COBOL INFORMATION BULLETIN

NUMBER 24

INTERPRETATIONS OF ANSI X3.23-1985

OCTOBER 1988

NOTICE

Recognizing the need for a uniform approach to the responsibility for disseminating interpretations to approved American National Standards, the Accredited Standards Committee, X3 Information Processing Systems -- has authorized the publication of COBOL Information Bulletins.

This COBOL Information Bulletin is issued in response to questions which have been raised regarding certain specifications contained in X3.23-1985 - American National Standard Programming Language COBOL.

This Bulletin was prepared by Technical Committee X3J4, which developed that Standard and was authorized for release by committee X3 in order to provide interpretations as quickly as possible in response to questions raised.

This Bulletin, while reflecting the technical opinion of the Technical Committee which developed the standard, is intended solely as supplementary information to other users of the standard. That standard, ANSI X3.23-1985, as approved through the publication and voting procedures of the American National Standards Institute is not altered by this Bulletin. Any subsequent revision to the standard, ANSI X3.23-1985, may or may not reflect the contents of this COBOL Information Bulletin.

TABLE OF CONTENTS

SECTION 1: INTRODUCTION

1.1 COBOL INFORMATION BULLETINS

COBOL Information Bulletins (CIBs) are published by the Computer and Business Equipment Manufacturers Association (CBEMA) on behalf of the X3J4 COBOL Technical Committee to provide a method of interacting with COBOL users who are involved with ANSI COBOL X3.23-1985 and are interested in the work of X3J4.

This COBOL Information Bulletin contains the first interpretations to ANSI COBOL X3.23-1985 resulting from inquiries received and processed by X3J4 since publication of ANSI COBOL X3.23-1985. A comprehensive index of all interpretations of ANSI COBOL X3.23-1985 begins on page 69.

Copies of the COBOL Information Bulletin can be obtained by corresponding directly with:

CBEMA
311 First Street N. W.
Washington, D. C. 20001

Copies of American National Standard COBOL X3.23-1985 can be obtained by corresponding directly with:

American National Standard Institute, Inc.
1430 Broadway
New York, New York 10018

1.2 X3J4 SCOPE AND PROGRAM OF WORK

An outline of the current scope and program of work of the X3J4 COBOL Technical Committee is being provided in order that the reader of this COBOL Information Bulletin can understand the type of work that X3J4 does and the tasks which it hopes to accomplish.

The scope of X3J4 can be divided into three distinct areas. The first part of the scope is to provide a mechanism for the solicitation and review of all activities regarding the current national COBOL Standard. The second part of the scope is to carry out the procedures necessary to maintain the continued responsiveness of the COBOL Standard to user needs. The third part of the scope calls for the continued support and publication of the COBOL Information Bulletin to interact with the COBOL community and disseminate relevant information about the status of the COBOL Standard, and any clarifications which may impact implementation of COBOL compilers.

The program of work for X3J4 includes several activities of interest which are outlined below:

- Support the current COBOL Standard by responding to inquiries about the Standard and requests for clarifications.

- Support the COBOL language as defined in both the CODASYL COBOL Journal of Development and the COBOL Standard to determine what a subsequent revision to the current COBOL Standard might contain or, more importantly, what it might not contain.

- Maximize compatibility of any future revision of the COBOL Standard with other American National Standards.

1.3 X3J4 RELATIONSHIP WITH ANSI AND CBEMA

The X3J4 COBOL Committee is the technical committee for COBOL operating under the Accredited Standards Committee X3 on Information Processing Systems. X3 is a committee that has under it the standardization activities that are concerned with all aspects of information processing systems. The secretariat for X3, i.e., the body that supports the coordination and administration for both X3 and its related technical committees, is the Computer and Business Equipment Manufacturers Association (CBEMA).

1.4 RELATIONSHIP BETWEEN X3J4 AND INTERNATIONAL ORGANIZATIONS

Throughout the COBOL standardization activity, close liaison with the various international groups has been maintained. In fact, the impetus for the entire undertaking of standardization in the area of computers and information processing can be traced directly to international sources. The American National Standards Institute represents the United States in the joint international committee of the International Standards Organization (ISO) and the International Electrotechnical Commission (IEC); this joint international committee is called ISO/IEC Joint Technical Committee 1 (JTC1), Information Processing Systems. ISO/IEC JTC1 Subcommittee 22 (SC22) is an ISO/IEC JTC1 subcommittee for application system environments and programming languages. Members of the X3J4 COBOL Technical Committee participate actively in meetings of the COBOL working group WG4 associated with ISO/IEC JTC1 Subcommittee 22. Thus, influence of and requirements for international considerations are present in the COBOL Standard.

1.5 RELATIONSHIP BETWEEN X3J4 AND CODASYL COBOL COMMITTEE

ANSI COBOL X3.23-1985 was derived from the CODASYL COBOL Journal of Development and the previous COBOL Standard, ANSI COBOL X3.23-1974. The CODASYL COBOL Journal of Development is the product of the CODASYL COBOL Committee (CCC), a separate distinct body from the X3J4 COBOL Technical Committee. Because these two entities are frequently confused in the public mind, it is appropriate to briefly describe each one and its function.

1.5.1. CODASYL COBOL Committee (CCC)

The CODASYL COBOL Committee is a committee within CODASYL (Conference on Data Systems Languages). It is the development body that both originates and maintains the specifications of the COBOL language. These are reflected in the CODASYL COBOL Journal of Development, which, as its name suggested, is a continually evolving document. The CODASYL COBOL Committee may be contacted by writing to:

Chairman, CODASYL COBOL Committee
50 Presidents Lane
Quincy, Massachusetts 02169

1.5.2 X3J4 COBOL Technical Committee

The X3J4 COBOL Technical Committee is concerned with standardizing COBOL from the output of the CODASYL COBOL Committee. X3J4 takes as its basic input the CODASYL COBOL Journal of Development as of a certain point in time and "freezes" it prior to molding it into a format suitable for a formal standard specification. X3J4 can be contacted by writing to:

Chairman, X3J4
CBEMA
311 First Street N. W.
Washington, D. C. 20001

1.5.3 Impact of CODASYL COBOL on ANSI COBOL X3.23-1985

When the X3J4 COBOL Technical Committee updates the COBOL Standard, ANSI COBOL X3.23-1985, the current method of operation permits inclusion of additional language specifications only from the CODASYL COBOL Journal of Development. This is unlike the other X3 language standardization committees which perform both the development and the standardization functions.

SECTION 2: RESPONSES TO INTERPRETATION REQUESTS

2.1 INTRODUCTION

This section of the COBOL Information Bulletin contains information related to actions taken by X3J4 in regard to questions that have been asked concerning American National Standard COBOL X3.23-1985 since its publication in 1985. The purpose of providing this information is to keep the public informed as to the current thinking and philosophy of the committee in regard to questions or discussions which involve American National Standard COBOL X3.23-1985.

Each of the responses is presented in a form showing the reference in American National Standard COBOL X3.23-1985, the question, and the response of the X3J4 COBOL Technical Committee. The reference number prefixed by the letter A is the document number used by X3J4 to identify the response to the question.

If the response to a question results in X3J4 formulating a proposed correction to American National Standard COBOL X3.23-1985, then X3J4's interpretation document includes a cross reference to the proposed correction as documented in the X3J4 working document called the Draft Proposed Correction Addendum to X3.23-1985 (see Section 3 beginning on page 61).

An index of all interpretations made relative to American National Standard COBOL X3.23-1985 is located in Section 5 beginning on page 69. This index is arranged in order by subject matter within American National Standard COBOL X3.23-1985.

2.2 DISCLAIMER

Recognizing the need for a uniform approach to the responsibility for disseminating the interpretations to approved American National Standards, the Accredited Standards Committee on Information Processing Systems, X3, has authorized the publication of COBOL Information Bulletins.

These interpretations are issued in response to questions that have been raised regarding certain specifications contained in American National Standard COBOL, X3.23-1985.

These interpretations were prepared by X3J4, a technical committee of X3, that developed American National Standard COBOL X3.23-1985, and were authorized for release by X3 in order to provide interpretations as quickly as possible in response to questions raised.

These interpretations, while reflecting the technical opinion of X3J4, are intended solely as supplementary information to users of American National Standard COBOL X3.23-1985. American National Standard COBOL X3.23-1985 was approved through the publication and voting procedures of the American National Standards Institute and is not altered by this bulletin. Any subsequent revision of American National Standard COBOL X3.23-1985 may or may not reflect the content of these interpretations.

X3J4 DOCUMENT A-1

SUBJECT: INSPECT TALLYING Statement

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VI-96, paragraph 6.18.4, general rule 2c, INSPECT statement

DATE: April 21, 1986

QUESTION:

Given the following example, is it true that COUNT-1 will contain the value 3, and COUNT-2 the value 2?

```
02  A  PICTURE XXX.
02  B  REDEFINES A  PIC S99  SIGN  LEADING SEPARATE.

INSPECT A TALLYING COUNT-1 FOR CHARACTERS.

INSPECT B TALLYING COUNT-2 FOR CHARACTERS.
```

(COUNT-1 and COUNT-2 are zero before execution of these INSPECT statements.)

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that COUNT-1 will contain the value 3 and COUNT-2 will contain the value 2 because a signed numeric data item is inspected as though it has been moved to an unsigned numeric data item with length equal to the length of the signed item excluding any separate sign position.

X3J4 DOCUMENT A-2

SUBJECT: Scope of Program-Names

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page X-5, paragraph 1.3.8.1, conventions for program-names
2. Page X-28, paragraph 5.2.4, general rule 4, CALL statement
3. Page II-24, paragraph 6.4.1.3, scope of names of programs

DATE: May 23, 1986

QUESTION:

The "conventions for program-names" (see reference 1) allow for a separately compiled program and a program contained within a different separately compiled program to have the same name. There are additional rules covering legal references to a contained program. Consider the following example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  A.
 ...
 ...
END-PROGRAM A.
IDENTIFICATION DIVISION.
PROGRAM-ID.  B.
PROCEDURE DIVISION.
P.  CALL "A".
 ...
 ...
IDENTIFICATION DIVISION.
PROGRAM-ID.  C.
 ...
 ...
IDENTIFICATION DIVISION.
PROGRAM-ID.  A.
 ...
 ...
END-PROGRAM A.
END-PROGRAM C.
END-PROGRAM B.
```

The call to A from B cannot be a legal call to the program A contained in program C by rule 1 of paragraph 1.3.8.1 (see reference 1). Is it a legal call to the separate A or an illegal call to the contained A? Note that if it is intended to be an illegal call to the contained A, the illegality cannot be expressed in a syntax rule because the call could have been a "CALL identifier" where the error is not known until run time.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the call to A from B is a legal call to the separate A (see references 1, 2, and 3).

X3J4 DOCUMENT A-3

SUBJECT: INITIAL Clause of CD Entry and Contained Programs

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page II-18, paragraph 6.1, program and run unit organization
2. Page II-31, paragraph 7.3.1, invoking the COBOL object program
3. Page III-15, definition of object program
4. Page III-23, definition of source program
5. Page IV-35, paragraph 6.4.1.3, execution
6. Page X-5, paragraph 1.3.8.1, conventions for program-names
7. Page X-8, paragraph 2, nested source programs
8. Pages XIV-3 and 4, paragraph 2.2.2, general format for CD entry
9. Pages XIV-4 through 6, paragraph 2.2.3, syntax rules for CD entry
10. Page XIV-8, paragraph 2.2.4, general rule 6, CD entry

DATE: April 9, 1986

QUESTION:

Syntax rule 2 of the communication description entry specifies that the INITIAL clause must not be specified for a CD in a program that specifies the USING phrase in the Procedure Division header and that only one CD in a program may have an INITIAL clause. It does not specify whether the INITIAL clause is allowed on a CD in a contained program which does not have a USING phrase in the Procedure Division header or in more than one such contained program. I suspect that it is an oversight that this is not disallowed, but it could be interpreted that the separate program containing the one with the initial clause is the one which the MCS should schedule.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the INITIAL clause is allowed in a communication description entry (CD entry) in a contained program (see references 1, 3, 4, 7, 8, and 9). The INITIAL clause may appear in only one CD entry within an entire separately compiled source program, including CD entries in any of its contained programs (see references 3, 4, and 9). When the MCS schedules an object program, execution begins with the first statement of the Procedure Division, excluding declaratives, of the separately compiled source program from which the object program was derived (see references 1, 2, 5, 6, 10). It is irrelevant where the CD entry with the INITIAL clause is defined, if one is defined, whether directly within the separately compiled program or within a program contained within the separately compiled program (see references 1, 2, 5, 7, and 10).

X3J4 DOCUMENT A-4

SUBJECT: File Attribute Conflict

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VII-8, paragraph 2.3.4, general rule 1, file control entry
2. Page VIII-8, paragraph 2.3.4, general rule 1, file control entry
3. Page IX-9, paragraph 2.3.4, general rule 1, file control entry

DATE: March 14, 1986

QUESTION:

There is a problem with the definition of file attribute conflict condition. Paragraph 1.3.7 on page VII-5 of the Sequential I-O module states that this condition can occur only on OPEN, WRITE, and REWRITE, but the only applicable file status is 39 which refers only to OPEN. Consider the case of an EXTERNAL file which is defined as sequential, for example, in one program which opens the file, and as indexed in a called program which reads the file. Surely this is a file attribute conflict but what status code should be returned?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example given is not a legal program because it violates general rule 1 of the file control entry (see references 1, 2, and 3).

The results of executing an illegal program are not defined in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-5

SUBJECT: COPY REPLACING Statement

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page III-25, definition of text word
2. Page XII-3, paragraph 2.4, general rule 5, COPY statement

DATE: April 7, 1986

QUESTION:

A COBOL library A contains the following text:

```
02  (I)-ITEM  PICTURE X.
```

What is the result of:

```
COPY A REPLACING ==(I)== BY ==RECORD-1==.
```

The suggestion is:

```
02  RECORD-1-ITEM  PICTURE X.
```

Page III-25, glossary definition of text word:

- the left and right parenthesis characters are always considered text words.
- any other sequence of contiguous COBOL characters is a valid text word.

So both "I" and "RECORD-1" are valid text words. This makes the first pseudo-text a valid series of text words.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the suggested replacing would take place. A replacing match would occur as the text words (see reference 1) in pseudo-text-1 would match, character for character, to the ordered sequence of text words found in the library text (see reference 2).

X3J4 DOCUMENT A-8

SUBJECT: Declarative Procedures

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VII-2, paragraph 1.3.5, I-O status
2. Page VII-4, paragraph 1.3.5, item 4a, I-O status 41
3. Page VII-51, paragraph 4.6.4, general rule 5, USE statement

DATE: May 2, 1986

QUESTION:

A COBOL program contains a declarative procedure for all input files and a declarative procedure for all output files. Nondeclarative statements are:

```
OPEN OUTPUT FILE-K9.
 ...
WRITE FILE-K9-REC.
 ...
OPEN INPUT FILE-K9.
```

After the OPEN OUTPUT FILE-K9 statement and the WRITE FILE-K9-REC statement have been executed, what will happen when an attempt is made to execute the statement OPEN INPUT FILE-K9? Will the input declarative procedure or the output declarative procedure be executed as a result of attempting the OPEN INPUT FILE-K9 statement?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example represents a logic error condition with unsuccessful completion. An I-O status of 41 is returned (see reference 2). An I-O status of 41 is a critical error condition and the implementor determines what action is taken after the execution of any applicable USE AFTER STANDARD EXCEPTION procedure (see reference 1). In this example, one of the declarative procedures is executed, however which declarative procedure, input or output, is not specified in Standard COBOL (see reference 3).

X3J4 DOCUMENT A-10

SUBJECT: CLOSE REEL/UNIT and File Position Indicator

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VII-37, paragraph 4.2.4, general rule 3F, CLOSE REEL/UNIT

DATE: September 3, 1986

QUESTION:

Reference 1 does not define any effect on the file position indicator when a CLOSE statement with the REEL or UNIT phrase is executed for a file open in the INPUT or I-O mode. This raises the question of whether ANSI X3.23-1985 indicates that the file position indicator is unchanged by a CLOSE statement with the REEL or UNIT phrase or that the effect on the file position indicator in this case is undefined. If the file position indicator is undefined after such a CLOSE statement, implementors could choose to implement the CODASYL definition. However, if X3.23-1985 by omission means the file position indicator is unchanged, the statement when executed other than at the end of a reel or unit simply causes a couple of reel swaps, a rewind, and a reposition.

Is the file position indicator defined following the execution of a CLOSE statement with the REEL or UNIT phrase, and, if defined, what is its definition?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the status of the file position indicator following the execution of a CLOSE statement with the REEL/UNIT phrase is undefined.

PROPOSED CORRECTION TO X3.23-1985:

X3J4 has proposed a correction to ANSI COBOL X3.23-1985 as a result of the processing of the question in document A-10. Pages 62 and 63 of this document contain the proposed corrections to pages VII-37 and VII-38 in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-11

SUBJECT: Extension Language Elements and Conforming Implementations

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page I-8, paragraph 1.5.2.5.2, third paragraph, extension language elements

DATE: August 31, 1986

QUESTION:

Request for official interpretation of page I-8, paragraph 1.5.2.5.2, Extension Language Elements, third paragraph which reads as follows:

"A conforming implementation of Standard COBOL must provide a warning mechanism, which optionally may be invoked by the user at compile time to indicate, if appropriate, that a program contains nonstandard extensions that are included in the implementation."

To put the whole discussion more concrete, let me illustrate the matter by the following examples:

Example 1:

```
PERFORM ... VARYING IDENT-2 FROM IDENT-3 BY IDENT-4 UNTIL ...
```

It is obviously a nonstandard extension if the implementation allows the BY variable IDENT-4 being equal to 0 (i.e. in the moment where it is added to the VARYING variable in accordance with the PERFORM logic). This extension is quite useful and not far fetched. On first sight one would say: How should the compiler know the value of the BY variable at run time, in particular, if it has again a variable subscript! At run time this can be tested since the compiler can, of course, generate checks into the PERFORM code as to whether the BY variable is just equal to 0. At compile time the matter is quite different.

<u>Example 2:</u>

```
MOVE ZERO TO I.
PERFORM VARYING K BY I UNTIL ...
  ...
END-PERFORM.
```

This shows clearly, without requiring any program run, that the BY variable is equal to 0. This case could therefore be recognized at compile time. Is it, however, reasonable to require that for the warning mechanism? The compiler must only keep the values of (BY) variables from one statement to the next statement as far as it knows those values from the literal assignments and then it can perform the required tests and report the case.

<u>Example 3:</u>

```
PERFORM VARYING K BY I UNTIL ...
  ...
MOVE ZERO TO I
END-PERFORM.
```

Apparently, the BY variable assumed the value 0 at the end of every iteration. This also can be recognized at compile time, it is however, not sufficient for the compiler to keep the values of the (BY) variables from one statement to the (statically) next statement. Rather the compiler must identify the dynamic sequence of statements, i.e. the control flow of the whole program and, in addition, the data flow ending in the BY variables. This is quite costly (some man years) and only a few compilers do that.

<u>Example 4:</u>

```
PERFORM VARYING K BY I UNTIL ...
  ...
MOVE J TO I
  ...
MOVE ZERO TO J
END-PERFORM.
```

<u>Example 5:</u>

```
MOVE A TO B
PERFORM VARYING K BY I UNTIL ...
  ...
IF A = B THEN
MOVE ZERO TO I
END-IF
END-PERFORM.
```

By data flow we must indeed understand the value flow across program branches. In example 5 this means, the value flow analysis helps to identify that the IF condition is fulfilled and that I indeed assumes the value 0. And the control flow analysis is by the way also required for example 2 since the BY variable I will, of course, not assume the value 0 if the described program fragment is dead code, i. e. it can never be reached and performed dynamically.

All this can be recognized at compile time but it is rather doubtful whether a standard or a validation should require that from a compiler.

The list of difficulties continues however.

First, there are the difficulties of value flow analysis. The assignment of the value 0 can be done as directly as shown in the previous examples, but it can also be done in a somewhat hidden way:

- the BY variable can be assigned its value by VALUE IS;
- the BY variable can be assigned its value by INITIALIZE;
- the BY variable can be redefined, and the assignment can be made to the redefined variable.

Problems of different kinds arise from the following examples:

<u>Example 6:</u>

```
COMPUTE I = (J * K) - (K * J)
PERFORM VARYING K BY I UNTIL ...
 ...
END-PERFORM
```

In this case, too, it is recognized at compile time that the BY variable is set to 0. A value analysis for J and K, however, does not help anything for that since the compiler must be able to calculate symbolically with the values.

<u>Example 7:</u>

```
COMPUTE I = J * J - (J + 1) * (J - 1) - 1
PERFORM VARYING K BY I UNTIL ...
 ...
END-PERFORM
```

To recognize here that I assumes the value 0, the compiler must be able to manipulate formulas. Certainly, this is no longer reasonable for a COBOL compiler.

Finally, it is to be noted that even the strongest analysis power of the compiler cannot exclude that there are programs where the BY variable may assume the value 0 though this can never be said definitely at compile time.

<u>Example 8:</u>

```
PERFORM VARYING K BY I UNTIL ...
 ...
IF ... THEN
MOVE ZERO TO I
END-IF
END-PERFORM
```

If the IF condition is never fulfilled in a program run, the BY variable will not assume the value 0. And even if the IF condition is fulfilled, the UNTIL condition can prevent the BY variable from being used with the value 0 for augmenting K. To put it more general: whether the BY variable is used with the value 0, may depend on one or more conditions in such a way:

- that it is definitely true that the BY variable is always used with the value 0,

  or

- that it is definitely true that it is never used with the value 0,

  or

- that it is definitely true that it can be used sometimes with the value 0 and sometimes with a value unequal to 0,

  or

- that, though something being true, it is undecidable.

The first two cases are applicable, for example, if the conditions are logically true or logically false. The third case applies if the computer knows that conditions depend on run time data. And the fourth case says that the conditions are logically undecidable which may also occur.

The last two cases are governed by the rule that it is not necessary to report the usage of the nonstandard extension if it cannot be recognized at compile time. The first two cases, however, give another dimension to the problem. If A is a condition which has just been set TRUE/FALSE, it is easy. If the condition is A OR NOT A or A AND NOT A, it is still relatively easy. The condition may however by a quite complicated logical formula that is decidable. To determine whether it is logically true or false, we need a theorem prover. Such theorem provers, however, have technical restrictions. They may be unable to decide certain conditions though these conditions are decidable, but they may be able to decide certain conditions even it requires five CPU hours.

That means that the attribute "can be recognized at compile time" is practically not applicable without modification:

- it may require an impractical amount of time (which cannot be calculated beforehand) to identify the usage of a nonstandard extension at compile time;

- the class of cases of such a usage which can be recognized at compile time is dependent on the power of theorem provers, i. e. on the state of the art of computer science in the field of "artificial intelligence".

Before making these conclusions, the following should be said. So far the argumentation is dependent on the example of a single rule of the Standard (see page VI-111, paragraph 6.21.4, general rule 1, namely that the BY variable must not be equal to 0. Such a rule is however no exception, but the argumentation is quite generally applicable to all cases where the Standard defines restrictions in data flow or control flow (as these two flows are totally dependent of each other) without defining what will happen in case of a violation of the restriction. By some browsing I have found the following examples: page IV-21, paragraph 4.3.8.2.4, general rules 1 and 3, page IV-23, paragraph 4.3.8.3.4, general rules 4a and 4b; page VI-27, paragraph 5.8.4, general rule 2b; page VI-53, paragraph 6.2.3, rule 5a; page VI-111, paragraph 6.21.4, general rule 2; page VI-120, paragraph 6.21.4, general rule 14; page VI-127, paragraph 6.23.3, syntax rule 4; page VI-128, paragraph 6.23.4, general rule 2; page VII-27, paragraph 3.7.4, general rules 2, 3, and 4 - all references according to DIS.

I think these examples will suffice. It is obvious that the formula "can be recognized at compile time" in its pure form will lead to - probably not intended - requirements to COBOL implementations and even to the limits of computer science. The long series of examples also shows that the addition "can be recognized at compile time" leaves a wide space for variations and can therefore not form a basis for contracts between manufacturers and validators. For this reason, it has to be defined precisely what is included in or excluded by "reasonably". I think that the above examples also show that the area beyond syntax does not only leave by a wide space for variation, but also has floating boundaries. This makes it particularly difficult to give a clear definition saying in a few sentences what has to be recognized and what not.

Therefore, the conclusion is that the warning mechanism must indeed be restricted to the "syntax". Insofar the line of the FIPS flagging requirements must be considered quite "reasonable". If this is also the line of WG4, it should be put into practice for the validation of COBOL 85.

This would require:

1. an exact formulation saying that the warning mechanism of nonstandard extensions must report precisely those usages whose syntax deviates from chapter V (DIS) or, if that is the intention, from chapter V plus all syntax rules of chapters VI to XVI;

2. a binding declaration of the validators and/or certifiers that the manufacturer can take this formula a guideline, i. e. that a non-error validation and certificate does not require more. This is important since otherwise nothing of binding character is said in the Standard except the unfulfillable paragraph 1.5.2.5.2;

3. an official interpretation of this section in the Standard by ISO or ANSI for supporting item 2.

For the declaration according to item 2, we need, and this will probably apply to all other manufacturers concerned, a longer lead time before putting into force a validation program testing this item. Six months' lead time as upon availability of the test suite would certainly be not enough since development can only be started when we know what to develop. A syntax flagger satisfying today's quality standards easily requires 1 to 1 1/2 years for development if there are many extensions and if it has to be newly constructed. Possibly, the validation of the warning mechanism has to be deferred. In this case a declaration according to item 2 should also define the date as from which the warning mechanism will be tested.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that a warning mechanism in a conforming implementation of ANSI COBOL X3.23-1985 that indicates a program contains nonstandard extensions is only required to flag extensions that are syntactically distinguishable.

PROPOSED CORRECTION TO X3.23-1985:

X3J4 has proposed a correction to ANSI COBOL X3.23-1985 as a result of the processing of the question in document A-11. Page 61 of this document contains the proposed correction to page I-8 in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-12

SUBJECT: Evaluation of Subscripts and Reference Modification in the INITIALIZE Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page IV-18, paragraph 4.3.8, uniqueness of reference, last paragraph
2. Page VI-92, paragraph 6.17.4, general rule 2, INITIALIZE statement
3. Page VI-103, paragraph 6.19.4, general rule 2, MOVE statement

DATE: September 11, 1986

QUESTION:

Reference 1 states that, "Unless otherwise specified by the rules for a statement, any subscripting and reference modification are evaluated only once as the first operation of the execution of the statement." Reference 2 states that for an INITIALIZE statement "all operations are performed as if a series of MOVE statements had been written." Reference 3 states that, for the MOVE statement, any length evaluation or subscripting associated with the receiving areas "is evaluated immediately before data is moved to the respective data item.

It is unclear whether the "as if" in reference 2 constitutes an "otherwise specified" for the time of evaluation of subscripting and reference modification for the INITIALIZE statement, particularly since reference 3 is a clear case of "otherwise specified" for the MOVE statement.

For an INITIALIZE statement, is any subscripting and reference modification evaluated only once as the first operation of the statement?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the subscripting and reference modification is not evaluated only once as the first operation of the execution of the INITIALIZE statement.

The results of the execution of an INITIALIZE statement are equivalent to the results of the implied series of MOVE statements defined by general rules 2 and 6 of the INITIALIZE statement.

X3J4 DOCUMENT A-13

SUBJECT: PICTURE Symbol 'P' and the MOVE Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-31, paragraph 5.9.4, general rule 8, symbol P, paragraph b.
2. Page VI-105, paragraph 6.19.4, general rule 5, MOVE statement
3. Page VI-31, paragraph 5.9.4, general rule 8, symbol P
4. Page VI-104, paragraph 6.19.4, general rule 4, MOVE statement

DATE: September 28, 1986

QUESTION:

Reference 1 states that the digit position specified by the symbol P in the PICTURE character-string of a data item is counted in the size and is presumed to contain a zero for "A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol 'P'." Note that reference 1 is introduced with the word "any" and that MOVE is not qualified as an elementary MOVE.

Reference 2 states that "Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another."

It is unclear whether reference 2 overrides the explicit statement made in reference 1.

When a data item described with the symbol 'P' in its PICTURE character-string is moved to a group data item, are the positions described with the symbol 'P' counted in the size of the data item and presumed to contain a zero?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following: Since the implied value zero in character-string positions described with the symbol 'P' will only be used during a valid elementary move, the result of a move from an item containing 'P' symbols in its PICTURE description to a group item, will be that the receiving group item will only receive the stored numeric characters for the sending item and not the positions in the sending item represented by the 'P' symbols.

PROPOSED CORRECTION TO X3.23-1985:

X3J4 has proposed a correction to ANSI COBOL X3.23-1985 as a result of the processing of the question in document A-13. Page 61 of this document contains the proposed correction to page VI-31 in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-14

SUBJECT: Transfer of Control in Format 1 READ Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VII-46, paragraph 4.4.4, general rule 10b, READ statement
2. Page VIII-28, paragraph 4.5.4, general rule 10b, READ statement
3. Page IX-30, paragraph 4.5.4, general rule 10b, READ statement
4. Page IV-25, paragraph 4.4.2, first paragraph, explicit and implicit transfers of control; and page IV-35, paragraph 6.4.1.3, execution
5. Page IV-27,. paragraph 4.4.4, explicit and implicit scope terminators
6. Page VII-46,paragraph 4.4.4, general rule 11c, READ statement; page VIII-29, paragraph 4.5.4, general rule 11c, READ statement; page IX-31, paragraph 4.5.4, general rule 11c, READ statement

DATE: January 12, 1987

QUESTION:

Reference 1 (Sequential I-O module), reference 2 (Relative I-O module), and reference 3 (Indexed I-O module) state that under certain circumstances, "control is transferred to imperative-statement-1 in the AT END phrase." What is the transfer of control in a format 1 READ statement following the execution of imperative-statement-1?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following:

After the execution of imperative-statement-1 in the AT END phrase of the READ statement, control flows to the next statement following the READ statement unless imperative-statement-1 contains a procedure branching or conditional statement that overrides this sequence. (See references 4, 5, and 6.)

PROPOSED CORRECTION TO X3.23-1985:

X3J4 has proposed a correction to ANSI COBOL X3.23-1985 as a result of the processing of the question in document A-14. Pages 63 and 64 of this document contain the proposed corrections to pages VII-46, VIII-28, and IX-30 in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-15

SUBJECT: SYMBOLIC CHARACTERS Clause

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-13, paragraph 4.5.2, general format
2. Page VI-16, paragraph 4.5.4, general rule 8
3. Page IV-2, paragraph 2.1.4, brackets, braces, and choice indicators

DATE: August 31, 1987

QUESTION #1:

In reference 1, the general format for the SYMBOLIC CHARACTERS clause has an extra pair of braces { } . The opening one follows the word CHARACTERS and the closing one follows the closing bracket of the optional IN ALPHABET phrase. The braces are not followed by an ellipsis, and no choices are listed within the braces. What is the meaning of these braces?

X3J4 RESPONSE TO QUESTION #1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that this is a typographical error and the braces should not be present. (See page 66 of this document for editorial changes to pages V-3 and VI-13 in ANSI COBOL X3.23-1985.)

QUESTION #2:

In reference 2, the IN ALPHABET phrase is described as applying to integer-1. Integer-1 is followed by two occurrences of ellipses indicating repetition. Does the IN ALPHABET phrase apply to all possible instances, as indicated by those ellipses, of integer-1 which occur between the words SYMBOLIC CHARACTERS and the IN ALPHABET phrase?

X3J4 RESPONSE TO QUESTION #2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the IN ALPHABET phrase does apply to all possible instances.

X3J4 DOCUMENT A-17

SUBJECT: Binary Data and READ Statement

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VI-47, paragraph 5.14.4, general rule 3, USAGE clause
2. Page VI-55, paragraph 6.3.1.1.1, comparison of numeric operands

DATE: April 4, 1987

QUESTION:

```
03  A  PICTURE 999  USAGE IS BINARY.
```

With reference to the above description of a numeric data item, item A is part of a record description for an input file. By a READ statement that references this record, item A has got the binary representation of the decimal value 1023. Since the binary usage of item A requires it to be at least 9 bits in size, the value of 1023 is a possible value. And during a READ no conversion takes place. Now if you compare item A as follows:

```
IF A IS GREATER THAN 900
```

What is the result of this comparison? According to paragraph 6.3.1.1.1 on page VI-55, the comparison is made with respect to the algebraic value of A. But what is the algebraic value of A in this case?

Is it 1023? Or is it the value of A modulo 999, which gives it the value of 24? Or is it 999, being the maximum value as defined by the PICTURE character-string? Or is it undefined?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following:

The value acquired via the READ statement for the data item provided in the example lies outside of the range of values for an item with the description given. Therefore the results of the comparison are undefined.

X3J4 DOCUMENT A-18

SUBJECT: SYNCHRONIZED and REDEFINES

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page IV-17, paragraph 4.3.7, item alignment
2. Page VI-45, paragraph 5.13.4, general rule 8, SYNCHRONIZED clause
3. Page VI-45, paragraph 5.13.4, general rules 9 and 10, SYNCHRONIZED clause

DATE: August 5, 1987

QUESTION:

Suppose that the memory unit of a specific computer consists of a fixed number of six character positions.

The following data description is given:

```
01  GROUP.
    03  A  PICTURE XX  VALUE IS "AA".
    03  B  PICTURE X(7)  VALUE IS "BBBBBBB".
    03  C  REDEFINES B  PICTURE X(7)  SYNC LEFT.
    03  D  PICTURE XXX  VALUE IS "DDD".
```

What will the memory look like?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following:

It is impossible to know what the memory would look like without the implementor's definition of data alignment (see reference 1), rules for generation of any implicit FILLER (see reference 2), and rules for synchronization of the record itself (see reference 3).

X3J4 DOCUMENT A-19

SUBJECT: OPEN OUTPUT and Unavailable File

REFERENCES:

American National Standard COBOL X3.23-1985
1. Pages VII-39 and 40, paragraph 4.3.4, general rule 1, OPEN statement
2. Pages VIII-21 and 22, paragraph 4.4.4, general rule 1, OPEN statement
3. Pages IX-23 and 24, paragraph 4.4.4, general rule 1, OPEN statement
4. Pages I-7 and 8, paragraph 1.5.2.4, externally provided functions

DATE: April 24, 1987

QUESTION:

If a COBOL program opens a file for output, and that file does not exist at run time, is it incumbent that the COBOL run time system create the file when creation is not an atomic (i.e., requires external data) function?

If interpretations of ANSI COBOL X3.23-1974 are no longer given, we would appreciate an interpretation following the ANSI COBOL X3.23-1985 Standard. In this area, the standards appear to be in agreement.

We believe that the answer to the above question regarding OPEN is "no", particularly when the file is ISAM (i. e., indexed). We have implemented our low-intermediate ANSI COBOL 1974 compiler accordingly. We have done so because of the potential to make erroneous assumptions if it were possible to create files in this manner. However, we concede that an interpretation of the ANSI COBOL Standard ALLOWS (but does not REQUIRE) ANSI COBOL 1974 implementors to cause a file to be created under these circumstances. Moreover, we know that some implementors have actually done so, in ANSI COBOL 1974.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following:

In the circumstances described, ANSI COBOl X3.23-1985 requires that the file be created. However, an implementation may require specification outside the source program to interface with the operating environment to support functions specified in the source program.

X3J4 DOCUMENT A-20

SUBJECT: CALL Literal-1 with ON EXCEPTION Phrase

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page X-27, paragraph 5.2.2, general format
2. Page X-28, paragraph 5.2.4, general rule 3

DATE: July 31, 1987

QUESTION:

When literal-1 is specified in the CALL statement it is often possible to determine whether the program to call is available before the program begins to execute. This allows an implementation for the CALL which is much more efficient. The programmer, also, would prefer to find errors as early in the process of developing a program as possible. Waiting until run time to find out that the subprogram does not exist is both inefficient and awkward for the programmer. If the ON EXCEPTION phrase is specified together with literal-1 in the CALL statement, must the implementor allow the program to begin to execute?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the implementor must allow the program to execute, so that the ON EXCEPTION path can be taken at run time, if required.

X3J4 DOCUMENT A-22

SUBJECT: NOT in Abbreviated Combined Relation Conditions

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page III-20, relational operator definition
2. Page IV-9, paragraph 4.2.2.1.3.2, optional words
3. Page VI-59, paragraph 6.3.2, complex conditions
4. Page VI-60, table 1, combinations of conditions, logical operators, and parentheses
5. Page VI-61, paragraph 6.3.3, abbreviated combined relation condition

DATE: August 3, 1987

QUESTIONS:

Question 1:

Does the presence of "IS" affect the interpretation of "NOT" in abbreviated combined relation conditions? ANSI COBOL X3.23-1985 is very explicit on this point:

1. Page VI-61, paragraph 6.3.3, shows the format for an abbreviated combined relation condition to consist of a relation condition, followed by one or more sequences of:

   - the word AND or OR,
   - optionally the word NOT,
   - optionally a relational operator, and
   - an object.

2. Relational operator is defined on page III-20, and may also contain the word NOT in certain combinations.

3. Page VI-61, paragraph 6.3.3, further includes the following rule: "The interpretation ... of NOT in an abbreviated combined relation condition is as follows:

   a. If the word immediately following NOT is GREATER, >, LESS, <, EQUAL, =, then the NOT participates as part of the relational operator; otherwise,

   b. The NOT is interpreted as a logical operator ... ".

One of the effects of these is apparently to distinguish between the two flavors of NOT in the following examples:

```
(a)  IF A IS EQUAL B OR IS NOT LESS THAN C OR D

(b)  IF A IS EQUAL B OR NOT IS LESS THAN C OR D
```

which are expanded to:

```
(a)  IF (A IS EQUAL B) OR (A IS NOT LESS THAN C) OR (A IS NOT LESS THAN D)

(b)  IF (A IS EQUAL B) OR (NOT (A IS LESS THAN C) OR (A IS LESS THAN D))
```

Note the semantic difference: in (b) the NOT only negates the comparison with C and does not affect the propagated relational operator. If you omit the word IS, the semantic difference disappears, and both examples are interpreted as (a). This appears to be in direct conflict with page IV-9, paragraph 4.2.2.1.3.2, which states that optional words may be specified with no effect on semantics.

A better wording for the interpretation of NOT might have been:

"a. If the word NOT and immediately adjacent words form one of the legal relational operators as defined on page III-20, then the NOT participates as part of the relational operator; otherwise,

b. The NOT is interpreted as a logical operator ..."

<u>Question 2:</u>

Is the sequence NOT NOT permitted in abbreviated combined relation conditions? According to the definitions quoted above, the following example seems to be legal:

```
(c)  IF A IS EQUAL B OR NOT IS NOT LESS THAN C OR D
```

and is expanded to:

```
IF (A IS EQUAL B) OR (NOT (A IS NOT LESS THAN C))
   OR (A IS NOT LESS THAN D)
```

The first NOT is interpreted as a logical operator, and the second NOT is part of the relational operator. Now if you omit the optional word IS, you get:

```
(d)  IF A EQUAL B OR NOT NOT LESS THAN C OR D
```

which should be just as legal as (c) and have the same semantics.

This interpretation seems to contradict Table 1 on page VI-60 and specifically the paragraph following the table. However, Table 1 refers to logical operators only, i.e. a pair of logical operators NOT is not allowed, but a logical operator NOT followed by a NOT as part of a relational operator in an abbreviated combined relation condition should be valid. Is this interpretation correct?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that:

Question 1: According to reference 5, the NOT in (b) does not participate as part of the relational operator since it does not precede any of GREATER, >, LESS, <, EQUAL, or =. It cannot be considered part of the relational operator because NOT IS LESS THAN is not a relational operator. This situation does violate the intention of ANSI COBOL X3.23-1985 with regard to optional words.

Question 2: Table 1 and the following text prohibiting use of two consecutive NOT's are part of paragraph 6.3.2, Complex Conditions, and apply only to use of NOT in complex conditions, not to its use in abbreviated combined relation conditions. Example (d) is valid and its expansion complies with the rules of Table 1 as required by paragraph 6.3.3.

X3J4 DOCUMENT A-23

SUBJECT: Overlapping Parameters in CALL Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-69, paragraph 6.4.5, overlapping operands
2. Page X-29, paragraph 5.2.4, general rule 12, CALL statement
3. Pages X-29 and 30, paragraph 5.2.4, general rule 13, CALL statement
4. Page X-30, paragraph 5.2.4, general rule 14, CALL statement

DATE: March 23, 1987

QUESTION:

Does the rule for overlapping operands (see reference 1) apply to the parameters of a CALL statement?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that:

1. Overlapping operands passed BY REFERENCE in a CALL statement may cause an overlapping operand condition in the called program.

2. For operands passed BY CONTENT, the overlapping operand condition cannot occur in the called program since copies of the operands are passed.

3. The rules of overlapping operands do not apply directly to the CALL statement.

X3J4 DOCUMENT A-25

SUBJECT: START Statement with Generic Alternate Key

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page IX-37, paragraph 4.7.4, general rule 4, START statement
2. Page IX-2, paragraph 1.3.2, access modes, first paragraph
3. Page IX-37, paragraph 4.7.4, general rule 10c, START statement
4. Page IX-9, paragraph 2.3.4, general rule 3, file control entry
5. Page IX-29, paragraph 4.5.4, general rule 4c, READ statement

DATE: March 20, 1987

QUESTION:

Given an indexed file with the following records:

Record Number	Primary Key	Alternate Key
1	222	a1
2	333	b2
3	444	b1
4	555	c1

and given the following code sequence:

```
SELECT F1  ASSIGN TO "F1"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS KEY1
    ALTERNATE RECORD KEY IS KEY2.
 ...
FD  F1.
01  REC1.
    05  KEY1  PICTURE XXX.
    05  KEY2.
        10  PARTKEY2  PICTURE X.
        10  FILLER  PICTURE X.
 ...
MOVE "b" TO PARTKEY2.
START F1  KEY IS EQUAL TO PARTKEY2.
```

Should the file position indicator be set to record 2 or record 3?

We feel that the file position indicator should be set to record 3 based on the collated order of the key of reference (alternate key).

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the file position indicator in the given example is set to the value of the alternate key with the lowest value starting with the letter "b" (see references 1, 2, and 3) according to the collating sequence associated with the native character set (see reference 4). Assuming that the native collating sequence orders the digit "1" before the digit "2", the file position indicator in the given example would be set to "b1"; a READ NEXT statement executed following the START statement would then make record 3 available (see reference 5).

X3J4 DOCUMENT A-26

SUBJECT: Group Item with OCCURS and VALUE Clauses

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VI-49, paragraph 5.15.6, general rule 3
2. Page VI-50, paragraph 5.15.6, general rule 6

DATE: August 31, 1987

QUESTION:

Consider the following case:

```
01  A.
    02  B  OCCURS 2 TIMES  VALUE IS "ABC".
           03  C  PICTURE XXX.
```

Since B is a group item, reference 1 applies, and the initial value of B should be "ABC   ", resulting in an initial value of B(1) of "ABC" and for B(2) of "   ".

Since B is a data description entry that contains an OCCURS clause, reference 2 applies, and the initial value of B(1) and B(2) should both be "ABC".

Which of these possibilities is correct?

The first choice does not seem natural, since any reference to B normally requires a subscript to determine uniqueness of reference. However, both general rules seem to apply and neither has an apparent precedence.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that B(1) and B(2) would both contain "ABC". Reference 1 is not pertinent in this case.

X3J4 DOCUMENT A-27

SUBJECT: Size of an OCCURS DEPENDING ON Item During CALL BY CONTENT

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-28, paragraph 5.8.4, general rule 3b, OCCURS clause
2. Page X-25, paragraph 5.1, rule 1, Procedure Division header
3. Page X-30, paragraph 5.2.4, general rule 14, CALL statement
4. Page IV-17, paragraph 4.3.6, rule 3, standard alignment

CODASYL COBOL Journal of Development 1984

5. Page III-8.1-2, Paragraph 8.1.4.1, Procedure Division Header

DATE: August 5, 1987

QUESTION:

Given the following example:

```
DATA DIVISION.
01  FIELDA.
        02  FIELDB  PICTURE 99  VALUE IS 5.
        02  FIELDC.
                03  FIELDD  PICTURE X  OCCURS 1 TO 10 TIMES DEPENDING
                                  ON FIELDB.
 ...
PROCEDURE DIVISION.
BEGIN.
    MOVE "1234567890" TO FIELDC.
    CALL "PROG"  USING BY CONTENT FIELDA.
```

Is the argument FIELDA considered as a sending item or receiving item? From the calling program the argument is a sending item, but from the called program the argument is a receiving item. What portion of FIELDC is passed to the called program? If it is a sending item, then based on reference 1 the size of FIELDC would be 5. If it is a receiving item, then based on the same reference the size of FIELDC would be 10.

Reference 2 in ANSI COBOL X3.23-1985 indicates that a move is done from the argument to a system defined storage item, whereas reference 5 in the CODASYL COBOL Journal of Development 1984 specifies that a move is done according to rules for the MOVE/SET statements. Is this the type of move intended by ANSI COBOL X3.23-1985?

Reference 3 states that no truncation should occur. Does this extend to the size of FIELDA?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following:

Seven characters of the sending item (FIELDA) are moved to the first seven characters of the twelve character system defined receiving item; the contents of the remaining five characters are undefined (see references 1 and 2).

X3J4 DOCUMENT A-29

SUBJECT: RECORD IS VARYING Clause

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page XI-7, paragraph 3.2.3, syntax rule 4, sort-merge file description entry
2. Page X-18, paragraph 4.2.3, syntax rules 1 and 2, file description entry for Inter-Program Communication module

DATE: May, 26, 1987

QUESTION:

The RECORD IS VARYING clause is leveled inconsistently:

1. In the Sequential I-O, Relative I-O, and Indexed I-O modules, the RECORD IS VARYING clause in the file description entry is a level 2 element.

2. In the Sort-Merge module, the RECORD IS VARYING clause in the file description entry and the sort-merge description entry is a level 1 element.

3. In the Inter-Program Communication module, the RECORD IS VARYING clause in the file description entry is a level 1 element. It appears the text was copied from the Sequential I-O module, but the boxes were forgotten.

If a compiler conforms to the intermediate COBOL subset, it is inconsistent to implement the RECORD IS VARYING clause for the SORT and Inter-Program Communication modules when it is not needed in the Sequential I-O, Relative I-O, and Indexed I-O modules. Shouldn't this clause be level 1 in all modules?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the RECORD IS VARYING clause is leveled consistently. According to reference 1, the availability of the VARYING phrase in the RECORD clause of the sort-merge description entry is dependent on the level of Sequential I-O module supported by the implementation. According to reference 2, in the Inter-Program Communication module the availability of specific clauses in the file description entry is dependent on the level of the relevant I-O module supported by the implementation.

X3J4 DOCUMENT A-30

SUBJECT: NOT INVALID KEY Phrase and REWRITE Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page IX-33, paragraph 4.6.3, syntax rule 3, REWRITE statement in Indexed I-O module
2. Page VIII-30, paragraph 4.6.3, syntax rule 4, REWRITE statement in Relative I-O module

DATE: July 31, 1987

QUESTION:

In the Indexed I-O module, the REWRITE rules state that when no USE procedure is specified, both the INVALID KEY phrase and the NOT INVALID KEY phrase must be specified. This is inconsistent with the rules for REWRITE stated in the Relative I-O module and with other statements that have INVALID KEY phrase. Was this intended? This would make all old programs that use the REWRITE statement for an indexed file nonstandard.

The words "and the NOT INVALID KEY phrase" should be deleted.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the syntax rule in reference 1 for the REWRITE statement in the Indexed I-O module contains a typographical error. The words "and the NOT INVALID KEY" should be deleted from the syntax rule.

PROPOSED CORRECTION TO X3.23-1985:

X3J4 has proposed a correction to ANSI COBOL X3.23-1985 as a result of the processing of the question in document A-30. Page 67 of this document contains the proposed correction to page IX-33 in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-31

SUBJECT: NOT INVALID KEY Phrase and WRITE Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VIII-6, paragraph 1.3.5, seventh paragraph, invalid key condition in Relative I-O module
2. Page VIII-38, paragraph 4.9.4, general rule 9, WRITE statement in Relative I-O module
3. Page VIII-36, paragraph 4.8.4, general rule 6, USE statement in Relative I-O module
4. Page IX-6, paragraph 1.3.5, seventh paragraph, invalid key condition in Indexed I-O module
5. Page IX-42, paragraph 4.9.4, general rule 9, WRITE statement in Indexed I-O module
6. Page IX-40, paragraph 4.8.4, general rule 6, USE statement in Indexed I-O module
7. Page VIII-31, paragraph 4.6.4, general rule 10, REWRITE statement in Relative I-O module
8. Page IX-34, paragraph 4.6.4, general rule 10, REWRITE statement in Indexed I-O module

DATE: July 31, 1987

QUESTION:

The rules which specify flow of control when both a NOT INVALID KEY clause and a USE procedure are specified are not consistent from statement to statement? Should NOT INVALID KEY act differently for the WRITE statement than it does for any other I-O statement?

Relative I-O module:

Page VIII-6, paragraph 1.3.5 for the invalid key condition, states that when an exception that is not an invalid key condition exists, the USE procedure, if any, is executed and any NOT INVALID KEY phrase is ignored. The NOT INVALID KEY phrase is executed only when no exception exists.

The following statements are consistent with these rules: DELETE, READ, and USE. However, the rules for the WRITE statement conflict with the above rules. Page VIII-39, general rule 9b for the WRITE statement states that the NOT INVALID KEY phrase is executed when execution of the WRITE statement is unsuccessful for a reason other than an invalid key condition.

Indexed I-O module:

Page IX-6, paragraph 1.3.5 for the invalid key condition, states that when an exception that is not an invalid key condition exists, the USE procedure, if any, is executed and any NOT INVALID KEY phrase is ignored. The NOT INVALID KEY phrase is executed only when no exception exists.

The following statements are consistent with these rules: DELETE, READ, and USE. However, the rules for the WRITE statement conflict with the above rules. Page IX-42, general rule 9b for the WRITE statement states that the NOT INVALID KEY phrase is executed when execution of the WRITE statement is unsuccessful for a reason other than an invalid key condition.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that transfer of control for the WRITE statement should be the same as specified for the REWRITE statement (see references 7 and 8) instead of as specified (see references 2 and 5).

PROPOSED CORRECTION TO X3.23-1985:

X3J4 has proposed a correction to ANSI COBOL X3.23-1985 as a result of the processing of the question in document A-31. Pages 63 and 64 of this document contain the proposed corrections to pages VII-53, VIII-38, and IX-42 in ANSI COBOL X3.23-1985.

X3J4 DOCUMENT A-32

SUBJECT: INVALID KEY Phrase and REWRITE Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VIII-30, paragraph 4.6.3, syntax rule 3, REWRITE statement in Relative I-O module
2. Page IX-33, paragraph 4.6.3, syntax rule 3, REWRITE statement in Indexed I-O module

DATE: July 31, 1987

QUESTION:

In the Relative I-O module, the REWRITE statement disallows the INVALID KEY clause with sequential access. However, the REWRITE statement in the Indexed I-O module does not.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the COBOL Standard is correct as stated. The invalid key condition cannot occur for the REWRITE statement on relative files for sequential access and, therefore, the INVALID KEY phrase is not allowed.

X3J4 DOCUMENT A-33

SUBJECT: SEARCH Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page II-13, paragraph 4.1, example 4, table definition
2. Page II-14, paragraph 4.3, references to table items
3. Page VI-124, paragraph 6.22.4, general rule 2b, SEARCH statement

DATE: July 28, 1987

QUESTION:

The rules of the SEARCH statement do hardly refer to a multi-dimensional table. In particular, it is not clear how the index(es) of higher hierarchy will be used.

Suppose the following definitions:

```
01  FINDER  PICTURE XXX.
01  TABLE.
    03  ELEMENT-1  OCCURS 10 TIMES;
           INDEXED BY INDEX-A1, INDEX-A2
        05  ELEMENT-2; OCCURS 20 TIMES;
                INDEXED BY INDEX-B;
                PICTURE XXX.
```

Question 1:

```
PERFORM WITH TEST AFTER VARYING INDEX-A1 FROM 1 BY 1
    UNTIL INDEX-A1 = 10
SET INDEX-B TO 1;
SEARCH ELEMENT-2
    WHEN ELEMENT-2 (INDEX-A1, INDEX-B) = FINDER CONTINUE;
END-SEARCH;
END-PERFORM;
```

Will INDEX-A1 be used for the search operation?

Question 2:

As in question 1, but now INDEX-A2 is referred to instead of INDEX-A1. Which index will be used: INDEX-A1 or INDEX-A2?

Question 3:

```
PERFORM WITH TEST AFTER
    VARYING INDEX-B FROM 1 BY 1
    UNTIL INDEX-B = 20
SET INDEX-A1 TO 1;
SEARCH ELEMENT-1
    WHEN ELEMENT-2 (INDEX-A1, INDEX-B) = FINDER CONTINUE;
END-SEARCH;
END-PERFORM;
```

It seems illogical to me to change the minor index in the outer loop. But I have the feeling that this is a "safe" method (compared to methods in questions 1 and 2) with regard to the COBOL Standard. Is that true?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that:

Question 1: INDEX-A1 is "used" in the search operation only to complete the reference to ELEMENT-2. The value of INDEX-A1 is not affected by the search operation.

Question 2: If INDEX-A2 is referred to, it will be used in the manner of INDEX-A1 in question 1.

Question 3: INDEX-B is "used" in the search operation only to complete the reference to ELEMENT-2. The value of INDEX-A2 is not affected by the search operation.

X3J4 DOCUMENT A-34

SUBJECT: OCCURS Clause

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VI-28, paragraph 5.8.4, general rule 3, OCCURS clause
2. Page VI-49, paragraph 5.15.5, rule 2, condition-name rules

DATE: July 28, 1987

QUESTION:

Does general rule 3 of the OCCURS clause also apply to a condition-name that is associated with a group item that contains a table with a variable number of occurrences, as in the following example?

```
01  N PICTURE 99.
01  GROUP.
88  NOT-VALID  VALUE IS SPACES.
    03  ELEM  OCCURS 1 TO 10 TIMES  DEPENDING ON N; PICTURE X.
  ...
MOVE 5 TO N.
IF NOT-VALID   DISPLAY GROUP.
```

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that general rule 3 of the OCCURS clause applies to a condition-name associated with a group item that contains a table with a variable number of occurrences.

X3J4 DOCUMENT A-35

SUBJECT: Size of an OCCURS DEPENDING ON Item with Value of DEPENDING ON Item Out of Bounds

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VI-27, paragraph 5.8.4, general rule 2b, OCCURS clause

DATE: August 5, 1987

QUESTION:

Given the following code:

```
01  NUMBER  PICTURE 9.
01  TABLE.
    03  ELEMENT  OCCURS 3 TO 5 TIMES  DEPENDING ON NUMBER.
        05  ITEM  PICTURE XX.

MOVE 2 TO NUMBER.
MOVE SPACES TO ELEMENT (NUMBER).
MOVE SPACES TO ITEM (NUMBER).
```

According to general rule 2b, third paragraph, of the OCCURS clause, the last two MOVE statements could cause problems since the value of NUMBER does not fall within the specified range of 3 through 5. Is this interpretation correct? If so, what is the reason behind this rule?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example is not standard conforming because the value of NUMBER when ELEMENT and ITEM are referenced is not between the minimum and maximum values.

X3J4 DOCUMENT A-36

SUBJECT: UNSTRING Statement

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page VI-43, paragraph 5.12.4, general rule 6a, SIGN clause
2. Page VI-137, paragraph 6.27.4, general rule 9, UNSTRING statements
3. Pages VI-31 and 32, paragraph 5.9.4, general rule 9, PICTURE clause

DATE: July 24, 1987

QUESTION:

The following has been coded:

```
01  A  PICTURE X(3)  VALUE IS "1**".
01  B  PICTURE 9.
01  C  PICTURE S9  SIGN IS LEADING SEPARATE CHARACTER.

UNSTRING A DELIMITED BY "*" INTO B, C.
```

Will data item C contain 00 or +0?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that applying general rule 9 of the UNSTRING statement to a data item defined as PICTURE S9 would yield +0.

X3J4 DOCUMENT A-37

SUBJECT: Global USE Statement

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page X-6, paragraph 1.3.8.2, conventions for condition-names, data-names, file-names, record-names, and report-names
2. Page X-34, paragraph 5.5.4, USE statement
3. Page I-7, paragraph 1.5.2.3, obsolete language elements

DATE: September 10, 1987

QUESTION:

We request a verification of some implementations in ANSI COBOL X3.23-1985. Please refer to references 1 and 2.

These paragraphs indicate that it is possible for a program to have a global file definition and associated global USE ... ERROR or USE ... DEBUGGING procedure(s) that are referenced by (directly and indirectly) contained programs.

However, that also implies that it is possible:

- for several programs to refer to the same global file and for each of these programs to have different USE ERROR or USE DEBUGGING procedure(s) for that file.

- for a program A to define a global file and to contain a program B; for the program B to define a global USE procedure for that global file and to contain a program C; and for the program C to refer to both the global file and the global USE procedure.

Are those implications true?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that:

1. The implications for the global USE statement suggested in the question are true for the USE AFTER EXCEPTION/ERROR statement.

2. The GLOBAL phrase on a USE FOR DEBUGGING statement is not part of ANSI COBOL X3.23-1985, and therefore no interactions are defined.

X3J4 DOCUMENT A-38

SUBJECT: SEARCH on Item Subordinate to an Item with OCCURS Clause

REFERENCES:

American National Standard COBOL X3.23-1985

1. Pages VI-123 and 124, paragraph 6.22.4, general rules 2 and 4, SEARCH statement
2. Pages IV-21 and 22, paragraph 4.3.8.2.4, general rules 1 and 3, subscripting

DATE: August 2, 1987

QUESTION:

We have a question about ANSI COBOL X3.23-1985 concerning the description of the verb SEARCH. ANSI COBOL 74 SEARCH rules require an index-name be associated with each level of a multi-dimensional table one of whose subordinate elements is to be searched. In ANSI COBOL X3.23-1974, general rule 8 of the OCCURS clause states:

"If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two or three dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause."

ANSI COBOL 85 does not indicate any such requirement. What is supposed to happen when an item that is subordinate to an item with an OCCURS clause is searched?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the Standard adequately describes the operation of a SEARCH statement whose identifier is an item subordinate to an item containing an OCCURS clause.

The ANSI COBOL 74 SEARCH statement rule quoted in the question above focuses on the specific requirement that was changed from the 1974 to the 1985 Standard for SEARCH statements. Under the 1974 Standard, if the search operation used an index-name, then all subscript references had to be index-names. This requirement has been removed from the 1985 Standard allowing a combination of index-names and data-names to be used by a single search operation.

As regards the specific question about how an item subordinate to an item with an OCCURS clause is searched, this is described in the current Standard (reference 1) and no additional clarification is required. According to general rules 1 and 3 for subscripting, index-names or data-names not directly manipulated by the search operation must be set prior to execution of the SEARCH statement (see reference 1 and 2).

X3J4 DOCUMENT A-39

SUBJECT: INITIALIZE Statement

REFERENCES:

American National Standard COBOL X3.23-1985
1. Pages VI-92 and 93, paragraph 6.17.4, general rule 2, INITIALIZE
2. Page VI-92, paragraph 6.17.2, function of INITIALIZE

DATE: June 2, 1987

QUESTION:

We request a verification of an implication in ANSI COBOL X3.23-1985. Please refer to page VI-93, Paragraph 6.17.4, general rule 2a and 2b:

" ... item is initialized only if it belongs to the category specified in the REPLACING phrase."

The above passage seems to imply that if an item does not belong in any of the REPLACING categories, it would not be initialized at all. For example:

```
01  A.
    02  A1  PICTURE 9.
    02  A2  PICTURE A.
 ...
INITIALIZE A REPLACING NUMERIC DATA BY 10.
```

What, if anything, does A2 get initialized to? Does the user have to specify:

```
INITIALIZE A REPLACING NUMERIC DATA BY 10, ALPHABETIC DATA BY SPACE.
```

in order to initialize A2 also?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that, when the REPLACING phrase is specified, data items which do not belong to any category specified in the REPLACING phrase are not affected by the execution of the INITIALIZE statement (see references 1 and 2). The answer to your first question is that the value of A2 would be unchanged. The answer to your second question is yes.

X3J4 DOCUMENT A-40

SUBJECT: SAME SORT/SORT-MERGE AREA Clause

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page XI-5, paragraph 2.5.4, general rule 2d, SAME SORT/SORT-MERGE AREA clause
2. Page XI-17, paragraph 4.4.3, syntax rule 9, SORT statement

DATE: July 31, 1987

QUESTION:

We request an official interpretation of ANSI COBOL X3.23-1985 on page XI-17, paragraph 4.4.3, syntax rule 9:

"No pair of file-names in the same SORT statement may be specified in the same SAME SORT AREA or SAME SORT-MERGE AREA clause ..."

Do pairs of file-names include pairs that consist of a sort file-name and a GIVING file-name?

The rules for the SAME RECORD/SORT/SORT-MERGE AREA clause discuss the sharing of storage for the following cases:

1. Sort and/or merge files. Page XI-5, paragraph 2.5.4, general rule 2a: "... Thus any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

2. Non-sort/merge files. Page XI-5, paragraph 2.5.4, general rule 2c: "Files other than sort or merge files do not share the same storage area with each other. ..."

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that a program which includes pairs of file-names, consisting of a sort file-name and a GIVING file-name from the same SORT statement, in the same SORT/SORT-MERGE AREA clause is a non-conforming program, because it violates reference 2.

X3J4 DOCUMENT A-43

SUBJECT: NO DATA and WITH DATA Phrases of RECEIVE Statement

REFERENCES:

American National Standard COBOL X3.23-1985
1. Page XIV-23, paragraph 3.5.4, general rule 4, RECEIVE statement
2. Page XIV-24, paragraph 3.5.4, general rule 5b, RECEIVE statement

DATE: September 10, 1987

QUESTION:

Reference 2 says that when no data is available and there is no NO DATA phrase then execution is suspended until data is available. Reference 1 says that when there is a WITH DATA phrase and data is made available then the WITH DATA phrase is executed.

This appears to mean that if there is a WITH DATA phrase and no NO DATA phrase then the WITH DATA phrase is executed unconditionally, as soon as data is made available. Is this correct?

Alternatively, is there an error in ANSI COBOL X3.23-1985? Should reference 2 say that execution is suspended only when no data is available and there is neither a NO DATA phrase nor a WITH DATA phrase?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the following:

ANSI COBOL X3.23-1985 is correct. If there is a WITH DATA phrase and no NO DATA phrase, then the WITH DATA phrase is executed unconditionally as soon as data is made available (see reference 1).

When there is no data available, execution is suspended in the absence of a NO DATA phrase, without regard to whether or not the WITH DATA phrase is specified (see reference 2). When data is subsequently made available, general rule 4 (see reference 1) applies: "When, during the execution of a RECEIVE statement the MCS makes data available ...". Then, if the WITH DATA phrase is specified, control is transferred to imperative-statement-2; otherwise, control is transferred to the end of the RECEIVE statement.

X3J4 DOCUMENT A-44

SUBJECT: MOVE Alphanumeric to Numeric or Numeric Edited

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-104, paragraph 6.19.4, general rules 3 and 4b-3, MOVE statement
2. Page VI-30, paragraph 5.9.4, general rule 3b, PICTURE clause
3. Page III-15, definition of numeric character
4. Page VI-70, paragraph 6.4.7, incompatible data

DATE: August 20, 1987

QUESTION:

Please explain how general rule 4b-3 of the MOVE statement applies to nonnumeric literals. For example, what results are expected in the following MOVE statements:

```
01  NUM-INT  PICTURE 999.
01  NUM-NONINT  PICTURE 999V999.
02  NUM-ED  PICTURE 999.999.

MOVE "123.456" TO NUM-INT, NUM-NONINT, NUM-ED.
MOVE "$123.45" TO NUM-INT, NUM-NONINT, NUM-ED.
MOVE "123ABC" TO NUM-INT, NUM-NONINT, NUM-ED.
MOVE "123    " TO NUM-INT, NUM-NONINT, NUM-ED.
```

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that since the content of the data items referenced by the MOVE statements contain incompatible data, the results of these operations are explicitly undefined.

SECTION 3: DRAFT PROPOSED CORRECTION ADDENDUM TO ANSI COBOL X3.23-1985

3.1 INTRODUCTION

This section of the COBOL Information Bulletin contains a copy of the draft proposed correction addendum to ANSI COBOL X3.23-1985 as of the September 1987 meeting of X3J4. This draft proposed correction addendum is subject to further modification resulting from future meetings of the X3J4 COBOL Technical Committee.

3.2 DISCLAIMER

It should be noted that the content of this draft proposed correction addendum to ANSI COBOL X3.23-1985 is an incomplete document that has not yet received final approval of the X3J4 COBOL Technical Committee. Thus the content of this draft proposed correction addendum is offered as an informational document for those who are interested in the current work of X3J4. Even though the changes listed below suggest alternate wording, and reference actual pages, paragraphs, and sentences, these changes have not been applied to ANSI COBOL X3.23-1985.

3.3 PROPOSED CHANGES TO ANSI COBOL X3.23-1985

The following proposed changes correct errors in ANSI COBOL X3.23-1985. The line under the page number contains the number of the A-document in Section 2 that is associated with the proposed change.

Page No.	Proposed Change to ANSI COBOL X3.23-1985
I-8 (A-11)	Paragraph 1.5.2.5.2 entitled "Extension Language Elements", last paragraph, add the following as the new last sentence: This warning mechanism is only required to flag extensions that are syntactically distinguishable.
VI-31 (A-13)	Paragraph 5.9.4 of the PICTURE clause, general rule 8, explanation of symbol 'P', second paragraph, change subparagraph b to read: b. An elementary MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol 'P'.

Page No.	Proposed Change to ANSI COBOL X3.23-1985
VI-56	Paragraph 6.3.1.1.3 entitled "Comparisons Involving Index-Names and/or Index Data Items", change the first sentence of subparagraph 2 to read: An index-name and a numeric data item or numeric literal.
VI-113	Paragraph 6.21.4 of the PERFORM statement, general rule 10d, change the fourth and fifth sentences to read: If index-name-1 is specified, the value of identifier-3, index-name-2, or literal-1 must correspond to an occurrence number of an element in the table associated with index-name-1 at the beginning of the execution of the PERFORM statement. If index-name-3 is specified, the value of identifier-6, index-name-4, or literal-3 must correspond to an occurrence number of an element in the table associated with index-name-3 at the beginning of the execution of the PERFORM statement.
VII-32	Paragraph 3.8.4 of the RECORD clause, general rule 10, add the following as a new last paragraph: If the number of character positions in the logical record to be written is less than integer-2 or greater than integer-3, the output statement is unsuccessful and, except during execution of a RELEASE statement, the associated I-O status is set to a value indicating the cause of the condition. (See page VII-2, I-O Status.)
VII-37 (A-10)	Paragraph 4.2.4 of the CLOSE statement in the Sequential I-O module, general rule 3, paragraph F, change "Input Files and Input-Output Files" to read: <u>Input Files and Input-Output Files (Reel/Unit Media):</u> The following operations take place: 1) If the current reel/unit is the last or only reel/unit for the file, there is no reel/unit swap, the current volume pointer remains unchanged, and the file position indicator is set to indicate that no next reel/unit exists. 2) If another reel/unit exists for the file, a reel/unit swap occurs, the current volume pointer is updated to point to the next reel/unit existing in the file, the standard beginning reel/unit label procedure is executed, and the file position indicator is set to one less than the number of the first record existing on the new current volume. If no data records exist for the current volume, another reel/unit swap occurs.

Page No. Proposed Change to ANSI COBOL X3.23-1985

VII-38 (A-10) Paragraph 4.2.4 of the CLOSE statement in the Sequential I-O module, general rule 3, paragraph F, change "Output Files (Non-Reel/Unit Media)" to read:

Input Files, Input-Output Files, and Output Files (Non-Reel/Unit Media):

Execution of this statement is considered successful. The file remains in the open mode, the file position indicator is unchanged, and no action takes place except as specified in general rule 4.

VII-46 (A-14) Paragraph 4.4.4 of the READ statement in the Sequential I-O module, general rule 10b, change to read:

b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement-1. Any USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1 is not executed. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules of that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the READ statement and the NOT AT END phrase, if specified, is ignored.

VII-53 (A-31) Paragraph 4.7.4 of the WRITE statement in the Sequential I-O module, general rule 9, change to read:

(9) If, during the successful execution of a WRITE statement with the NOT END-OF-PAGE phrase, the end-of-page condition does not occur, control is transferred to imperative-statement-2 after execution of the input-output operation.

VIII-28 (A-14) Paragraph 4.5.4 of the READ statement in the Relative I-O module, general rule 10b, change to read:

b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement-1. Any USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1 is not executed. Execution than continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules of that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the READ statement and the NOT AT END phrase, if specified, is ignored.

Page No.	Proposed Change to ANSI X3.23-1985
VIII-38 (A-31)	Paragraph 4.9.4 of the WRITE statement in the Relative I-O module, general rule 9, change to read: (9) Transfer of control following the successful or unsuccessful execution of the WRITE operation depends on the presence or absence of the INVALID KEY and NOT INVALILD KEY phrases. (See Invalid Key Condition on page VIII-5.)
IX-30 (A-14)	Paragraph 4.5.4 of the READ statement in the Indexed I-O module, general rule 10b, change to read: b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement-1. Any USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1 is not executed. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules of that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the READ statement and the NOT AT END phrase, if specified, is ignored.
IX-42 (A-31)	Paragraph 4.9.4 of the WRITE statement in the Indexed I-O module, general rule 9, change to read: (9) Transfer of control following the successful or unsuccessful execution of the WRITE operation depends on the presence or absence of the INVALID KEY and NOT INVALID KEY phrases. (See Invalid Key Condition on page IX-6.)
XI-10	Paragraph 4.1.4 of the MERGE statement, general rule 7b, add the following as a separate paragraph: For a relative file, the content of the relative key data item is undefined after the execution of the MERGE statement.

SECTION 4: EDITORIAL CHANGES TO ANSI COBOL X3.23-1985

4.1 INTRODUCTION

This section of the COBOL Information Bulletin contains a copy of the X3J4 document that lists editorial changes to American National Standard COBOL X3.23-1985 made by X3J4 through its September 1987 meeting.

4.2 EDITORIAL CHANGES TO ANSI COBOL X3.23-1985

The following are editorial changes to be applied to American National Standard COBOL X3.23-1985 in order to correct editorial errors in the document. These editorial changes have been forwarded to the American National Standard Institute for inclusion with American National Standard COBOL X3.23-1985. The notation (A-xx) on the line under the page number refers to an A-document within Section 2 of this CIB document that is associated with the editorial change.

Page No.	Editorial Change to ANSI COBOL X3.23-1985
I-12	Under "Reference Format", change "Asterisk (8) comment line" to "Asterisk (*) comment line".
I-41	Under "Character Set", change entry in MODULE column for "Characters used in punctuation =" from "1 STM" to "2 STM".
I-45	Under "OBJECT-COMPUTER paragraph", change entry in MODULE column for "SEGMENT-LIMIT clause" from "1 SEG Z" to "2 SEG Z".
I-51	In the line following "Level-number clause", change "may be 1 or 1 digits" to "may be 1 or 2 digits".
II-23	Paragraph 6.4.1.1, entitled "Names of Programs", second paragraph, first line, change "compiled program" to "compiled programs".
II-25	Paragraph 6.4.2.2, entitled "Values of Parameters", second paragraph, penultimate line, change "may be used by a called program to return to the" to "may be used by a called program to return a result to the".
IV-2	Paragraph 2.1.5, entitled "Ellipses", second paragraph, first line, change "In the general format," to "In the general formats,".

Page No.	Editorial Change to ANSI COBOL X3.23-1985
IV-4	Paragraph 4.1, entitled "Character Set", third paragraph, first line, change "fewer than 51 characters, double" to "fewer than 52 characters (all characters of the COBOL character set except the lowercase letters), double".
V-3 (A-15)	SYMBOLIC CHARACTERS clause, delete outermost set of braces.
V-7	ORGANIZATION clause, delete second occurrence of a right bracket after the word SEQUENTIAL.
V-15	Insert a terminal period following the last bracket in format 1.
V-15	Delete the commas between data-name-1 through data-name-11, inclusively, in format 1.
V-16	Insert a terminal period following the last bracket in format 3.
V-16	Delete the commas between data-name-1 through data-name-6, inclusively, in format 3.
V-27	PERFORM format, AFTER phrase, change the "literal-3" immediately after the reserved word AFTER to "index-name-3".
V-28	Insert "[END-REWRITE]" at the end of the first REWRITE statement.
VI-13 (A-15)	Paragraph 4.5.2, SYMBOLIC CHARACTERS clause, delete outermost set of braces.
VI-20	Paragraph 5.3.2 of the data description entry, OCCURS clause, delete the box around the first occurrence of "[INDEXED BY {index-name-1} ...]".
VI-31	Paragraph 5.9.4 of the PICTURE clause, general rule 8, second line, change "explain" to "explained".
VI-50	Paragraph 5.15.6 of the VALUE clause, general rule 6, second line, change "or in a entry" to "or in an entry".
VI-104	Paragraph 6.19.4 of the MOVE statement, general rule 3, fourth line, change "alphabetic, numeric edited," to "alphabetic, alphanumeric, numeric edited,".
VI-105	Paragraph 6.19.4 of the MOVE statement, general rule 4c, change indentation to align with general rule 4b.
VII-6	Paragraph 2.1 in the Input-Output Section of the Sequential I-O module, delete the box around the general format: [I-O-CONTROL. [input-output-control-entry]]

Page No.	Editorial Change to ANSI COBOL X3.23-1985
IX-3	Paragraph 1.3.4, entitled "I-O Status", second occurrence of a paragraph numbered (1), subparagraphs c and d, change indentation to align with subparagraph b.
IX-4	Paragraph 1.3.4, entitled "I-O Status", second occurrence of a paragraph numbered (3), subparagraph b, first line, box "or rewrite".
IX-6	Paragraph 1.3.5, entitled "The Invalid Key Condition", second occurrence of a paragraph numbered (2), first line, change "If not exception" to "If no exception".
IX-33 (A-30)	Paragraph 4.6.3 of the REWRITE statement in the Indexed I-O module, syntax rule 3, change to read: "The INVALID KEY phrase must be specified in the REWRITE statement for indexed files, and for which an appropriate USE AFTER STANDARD EXCEPTION procedure is not specified."
X-1	Paragraph 1.1, Function for Inter-Program Communication module, fifth line, change "data value available" to "data values available".
X-2	Paragraph 1.3.4, entitled "External Objects and Internal Objects", second paragraph, last sentence, change "representative" to "representation".
X-5	Paragraph 1.3.8, entitled "Scope of Names", second paragraph on page X-5, second line, box the word "either". In the same paragraph, third and fourth lines, box "which contains a Configuration Section or in any program contained within that program".
X-6	Paragraph 1.3.8.1, entitled "Conventions for Program-Names", rule 3, box "except programs it directly or indirectly contains".
X-10	Paragraph 2.4.2, entitled "Programs in the Initial State", box numbered paragraphs 3 and 4.
X-19	Paragraph 4.3.1, Function for data description entry in the Inter-Program Communication module, first paragraph, last line, box "or global names".
X-19	Paragraph 4.3.1, Function for data description entry in the Inter-Program Communication module, second paragraph, second line, box "or external".
X-29	Paragraph 5.2.4 of the CALL statement, general rule 10, change two occurrences of "data-names" to "parameters"; also change two occurrences of "data-name" to "parameter".
XI-8	Paragraph 4.1.3 of the MERGE statement, syntax rule 3, fifth line, change "in the file" to "in the files".

Page No.	Editorial Change to ANSI COBOL X3.23-1985
XII-4	Paragraph 2.4 of the COPY statement, general rule 7, fifth and sixth lines, change two occurrences of "pseudo-text-delimiter" to "pseudo-text delimiter".
XII-4	Paragraph 2.4 of the COPY statement, general rule 9, third paragraph, fourth line, change "When a text word" to "When a text word specified in the BY phrase is introduced, it appears on a debugging line if the first library text word being replaced is specified on a debugging line. Except".
XIII-7	Paragraph 3.2.2 of the file description entry in the Report Writer module, VALUE OF clause, delete fourth period in the ellipsis.
XIV-3	Paragraph 2.2.2 of the communication description entry, delete the commas between data-name-1 through data-name-11, inclusively, in format 1.
XIV-4	Paragraph 2.2.2 of the communication description entry, delete the commas between data-name-1 through data-name-6, inclusively, in format 3.
XIV-19	Paragraph 3.2.4 of the DISABLE statement, general rule 4, third line, change "SOURCE)" to "SOURCE))".
XV-5	Paragraph 3.2.3 of the USE FOR DEBUGGING statement, syntax rule 10, last line, delete "or indexing".
XVII-8	Paragraph 2.11, entitled "CODASYL COBOL JOURNAL OF DEVELOPMENT 1981", item 12, change "EXIT PROGRAM" to "EXIT PERFORM".
XVII-19	Under "Reference Format", change entry in 3RD STD column for "Continuation of COBOL word, numeric literal" from "1 NUC" to "2 NUC".
XVII-64	Substantive change 26, entitled "PERFORM statement", prior to the last paragraph on the page, delete: "Under second Standard COBOL, PARA3 will be executed 8 times as shown above. Under third Standard COBOL, PARA3 will be executed 6 times as shown above."
XVII-70	Substantive change 37, entitled "File position indicator", first paragraph, second line, change "access made" to "access mode".

SECTION 5: CROSS REFERENCE INDEX TO INTERPRETATIONS TO ANSI COBOL X3.23-1985

5.1 INTRODUCTION

This section of the COBOL Information Bulletin contains a cross reference index to the interpretations made by the X3J4 COBOL Technical Committee to ANSI COBOL X3.23-1985. This index is arranged in order by subject matter within American National Standard COBOL X3.23-1985.

5.2 DEFINITION OF AN IMPLEMENTATION OF STANDARD COBOL

5.3 NUCLEUS MODULE

Procedure Division of Nucleus Module