CVT file contains the COPY statement as a comment (an asterisk appears in the continuation column) and the text appears between two additional comment lines as:

   ***** Start of copy library text *****

   (text copied from copy library)

   ***** End of copy library text *****
3. EDIT (or SOS) line numbers are added (or replaced) to the copied text in the .CVT file for editing purposes (if the input file is line–sequenced).

With the .LST file, you can examine the COPY verb statement and its text for correctness. The text appears in the .LST file as it appears in the COPY library. No conversion on this text has taken place. In addition, any warning messages that pertain to the text for conversion also appears. This permits you to make the correct edits to the .CVT file.

In the .CVT file, the COPY verb statement becomes a comment, two additional comment lines are added (as shown above), the text is converted to COBOL–74 syntax, if needed, and the text becomes permanent source code. For example, a COBOL–68 program with:

```
0079   07900        COPY EXAMINE OF "COPYIT.LIB".
0080C               EXAMINE ALPHA–TEXT TALLYING ALL "Z".
```

appears in your .CVT file as:

```
07900   *        COPY EXAMINE OF "COPYIT.LIB".
07902   ***** Start of copy library text *****
07903              INSPECT ALPHA–TEXT TALLYING TALLY FOR ALL "Z".
07904   ***** End of copy library text *****
```

Thus, the .CVT file can be compiled as it now appears. If you wish, you can remove the comment lines before compiling your converted COBOL–68 program.

If your COBOL–68 source program has line numbers, then the copied text from the copy library must also have line numbers so that you can edit the copied text in the .CVT file. The COPY library text with line numbers, which when copied into your source program, is replaced with new line numbers generated from the last source line number, such as the COPY statement, adding one for each new line. If the copy library text does not have line numbers, the line numbers are generated automatically, adding one for each new line.

2.4 Reserved Word Considerations

Your COBOL–68 source program may contain one or more of the following words as a user–name:

ALSO	END–OF–PAGE	REFERENCES
BOTTOM	EOP	REMOVAL
CHARACTER	EXCEPTION	RMS
CODE–SET	INSPECT	SEPARATE
COLLATING	LINAGE	SORT–MERGE
DAY	LINAGE–COUNTER	STANDARD–1
DEBUGGING	NATIVE	START
DUPLICATES	ORGANIZATION	TOP
DYNAMIC	PRINTING	TRAILING
EBCDIC	PROCEDURES	

These words are reserved words in COBOL–74. When you convert a COBOL–68 program that contains any of the above words, the word is flagged as a reserved word in COBOL–74 with the following error message:

```
***        This is a new reserved word in COBOL–74.
```

Therefore, when this warning message appears in your .LST file, you must change the word in your .CVT file so that the COBOL–74 compiler does not produce an error.

Chapter 3
COBOL–68 to COBOL–74 Conversion

COBOL–68 to COBOL–74 conversions occur in either of the following ways:

1. The COBOL–68 syntax is automatically converted to COBOL–74 syntax.
2. The COBOL–68 syntax is not needed in the COBOL–74 program format and is converted into a comment by placing an asterisk (*) in the continuation field.

If the COBOL–68 source line contains more than one word to be converted or made into a comment, the source line is broken into two or more parts and each part is treated separately.

If the COBOL–68 syntax cannot be converted or made into a comment, a warning message appears below the syntax in the .LST file. Warning messages are described in Chapter 4.

The following four sections describe the conversions that occur in your COBOL–68 program when processed by the 68274 utility. To check which lines are converted, use the FILCOM program and specify the .CBL and .CVT files as input.

3.1 Identification Division Conversion

DATE–COMPILED.

All comment paragraph lines after the first line (DATE–COMPILED. comment) have an asterisk (*) inserted in the continuation field, since COBOL–74 compiler would otherwise delete the lines.

REMARKS.

All lines, including the first one (REMARKS. comment), have an asterisk (*) inserted in the continuation field, since the COBOL–74 compiler would otherwise delete the lines.

3.2 Environment Division Conversion

CONFIGURATION SECTION.

No conversion takes place, no action is required.

INPUT–OUTPUT SECTION.

No conversion takes place, no action is required.

FILE–CONTROL paragraph.

FOR MULTIPLE REEL/UNIT

This segment of the FILE–CONTROL paragraph is converted into a comment by insertion of an asterisk in the continuation field.

RESERVE integer ALTERNATE AREAS

The ALTERNATE phrase is deleted by the converter and the integer value is incremented by 2.

RESERVE NO ALTERNATE AREAS

This segment of the FILE–CONTROL paragraph is converted to RESERVE 1 AREA.

FILE–LIMITS clause

This segment of the FILE–CONTROL paragraph is converted into a comment by placing an asterisk in the continuation field.

ACCESS MODE IS SEQUENTIAL

This segment of the FILE–CONTROL paragraph is converted to ORGANIZATION IS SEQUENTIAL.

ACCESS MODE IS RANDOM

This segment of the FILE–CONTROL paragraph is converted to ORGANIZATION IS RELATIVE; ACCESS MODE IS DYNAMIC.

ACCESS MODE IS INDEXED

This segment of the FILE–CONTROL paragraph is converted to ORGANIZATION IS INDEXED; ACCESS MODE IS DYNAMIC.

PROCESSING MODE IS SEQUENTIAL

This segment of the FILE–CONTROL paragraph is converted to ACCESS MODE IS SEQUENTIAL.

ACTUAL KEY

This segment of the FILE–CONTROL paragraph is converted to RELATIVE KEY.

SYMBOLIC/NOMINAL KEY clause

This segment of the FILE–CONTROL paragraph is converted to a comment by placing an asterisk in the continuation field.

I–O–CONTROL paragraph.

No conversion takes place, no action is required.

3.3 Data Division Conversion

FILE SECTION.

LABEL RECORDS ARE record–name

This statement is converted to a comment by placing an asterisk in the continuation field. In addition, a warning message is provided to indicate an incompatible change.

DATA description entry

TALLY

The TALLY clause is no longer generated automatically. 68274 does not determine whether this clause is required. Thus, 68274 always generates the following TALLY clause as a DATA description entry:

```
01    TALLY PIC S9(5) COMP.
```

3.4 Procedure Division Conversion

EXAMINE

The EXAMINE statement is converted to an equivalent INSPECT statement as shown in the following formats:

1. EXAMINE identifier TALLYING ALL literal–1

   Converted to:

   INSPECT identifier TALLYING TALLY FOR ALL literal–1

2. EXAMINE identifier TALLYING ALL literal–1 REPLACING BY literal–2

   Converted to:

   INSPECT identifier TALLYING TALLY FOR ALL literal–1 REPLACING ALL literal–1 BY literal–2

3. EXAMINE identifier TALLYING LEADING literal–1

   Converted to:

   INSPECT identifier TALLYING TALLY FOR LEADING literal–1

4. EXAMINE identifier TALLYING LEADING literal–1 REPLACING BY literal–2

   Converted to:

   INSPECT identifier TALLYING TALLY FOR LEADING literal–1 REPLACING LEADING literal–1 BY literal–2

5. EXAMINE identifier TALLYING UNTIL FIRST literal–1

   Converted to:

   INSPECT identifier TALLYING TALLY FOR CHARACTERS BEFORE INITIAL literal–1

6. EXAMINE identifier TALLYING UNTIL FIRST literal–1 REPLACING BY literal–2

   Converted to:

   INSPECT identifier TALLYING TALLY FOR CHARACTERS BEFORE INITIAL literal–1 REPLACING CHARACTERS BY literal–2

7. EXAMINE identifier REPLACING ALL literal–1 BY literal–2

   Converted to:

   INSPECT identifier REPLACING ALL literal–1 BY literal–2

8. EXAMINE identifier REPLACING LEADING literal–1 BY literal–2

   Converted to:

   INSPECT identifier REPLACING LEADING literal–1 BY literal–2

9. EXAMINE identifier REPLACING FIRST literal–1 BY literal–2

   Converted to:

   INSPECT identifier REPLACING FIRST literal–1 BY literal–2

10. EXAMINE identifier REPLACING UNTIL FIRST literal–1 BY literal–2

    Converted to:

    INSPECT identifier REPLACING CHARACTERS BY literal–2 BEFORE INITIAL literal–1

GOBACK

This COBOL–68 statement is converted to EXIT PROGRAM.

NOTE

The NOTE statement is converted into a comment by placing an asterisk in the continuation field.

SEEK

The SEEK verb is converted into a comment by placing an asterisk in the continuation field. In addition, a warning message is provided to indicate an incompatible change.

USE LABEL PROCEDURE

The USE LABEL PROCEDURE statement is converted into a comment by placing an asterisk in the continuation field. In addition, a warning message is provided to indicate an incompatible change.

WRITE record–name

In COBOL–68, the WRITE statement defaults to BEFORE ADVANCING when writing ASCII files. Thus, the WRITE statement is converted to:

```
WRITE record–name BEFORE ADVANCING 1 LINE.
```

since the COBOL–74 default is AFTER 1 LINE.

Chapter 4
COBOL–68 to COBOL–74 Messages

The messages that can appear in your .LST file when you convert a COBOL–68 program to a COBOL–74 are either:

1. Those the program would get if compiled with COBOL–74, or
2. Those indicating impossible or incorrect conversions.

In addition, all COBOL–68 fatal and warning messages are output if your program contains illegal COBOL–68 syntax.

The following four sections describe the messages that can appear in your .LST file. Corrective action is programmer dependent.

4.1 Identification Division Messages

The only message that may occur in the IDENTIFICATION DIVISION of your converted COBOL–68 program is as follows:

If you have a hyphen (–) in the continuation field for a comment paragraph, such as REMARKS or SECURITY, the message:

Continuation of comment–entry by '–' is not permitted in COBOL–74.

appears below the comment line in the .LST file. You must change the hyphen to an asterisk in the continuation field in your .CVT file.

4.2 Environment Division Messages

CONFIGURATION SECTION.

SWITCH (n)

Switches are defined differently in COBOL–74. Therefore, all definitions of SWITCH are followed by the message:

SWITCHes are defined differently in COBOL–74.

CURRENCY SIGN IS

The characters "L", "/", and "=" cannot be specified in the CURRENCY SIGN clause. If one of these characters are specified, the following message appears:

Improper character for CURRENCY SIGN in COBOL–74.

INPUT–OUTPUT SECTION.

FILE–CONTROL paragraph.

ACTUAL KEY IS key

ACTUAL KEY is converted to RELATIVE KEY and the "key" must be an unsigned integer in COBOL–74. If the COBOL–68 program has a signed key integer, the following message appears:

Should be unsigned integer in COBOL–74.

I–O–CONTROL paragraph.

No messages are generated for the I–O–CONTROL paragraph.

4.3 Data Division Messages

FILE SECTION.

LABEL RECORDS ARE record–name

The LABEL RECORDS ARE record–name clause in COBOL–68 is no longer legal in COBOL–74. If this clause exists, the following message appears in the .LST file:

Non–standard labels are illegal in COBOL–74.

DATA description entry

level–number

When you specify a group item in your COBOL–68 program, all data items subordinate to the group item data–name may have different level numbers. However, COBOL–74 does not allow this to occur. For example,

```
01    PRINT–OUTPUT–RECORD.
      03    LINE–ONE PIC X(30).
      02    LINE–TWO PIC X(30).
                .
                .
                .
      02    LINE–TEN PIC X(30).
```

is valid source code in COBOL–68. The above code, when converted, generates the following warning message:

All items that are immediately subordinate to a group item must have the same level–number in COBOL–74.

BLANK WHEN ZERO

The BLANK WHEN ZERO clause and an asterisk (*) as zero suppression in a PICTURE clause cannot appear in the same entry. For example:

```
02      ITEM–NUMBER      PIC ****** BLANK WHEN ZERO
```

is illegal in COBOL–74. If this situation exists, the following message appears in the .LST file:

BLANK WHEN ZERO and '*' not allowed together in COBOL–74.

TODAY

The TODAY entry in COBOL–68 can no longer be generated automatically in COBOL–74. For example, in COBOL–68 you may have the statement:

```
MOVE TODAY TO CURRENT–DATE.
```

The contents of CURRENT–DATE, in the WORKING–STORAGE SECTION, could be:

```
01      CURRENT–DATE      PIC X(12).
```

The contents of TODAY (CURRENT–DATE) is in the format of 'yymmddhhmmss', where:

yy = year
mm = month
dd = day
hh = hour
mm = minute
ss = second

If this entry exists in your COBOL–74 program, the following message appears:

Replace TODAY with ACCEPT FROM DAY, DATE, or TIME as appropriate.

A possible method of constructing TODAY in COBOL–74 is to create the following entries:

```
01    TODAY.
      02    NEW–TODAY–DATE     PIC X(6).
      02    NEW–TODAY–TIME     PIC X(6).
```

Then in the COBOL–74 PROCEDURE DIVISION, you can create the following statements to obtain the date and time:

```
ACCEPT NEW–TODAY–DATE FROM DATE.
ACCEPT NEW–TODAY–TIME FROM TIME.
```

VALUE

The VALUE clause in the PICTURE clause is initialized independent of the JUSTIFIED clause. If this is different from COBOL–68, the JUSTIFIED clause is followed by the following message in the .LST file:

The VALUE clause is initialized independent of the JUSTIFIED clause in COBOL–74.

4.4 Procedure Division Messages

Abbreviated Combined Relation Condition

If your COBOL–68 source program has an abbreviated combined relation condition containing the NOT clause, the following message may be generated:

NOT is abbreviated combined relation conditions may generate different code.

For example, an IF test of the expression:

A > B AND NOT < C OR D

may be different for COBOL–74. If the above message occurs, the best solution is to expand the expression using parenthesis, where required.

MOVE LOW–VALUE TO identifier

If the statement MOVE LOW–VALUE TO identifier exists in your COBOL–68 and the identifier is one of the following keys:

- ACTUAL KEY (for a RANDOM file)
- SYMBOLIC KEY (for an ISAM file)
- RECORD KEY (for an ISAM file)

or a data–item subordinate to it, then the MOVE LOW–VALUE TO statement should be deleted and the next READ or RETAIN verb should be replaced, with its approximate key, by one of the following:

- READ NEXT
- RETAIN NEXT
- START

The MOVE LOW–VALUE TO "some–key" produces the following message:

Replace LOW–VALUE hack with appropriate COBOL–74 syntax.

in your .LST file.

STRING/UNSTRING

If your COBOL–68 program has consecutive occurrences of either of the following statements:

STRING DELIMITED BY ALL literal/data–name DELIMITER IN item–1

or

UNSTRING DELIMITED BY ALL literal/data–name DELIMITER IN item–1

and item–1 is larger than DELIMITER, then the following message appears in your .LST file:

Contents of DELIMITER may be different in COBOL–74.

The converted COBOL–74 program contains only one occurrence of DELIMITER. (Your COBOL–68 program contains the actual text of the delimiter.)

Unsigned Integer

If your COBOL–68 program is using either:

- DISPLAY literal
- STOP literal

and literal is a signed integer, the following warning message appears in your .LST file:

Numeric literal must be unsigned integer in COBOL–74.

The integer must be changed to an unsigned integer for COBOL–74.

Appendix A
Differences Between COBOL–68 and COBOL–74

The terms COBOL–68 and COBOL–74, which are used in the following text, refer to DIGITAL's implementation of ANS–68 and ANS–74 COBOL, respectively. Any references to ANS COBOL are made clear by the use of the initials "ANS".

The symbols within angle brackets (<>) following the differences listed below represent:

- The module affected (nNUC, nSEG, nTBL, nSRT, RPW, nSEQ, nREL, nINX, nDEB, nIPC, nLIB, and nCOM).
- A change that does not impact existing programs (1).
- A change that could impact existing programs and/or some re-programming may be needed (2).
- General remarks with ANS numbers, where applicable.

COBOL–74 differs from COBOL–68 in the following ways:

1. Two contiguous quotation marks can be used to represent a single quotation mark character in a non–numeric literal. <1NUC (1) New feature.>
2. REMARKS paragraph is deleted. <1NUC (2) Function was replaced by the comment line.>
3. Continuation of Identification Division comment–entries must not have a hyphen in the continuation indicator area. <1NUC (2)>
4. PROGRAM COLLATING SEQUENCE clause specifies that the collating sequence associated with alphabet–name is used in non–numeric comparisons. <1NUC (1) New feature.>

5. SPECIAL–NAMES paragraph: "L", "/", and "=" cannot be specified in the CURRENCY SIGN clause. <2NUC (2) This restriction did not exist in X3.23–1968.>

6. Alphabet–name clause relates a user–defined name to a specified collating sequence or character code set (ANSI, native, or implementor–specified). <1NUC (1) New feature.>

7. Alphabet–name clause: the literal phrase specifies a user–defined collating sequence. <2NUC (1) New feature.>

8. All items that are immediately subordinate to a group item must have the same level–number. <1NUC (2)>

9. Object of a REDEFINES clause can be subordinate to an item described with an OCCURS clause, but must not be referred to in the REDEFINES clause with a subscript or an index. <1NUC (1) New feature.>

10. An asterisk used as a zero suppression symbol in a PICTURE clause and the BLANK WHEN ZERO clause cannot appear in the same entry. <1NUC (2)>

11. Alphabetic PICTURE character–string can contain the character B. <1NUC (1) New feature.>

12. Stroke (/) permitted as an editing character. <1NUC (1) New feature.>

13. SIGN clause allows the specification of the sign position. <1NUC (1) New feature.>

14. In the Procedure Division a section can contain zero or more paragraphs and a paragraph can contain zero or more sentences. <1NUC (1) New feature.>

15. In relation and sign conditions, arithmetic expressions must contain at least one reference to a variable. <1NUC (2)>

16. Comparison of non–numeric operands: If one of the operands is described as numeric, it is treated as though it were moved to an alphanumeric item of the same size and the contents of this alphanumeric item were then compared to the non–numeric operand. <1NUC (3)>

17. Abbreviated combined relation condition: When any portion is enclosed in parentheses, all subjects and operators required for the expansion of that portion must be included within the same set of parentheses. <2NUC (2) No such restriction appeared in X3.23–1968.>

18. Abbreviated combined relation condition: If NOT is immediately followed by a relational operator, it is interpreted as part of the relational operator. <2NUC (2) In X3.23–1968, NOT was a logical operator in such cases.>

19. Class condition: The numeric test cannot be used with a group item composed of elementary items described as signed. <1NUC (3)>

20. In an arithmetic operation, the composite of operands must not contain more than 18 decimal digits. However, if your COBOL–74 compiler makes use of the Business Instruction Set, the maximum is 36 digits. <1NUC (2) X3.23–1968 specified limits only for ADD and SUBTRACT.>

21. ACCEPT identifier FROM DATE/DAY/TIME allows the programmer to access the date, day, and time. <2NUC (1) New feature.>

22. COMPUTE statement: the identifier series. <2NUC (1) New feature.>

23. DISPLAY statement: If the operand is a numeric literal, it must be an unsigned integer. <1NUC (2)>

24. DIVIDE statement: the INTO identifier series and the GIVING identifier series. <1NUC (2)>

25. DIVIDE statement: The remainder item can be numeric–edited. <2NUC (1) New feature.>

26. EXAMINE statement and the special register TALLY were deleted. <1NUC (2) Function was replaced by the INSPECT statement.>

27. INSPECT statement provides ability to count or replace occurrences of single characters or groups of characters. <1NUC (1) New feature.>

28. MOVE statement: A scaled integer item (that is, the rightmost character of the PICTURE character–string is a P) can be moved to an alphanumeric or alphanumeric–edited item. <1NUC (1) New feature.>

29. MULTIPLY statement: the BY identifier series and the GIVING identifier series. <2NUC (1) New feature.>

30. PERFORM statement: There is no logical difference needed between fixed and fixed overlayable segments. <1NUC (1) X3.23–1968 did not permit fixed overlayable segments to be treated the same as a fixed segments.>

31. PERFORM statement: Control is passed only once for each execution of a Format 2 PERFORM statement (that is, an independent segment referred to by such a PERFORM is made available in its initial state only once for each execution of that PERFORM statement). <1NUC,1SEG (3)>

32. STOP statement: If the operand is numeric literal, it must be an unsigned integer. <1NUC (2)>

33. A data description entry with an OCCURS DEPENDING clause can be followed within that record only by entries subordinate to it. That is, only the last part of the record can have a variable number of occurrences. <2TBL (2) This rule did not appear in X3.23–1968.>

34. When a group item, having subordinate to it an entry that specifies Format 2 of the OCCURS clause, is referenced, only that part of the table area that is defined by the value of the operand of the DEPENDING phrase is used in the operation. That is, the actual size of a variable length item is used, not the maximum size. <2TBL (2)>

35. The subject of the condition in the WHEN phrase of the SEARCH ALL statement must be a data item named in the KEY phrase of the table; the object of this condition cannot be a data item named in the KEY phrase. <2TBL (2) X3.23–1968 specified that either the subject or object could be a data item named in the KEY phrase.>

36. SORT statement: COLLATING SEQUENCE phrase provides the ability to override the program collating sequence. <2SRT (1) New feature.>

37. No more than one file–name from a multiple file reel can appear in a SORT statement. <2SRT (2)>

38. Segment–numbers are permitted in DECLARATIVES. <1SEG (1)>

39. ACCESS MODE IS DYNAMIC clause: provides the ability to access a file sequentially or randomly in the same program. <2REL,2INX (1) New feature.>

40. ACTUAL KEY clause deleted. <(2)>

41. RELATIVE KEY clause added for relative organization. <1REL (1) New feature.>

42. FILE–LIMITS clause deleted. <(2)>

43. PROCESSING MODE clause deleted. <(2)>

44. ORGANIZATION IS RELATIVE clause. <1REL (2) New feature.>

45. ORGANIZATION IS SEQUENTIAL clause. <1SEQ (2) New feature.>

46. ORGANIZATION IS INDEXED clause. <1INX (2) New feature.>

47. MULTIPLE REEL/UNIT clause deleted. <(2)>

48. RESERVE...ALTERNATE AREAS deleted. <(2)>

49. RESERVE integer AREAS specifies the exact number of areas to be used. <1SEQ,1REL,1INX (1) New feature.>

50. The data–name option of the LABEL RECORDS clause deleted. <1SEQ,1REL,1INX (2) X3.23–1968 provided for user–defined label records.>

51. LINAGE clause permits programmer definition of logical page size. <2SEQ (1) New feature.>

52. CLOSE...FOR REMOVAL statement. <2SEQ (1) New feature.>

53. DELETE statement. <1REL (1) New feature.>

54. OPEN REVERSED positions the file at its end. <2SEQ (2)>

55. OPEN EXTEND statement adds records to an existing file. <2SEQ (1) New feature.>

56. The OPEN REVERSED statement applies to all devices that claim support for this function. <2SEQ (1) X3.23–1968 restricted the application of this phrase.>

57. READ statement: AT END phrase required only if no applicable USE AFTER ERROR/EXCEPTION procedure specified. <1SEQ,1REL,1INX (1) New feature.>

58. READ statement: INVALID KEY phrase required only if no applicable USE AFTER ERROR/EXCEPTION procedure specified. <1REL,1INX (1) New feature.>

59. READ...NEXT statement retrieves the next logical record from a file when the access mode is dynamic. <2REL, 2INX (1) New feature.>

60. REWRITE statement. <1SEQ,1REL (1) New feature.>

61. SEEK statement deleted. <(2)>

62. START statement provides for logical positioning within a relative or indexed file for sequential retrieval of records. <2REL, 2INX (1) New feature.>

63. USE statement: the label processing options were deleted. <1SEQ,1REL,1INX (2) X3.23–1968 provided for the processing of user–defined labels.>

64. USE...ERROR/EXCEPTION statement. <1SEQ,1REL,1INX (1) New feature.>

65. Recursive invocation of USE procedures prohibited. <1SEQ,1REL,1INX (2)>

66. WRITE statement: INVALID KEY phrase required only if no applicable USE AFTER ERROR/EXCEPTION procedure specified. <1REL,1INX (1)>

67. WRITE statement: BEFORE/AFTER PAGE phrase provides ability to skip to top of a page. <1SEQ (1)>

68. WRITE statement: END–OF–PAGE phrase. <2SEQ (1) New feature.>

69. CALL identifier statement. <1IPC (1) New feature.>

70. EXIT PROGRAM statement replaces the GOBACK statement.

Note A. (RELATIVE files)

The RANDOM file access method of COBOL–68 has been replaced by the RELATIVE file organization in COBOL–74. This means a number of syntactic changes, but in addition it means some important semantic changes as well.

In the Environment Division, the syntactic changes include the substitution of an ORGANIZATION IS RELATIVE clause for the old ACCESS IS RANDOM clause, and the substitution of the ACCESS IS SEQUENTIAL / RANDOM / DYNAMIC for the old PROCESSING IS SEQUENTIAL clause. The FILE LIMITS clause goes away. The ACTUAL KEY clause is replaced by the RELATIVE KEY clause, although the meaning of the key value is identical to that in COBOL–68.

The Data Division is unchanged.

The Procedure Division verbs are changed considerably. OPEN, CLOSE and the USE ON ERROR procedures are unchanged. The WRITE statement is unchanged in syntax, but its meaning is restricted to writing a record into an empty position in the file. If the record position in the file into which the record is being written is already occupied, the WRITE must not alter the existing contents of the record position, but must instead take the INVALID KEY path (or execute a USE procedure). In order to change the contents of an occupied record position one either has to REWRITE it or DELETE and WRITE it. Attempting to DELETE or REWRITE a record position that is already empty causes the INVALID KEY path to be taken. In other words, each record position of the relative file must have an occupied state that can be recognized by the object time I/O routines. There is also a START verb that can be used to position at or beyond a given record position. Then sequential READs or WRITEs can be done. The sequential READ is done with a READ NEXT statement, whereas the random READ is simply a READ statement. The sequential READ uses the AT END phrase (which is optional) and the random READ uses the INVALID KEY phrase (also optional). Thus, there are not only many syntactic changes in existing verbs, but new verbs, and a markedly different approach to the file's contents.

Note B. (INDEXED files)

The INDEXED I/O module of COBOL–74 is fairly similar to that of COBOL–68. There are syntactic differences in the Environment Division and in the Procedure Division.

COBOL–68 had a SYMBOLIC KEY clause to designate the key used in READ, WRITE, REWRITE and DELETE statements. COBOL–74 does not have a SYMBOLIC KEY clause. The random READ statement has a KEY IS identifier phrase that supplies a key value. The WRITE statement uses key values from the record being written, and the DELETE and REWRITE statements must follow a successfully executed READ and use the remembered key from that operation.

COBOL–74 includes a START statement in the Procedure Division that positions the record pointer in the file specified. Also, the READ NEXT statement is used to do sequential reading through the existing records of the file.

Note C. (Segmentation and PERFORM rules)

In COBOL–74, sections in the Procedure Division can have segment numbers (called priority numbers in COBOL–68) that range from 00 to 99. Segments with numbers 50 and above are called independent segments. Also, the programmer can specify a SEGMENT LIMIT IS clause with a value between 00 and 49. This divides the segments with numbers below 50 into two groups. Thus all segments fall into one of three groups:

1. Below the segment limit, called fixed permanent, that are always resident.
2. From the segment limit to 49, called fixed overlayable; that is, each segment number defines an overlay and the code in such a segment is brought into memory only as needed. Any GO TOs that have been ALTERED retain their most recently set values when they are brought into memory.
3. From 50 up, called independent; that is, each segment number defines an overlay and the code is brought into memory only as needed. Any GO TOs that have been altered are reset each time the segment is brought into memory.

The restrictions on the ALTER and PERFORM verbs have not really changed from ANS–68 COBOL to ANS–74 COBOL but they have become more explicit. COBOL–74 implements the restrictions on the ALTER statement correctly (by either standard) but implements the restrictions on PERFORM in a manner different from either standard. COBOL–74 uses the segment–limit value as the dividing line for the PERFORM restrictions, whereas the standards use the segment number 50 as the dividing line. When you do not specify a segment limit value the compiler supplies 50 as the default, making the restrictions the same for COBOL–74 and the standards. However, when you do supply the segment limit value, COBOL–74 applies the rules in such a way as to make all overlayable segments behave the same.

Note D. (CALL and CANCEL rules)

There are many differences between the COBOL–74 implementation of CALL and CANCEL and the ANS–74 COBOL standard.

1. The syntax is different for both statements in that COBOL–74 interprets a user–word as a program–name with or without quotes around it, whereas ANS–74 COBOL interprets a user–word as a data–name in which is stored the program–name.

2. In ANS–74 COBOL there is an ON OVERFLOW ... clause for handling instances in which there is insufficient memory space available to load the called subprogram. This cannot happen in COBOL–74.

3. COBOL–74 allows alternate entry points to subprograms, not allowed in ANS–74 COBOL, and COBOL–74 uses the ENTER (MACRO/FORTRAN) statement to allow you to call subprograms written in those languages.

4. The semantics are very different. COBOL–74 uses LINK to construct a tree–structured overlay scheme from user–supplied commands to LINK. When a subprogram is CALLed, the branch of the tree up to that subprogram is loaded along with the subprogram. Likewise, when a subprogram is CANCELled, the entire tree beyond that subprogram is cancelled. ANS–74 COBOL recognizes no such tree structure, and allows loading and cancelling to occur strictly on a subprogram basis. In addition, LINK allows more than one subprogram to be linked into a single overlay, with the effect that a cancel of one of the subprograms in the overlay results in a cancel of all subprograms in that overlay.

Index

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____ Telephone _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear – Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MR1–2/L12
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear – Fold Here and Tape

Cut Along Dotted Line