Accredited Standards Committee
X3 INFORMATION PROCESSING SYSTEMS

Document No. CIB-26

Chair, X3J4
CBEMA
1250 Eye Street NW, Suite 200
Washington, D.C. 20005
Tel: (202) 737-8888

COBOL INFORMATION BULLETIN

NUMBER 26

INTERPRETATIONS OF

ANSI X3.23-1985

AND

ANSI X3.23a-1989

September 1993

TABLE OF CONTENTS

SECTION 1: INTRODUCTION

1.1 COBOL INFORMATION BULLETINS

COBOL Information Bulletins (CIBs) are published by the Accredited Standards Committee on Information Processing Systems, X3, on behalf of the Technical Committee X3J4 to provide a method of interacting with COBOL users who are involved with 'American National Standard for Information Systems - Programming Language - COBOL,' ANSI X3.23-1985, and are interested in the work of X3J4. To comment on the work of X3J4 or to receive notification of the availability of subsequent COBOL Information Bulletins and the availability of draft COBOL standards for public review, contact:

Chair, X3J4
CBEMA
1250 Eye Street NW, Suite 200
Washington, D.C. 20005

This COBOL Information Bulletin contains the third group of interpretations of ANSI X3.23-1985 and ANSI X3.23a-1989 resulting from inquiries received and processed by X3J4 since publication of ANSI X3.23-1985 and ANSI X3.23a-1989. The first and second group of interpretations of ANSI X3.23-1985 were published in CIB-24 and CIB-25. A comprehensive index of all interpretations of ANSI X3.23-1985 and ANSI X3.23a-1989 is in Section 3.

Copies of CIB-24, CIB-25 and CIB-26 can be ordered from Global Engineering Documents, Inc., using telephone number (800) 854-7179 from within the USA, (714) 261-1455 from outside the USA, or fax number (714) 261-7892.

Copies of ANSI X3.23-1985 and ANSI X3.23a-1989 can be obtained by corresponding directly with:

American National Standards Institute, Inc.
11 West 42nd Street
New York, New York 10036

or by calling (212) 642-4900.

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 four 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 fourth part of the scope is to carry out the procedures to develop the next full revision of the COBOL Standard.

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

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

2. Support the COBOL language as defined in the COBOL Standard and to determine what a subsequent revision to the current COBOL Standard might contain or, more importantly, what it might not contain.

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

1.3 RELATIONSHIP BETWEEN X3J4 AND PARENT COMMITTEE X3

Technical Committee X3J4, responsible for the development and maintenance of Programming Language COBOL, operates under the Accredited Standards Committee X3 on Information Processing Systems. "Accredited" means that Standards Committee X3 is operating under ANSI procedures. 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 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. Many 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 X3.23-1985 was derived from the CODASYL COBOL Journal of Development and the previous COBOL Standard, ANSI X3.23-1974. The CODASYL COBOL Journal of Development was the product of the CODASYL COBOL Committee (CCC), formerly a separate and distinct body from the X3J4 COBOL Technical Committee.

1.5.1 X3J4 Technical Committee

The X3J4 Technical Committee is concerned with standardizing COBOL. X3J4 takes as its basic input the CODASYL COBOL Journal of Development and any other COBOL specification 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:

Chair, X3J4
CBEMA
1250 Eye Street, NW, Suite 200
Washington, D.C. 20005

1.6 TERMINOLOGY AND REFERENCES

As requests for interpretations are submitted by members of the public as well as members of Technical Committee X3J4, there are some inconsistencies within this document when referring to various references or groups. Within this document, the following meanings should be assumed:

"American National Standard for information systems - programming language - COBOL, ANSI X3.23-1985" may be referred to as:

- American National Standard COBOL, X3.23-1985
- Third Standard COBOL
- ANSI X3.23-1985
- ANSI COBOL X3.23-1985
- The '85 Standard

"American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL, X3.23a-1989" may be referred to as:

- American National Standard COBOL, X3.23a-1989
- The Intrinsic Function Amendment
- The Intrinsic Function Addendum
- ANSI COBOL X3.23a-1989
- ANSI X3.23a-1989

Note: Within this document, most references to ANSI X3.23a-1989 actually refer only to those pages which appear in the separately published amendment to the base document, ANSI X3.23-1985. However, it should be understood that a "virtual COBOL standard" is created by applying the changes from the amendment to the base document.

"COBOL Information Bulletin Number 24" may be referred to as:

- CIB-24
- CIB 24
- COBOL Information Bulletin (CIB) - 24

"COBOL Information Bulletin Number 25" may be referred to as:

- CIB-25
- CIB 25
- COBOL Information Bulletin (CIB) - 25

"Draft COBOL Information Bulletin Number 26" may be referred to as:

- Draft CIB 26
- CIB 26
- CIB-26
- COBOL Information Bulletin (CIB) - 26

Note: Within this document either a question or a response may refer to an interpretation that was included in a working draft of this document itself. It is these drafts which are referred to by such references. X3J4 does not ensure that all such references, especially those in requests for additional interpretations, have remained unchanged by the time that the final version of CIB 26 is approved and published.

"Technical Committee X3J4" may be referred to as:

- Technical Committee X3J4 on COBOL
- X3J4
- X3J4 Technical Committee
- X3J4 COBOL Technical Committee
- ANSI X3J4 (COBOL Committee)

Note: Although some questions may refer to X3J4 as an ANSI committee, it should be understood by the reader that X3J4's parent body, X3, is accredited by ANSI - but that X3J4 is only a technical committee under X3 (working under the rules of X3 for such technical committees) and it is not an ANSI accredited committee itself.

For additional terminology currently being proposed or used by X3J4 for referencing various levels of the COBOL Standard, see pages iii and iv of the "Draft Proposed X3.23b-199x Correction Amendment to American National Standard COBOL X3.23-1985 and COBOL X3.23a-1989" that appears in Section 4 of this document.

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, ANSI X3.23-1985 and ANSI X3.23a-1989. 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 ANSI X3.23-1985 and ANSI X3.23a-1989.

Each of the responses is presented in a form showing the reference in ANSI X3.23-1985 and/or ANSI X3.23a-1989, 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. The gap in numbers between A-99 and A-225 is due to a change in X3J4 document numbering procedures.

If the response to a question results in X3J4 formulating a proposed correction to ANSI X3.23-1985, then X3J4's interpretation document may include a cross reference to a proposed correction as documented in Section 4.

An index of all interpretations made relative to ANSI X3.23-1985 and ANSI X3.23a-1989 is located in Section 3. This index is arranged in order by subject matter within ANSI X3.23-1985 and ANSI X3.23a-1989.

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 ANSI X3.23-1985 and ANSI X3.23a-1989.

These interpretations were prepared by X3J4, a technical committee of X3 that developed ANSI X3.23-1985 and ANSI X3.23a-1989, 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 ANSI X3.23-1985 and ANSI X3.23a-1989. ANSI X3.23-1985 and ANSI X3.23a-1989 were approved through the publication and voting procedures of the American National Standards Institute and are not altered by this bulletin. Any subsequent revision of ANSI X3.23-1985 and ANSI X3.23a-1989 may or may not reflect the content of these interpretations.

X3J4 DOCUMENT A-45

SUBJECT: Debugging line interactions with COPY and REPLACE

REFERENCE:

American National Standard COBOL, X3.23-1985

1. Page XII-4, paragraph 2.4, general rule 7, COPY statement
2. Page XII-4, paragraph 2.4, general rule 9, COPY statement
3. Page XII-5, paragraph 2.4, general rule 11, COPY statement
4. Page XII-7, paragraph 3.4, general rules 4 and 8, REPLACE statement
5. Page XII-8, paragraph 3.4, general rules 10 and 11, REPLACE statement
6. Page XII-3, paragraph 2.4, general rules 2, 3, and 4, COPY statement
7. Page VI-141, paragraph 7.3, definition of Debugging lines

DATE: August 28, 1990

QUESTION:

Question 1:

There are a number of distinct contexts in which debugging lines can appear in a COBOL program:

1) The line on which a COPY statement begins
2) The line on which a REPLACE statement begins
3) Library text
4) Pseudo-text-1
5) Pseudo-text-2
6) Clauses of a COPY or REPLACE statement not included within pseudo-text

The author's interpretation of reference 1-5 is that cases 4 and 6 have no effect on the resultant program.

Question 2:

In the case of COPY processing, a text-word appears in the resultant program on a debugging line if any of the following are true:

1) The COPY statement began on a debugging line
2) Library text was not replaced began on a debugging line
3) The first library text-word that was involved in a match began on a debugging line
4) Library text was replaced by pseudo-text-2 which included debugging lines

A text-word does not appear on a debugging line in the resultant program only if all of the above conditions are false.

Are the above assumptions correct, i.e., is this restatement of the rules equivalent to ANSI X3.23-1985?

Question 3:

In the case of REPLACE processing, a text word appears in the resultant program on a debugging line if:

1) Source text that was not replaced began on a debugging line
2) Source text that was replaced began on a debugging line

This is considerably different from the COPY rules. If the COPY statement begins on a debugging line, all the resultant text due to COPY processing is considered to occur on debugging lines. There is no equivalent statement for the REPLACE statement.

If a REPLACE statement begins on a debugging line, should the resultant text due to REPLACE processing be considered to be on debugging lines?

Question 4:

Resultant text due to replacement by pseudo-text-2 that includes debugging lines is considered to occur on debugging lines if due to COPY processing, but not if due to REPLACE processing.

This is not consistent. Is this intentional?

Question 5:

Reference 1 contains the sentence:

"A debugging line is specified within pseudo-text if the debugging line begins in the source program after the opening pseudo-text-delimiter but before the matching closing pseudo-text-delimiter."

Reference 4, which is the equivalent general rule for REPLACE does not include this sentence.

Does some other specification of a debugging line in pseudo-text apply to REPLACE, or is the sentence just missing?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is as follows:

Question 1:

The presence of debugging lines in pseudo-text-1 of the COPY and REPLACE statements and other COPY REPLACING clauses not included within pseudo-text has no effect on the resultant program (reference 1, 4, and 6).

Question 2:

In the case of COPY processing, a text-word appears in the resultant program on a debugging line only if one of the following is true: (reference 2)

1. The COPY statement began on a debugging line.
2. Library text that was not replaced began on a debugging line.
3. The first library text-word that was involved in a match began on a debugging line.
4. Library text was replaced by pseudo-text-2 that included debugging lines.

Question 3:

A REPLACE statement specified on a debugging line has no effect on the indicator area of the line of the resultant text.

Question 4:

The standard is in error because General Rules 9 and 11 of the COPY statement conflict. Therefore, the resultant text is undefined.

Question 5:

The definition of debugging lines (reference 7) provides the specification of debugging lines and applies to debugging lines within pseudo-text.

PROPOSED CORRECTION TO ANSI X3.23-1985

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-61

SUBJECT: Two Possible Anomalies in the OCCURS Clause

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-27, paragraph 5.8.3, syntax rule 12, OCCURS clause

2. Page VI-26, paragraph 5.8.3, syntax rule 3, OCCURS clause

DATE: July 28, 1989

QUESTION:

Question 1:

What is the reason behind syntax rule 12 of the OCCURS clause? I can see no difference between the following two key specifications:

```
03  A  OCCURS 10 TIMES  ASCENDING KEY C, D.
    05  B  PIC XXX.
    05  C  PIC XXX.
    05  D  PIC XXX.
```

and

```
03  A  OCCURS 10 TIMES  ASCENDING KEY C, D.
    05  B  PIC X   OCCURS 3 TIMES.
    05  C  PIC XXX.
    05  D  PIC XXX.
```

Question 2:

On the other hand, syntax rule 3 of the OCCURS clause seems to allow the following (ridiculous) specification:

```
03  A  OCCURS 10 TIMES  ASCENDING KEY A, C.
    05  B  PIC XXX.
    05  C  PIC XXX.
```

Is that true? If so, what is the reason behind this?

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the second example presented in question 1 above is invalid. The rationale behind reference 1 was to prevent the situation in which item B from the second example above is a group item as in the following example:

```
03  A  OCCURS 10 TIMES  ASCENDING KEY C, D.
    05  B  OCCURS 3 TIMES.
        07 C  PIC XXX.
        07 D  PIC XXX.
```

Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example in question 2 is valid. There is no reason to disallow that example.

PROPOSED CORRECTION TO X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-66

SUBJECT: SORT and Output Record Varying in Size

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page XI-17, paragraph 4.4.3, syntax rule 10, SORT statement
2. Page XI-20, paragraph 4.4.4, general rule 12b, SORT statement
3. Page XI-19, paragraph 4.4.4, general rule 9, SORT statement
4. Page VII-2, paragraph 1.3.5, I-O status
5. Page VI-10, paragraph 2.5.4, general rule 1, file status clause

DATE: May 4, 1989

QUESTION:

Question 1:

What will be the size of the records of OUTPUT-FILE shown in the following COBOL source lines? Reference 2 states that a WRITE statement without any optional phrases will be executed. But it does not mention what record-name will be used in that WRITE statement.

```
DATA DIVISION.
FD  INPUT-FILE.
01  INPUT-RECORD  PIC X(50).

FD  OUTPUT-FILE, RECORD IS VARYING IN SIZE.
01  OUTPUT-RECORD-1  PIC X(50).
01  OUTPUT-RECORD-2  PIC X(60).

SD  SORT-FILE.
    03  SORT-KEY  PIC XXX.
    03  PIC X(47).
```

```
PROCEDURE DIVISION.
    SORT SORT-FILE  ON ASCENDING KEY SORT-KEY
        USING INPUT-FILE  GIVING OUTPUT-FILE.
```

Question 2:

Will the file status data item of a file for which a USE AFTER EXCEPTION/ERROR procedure is executed, be updated as specified in the introduction for the relevant file organization?

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the size of the records of OUTPUT-FILE is undefined.

Question 2:

The X3J4 interpretation of ANSI X3.23-1985 is that the file status is updated prior to executing a USE procedure. See references 4 and 5 for the sequential I-O case.

PROPOSED CORRECTION TO ANSI X3.23-1985:

X3J4 has proposed a correction to ANSI X3.23-1985 as a result of the processing of the question in document A-66. The proposed correction appears in Section 4 of this bulletin as changes to pages XI-10, XI-11 (2nd and 3rd changes), XI-12 (2nd change), XI-19, and XI-20 (2nd and 3rd changes) of ANSI X3.23-1985.

X3J4 DOCUMENT A-67

SUBJECT: EXIT PROGRAM Statement in Declarative Procedure

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page X-33, paragraph 5.4.3, syntax rule 2, EXIT PROGRAM statement
2. Page VII-51, paragraph 4.6.4, general rule 3, USE statement
3. Page X-33, paragraph 5.4.4, general rule 2, EXIT PROGRAM statement
4. Page III-17, definition of procedure

DATE: February 3, 1989

QUESTION:

Reference 1 specifies that an EXIT PROGRAM statement must not appear in a declarative procedure in which the GLOBAL phrase is specified. This rule is apparently provided so that the semantics of an EXIT PROGRAM statement within the context of the execution of a GLOBAL declarative procedure need not be defined. Unfortunately, it is not sufficient. Reference 2 allows a declarative procedure to perform other declarative procedures. Thus, a GLOBAL declarative procedure may perform a procedure in a non-global declarative procedure and that performed procedure may contain an EXIT PROGRAM statement. Such an EXIT PROGRAM statement could be treated as:

1. nonconforming to standard COBOL, or

2. a return to the program that called the program containing the EXIT PROGRAM statement (see reference 3), or

3. a return to the program that called the program that caused invocation of the USE procedure.

ANSI COBOL X3.23-1985 appears to require choice 2 above, but this can result in one or more nested programs being left in a called state after the caller has returned control to its calling program. Choice 3 might be what a COBOL application programmer expects. Reference 1 seems to intend choice 1, but is insufficient; a general rule is needed that disallows execution of an EXIT PROGRAM statement within the range of statements reference by a declarative procedure in which the GLOBAL phrase is specified.

What are the semantics of an EXIT PROGRAM statement executed within the range of statements referenced by a declarative procedure in which the GLOBAL phrase is specified?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that reference 1 and reference 4 disallow the execution of an EXIT PROGRAM statement within the range of statements referenced by a declarative procedure in which the GLOBAL phrase is specified, except within a program called while executing that declarative procedure.

X3J4 DOCUMENT A-68

SUBJECT: GLOBAL File Record with Local File Description

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page II-19, paragraph 6.2.1, names
2. Page X-2, paragraph 1.3.3, global names and local names
3. Page II-1, paragraph 2, files
4. Page II-26, paragraph 6.4.4, sharing files, Part (2)
5. Page X-24, paragraph 4.6, GLOBAL clause

DATE: February 9, 1990

QUESTION:

If a record in the file section of a containing program is GLOBAL but the file descriptor for that file is not GLOBAL, can a contained program do a WRITE or REWRITE on the file?

The record-name is GLOBAL and accessible to the contained program (see references 1 and 2), but the file-name is not accessible to the contained program (see references 1 and 2).

Since the other I-O verbs require the file-name, they clearly cannot be used on the file since the file-name itself is local to the containing program. But the verbs WRITE and REWRITE only require the record-name. This seems to imply that these operations can be done in the contained program.

Reference 3 states that the file connector is required for all accesses to the file. References 1 and 2 state that a data-name refers to a data item, and a file-name refers to a file connector. Since the file connector is required for all accesses to the file (see reference 3), the file-name must be accessible to the program doing the I-O even though it is not explicitly needed by the statement syntax. Therefore, in the example given, the contained program can do neither a WRITE nor a REWRITE to the file.

Is this interpretation correct?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that, in the example given, it is undefined whether the contained program can do a WRITE and a REWRITE to the file even though the file-name is not global.

It is the intention of X3J4 to add, in a future standard, a new rule to the GLOBAL clause which states that if the GLOBAL clause is not specified in the file description entry or report description entry of a containing program, the file may not be referenced directly or indirectly in any input-output statements in any contained program.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-79

SUBJECT: Uppercase and Lowercase Letters in Program-Name

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page IV-25, paragraph 4.2.2.1, COBOL words
2. Page IV-6, paragraph 4.2.2.1.1, user-defined words
3. Page VI-7, paragraph 3.3, PROGRAM-ID paragraph
4. Page X-27, paragraph 5.2.4, CALL statement
5. Page X-12, paragraph 3.1, PROGRAM-ID paragraph

DATE: April 6, 1989

QUESTION:

Question 1:

A called program starts with the following lines:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  AA.
```

In a calling program, the following CALL statements occur:

```
PROCEDURE DIVISION.
   CALL "aA"

   CALL "AA"
```

Will both CALL statements refer to program AA?

If yes, then does this mean that, as an exception, in an alphanumeric constant used in a CALL statement, no distinction exists between uppercase and lowercase letters?

If no, then does this mean that, as an exception, a distinction is made in the program-name between uppercase and lowercase letters?

Question 2:

Is it relevant to question 1, whether or not program AA is a contained program?

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the literal or content of the data item specified in the CALL statement must contain the program-name contained in the PROGRAM-ID paragraph of the called program (see reference 4). A program-name is a COBOL user-defined word (see reference 2). Each lowercase letter is considered to be equivalent to its corresponding uppercase letter in a COBOL word (see reference 1).

In the example "aA" and "AA" will both refer to the same program. For a COBOL program-name, no distinction exists between uppercase and lowercase letters.

Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that it is not relevant whether program AA is a contained program.

X3J4 DOCUMENT A-83

SUBJECT: EXTERNAL

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page X-2, paragraph 1.3.4, external objects and internal objects
2. Page X-7, paragraph 1.3.8.3, convention for index-names
3. Page IV-18, paragraph 4.3.8.1, second paragraph, qualification
4. Page X-23, paragraph 4.5, syntax rule 2, EXTERNAL clause
5. Page VI-27, paragraph 5.8.3, syntax rule 13, OCCURS clause
6. Page II-15, paragraph 4.2.2, subscripting using index-names

CODASYL Journal of Development 1984, revised September 1985

7. Page III-7.27-1, paragraph 7.27.4, general rule 2, EXTERNAL clause

DATE: August 30, 1989

QUESTION:

Question 1:

What is the *external attribute* for an index-name?

A number of objects are defined to have the *external attribute* (see reference 1):

1. data records in Working-Storage section having the EXTERNAL clause

2. data items subordinate to external records

3. file connectors for files defined with the EXTERNAL clause

4. data records and data items subordinate to external files

According to reference 2, index-names associated with external data items also have the external attribute.

Although there is no explicit definition of the term *external attribute* in the Standard, reference 1 provides a definition of external data items and external file connectors which can be used for all the above except for index-names (see reference 1, second paragraph). It can be attempted to treat index-names in an analogous way. But how should the following be interpreted for an index-name: "An external object may be referenced by any program in the run-unit which describes the object"?

An index-name cannot independently be described. An external index-name is associated with an external data item, that is, a data item subordinate to an external data record; there is no requirement that an external record is described identically in all programs referencing it. A table appearing in one program may be differently structured or not appear as a table at all in another one. Index-names may not have any relationship with each other.

CODASYL has recognized this ambiguity and added the following rule to the definition of the EXTERNAL clause: "If any of the data description entries in any of the record description entries contain the INDEXED phrase and the record description entries of all references to the same external record are not exactly the same, the results of the use of any indexes defined in the record description entries are undefined" (see reference 7).

Such a rule does not exist in the Standard. It also would not be in the spirit of the definition of EXTERNAL in the Standard; this rule would narrow down the shareability of external indexes to the very special case where all subordinate items are the same and have the same names. It is more likely that it was an oversight not to delete reference 2 when X3J4 accepted only a subset of the JOD definition existing at that time (requiring complete identity of all definitions of an external record). It seems much more natural to regard index-names as always internal; they can always be communicated via external index data items if necessary.

What is the X3J4 interpretation in this case?

Question 2:

What is the meaning and purpose of the following rule: "... the same data-name must not be used as the name of an external record and as the name of any other external data item described in any program contained within or containing the program which describes that external data record." (See reference 3.)

According to reference 4, the name of an external record can be the same as the name of the subordinate external data item in the same program, but according to reference 3 not in a containing or contained program, although the names of subordinate data items are only locally known (unless they are also GLOBAL).

What is the purpose of reference 3?

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI X3.23-1985 is the same as that stated in document A-58 and that is that indexes are not external. Indexes are not data and are not associated with any data hierarchy (see reference 5). The value of an index can be made accessible to an object program by storing the value in an index data item (see reference 6). Index data items may attain the external attribute by the rules pertaining to the EXTERNAL clause. Reference 2 should not have referenced the external attribute since its subject is scope of names. Scope of names refers to the recognition of names within directly or indirectly contained programs and is unrelated to the external attribute. The external attribute applies only to data items and file connectors (see reference 1).

Question 2:

The X3J4 interpretation of ANSI X3.23-1985 is that the purpose of reference 3, which states "the same data-name must not be used as the name of an external record and as the name of any other external data item described in any program contained within or containing the program which describes that external data record" is unknown.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-87

SUBJECT: RECORD Clause and I-O Status 04

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VII-47, paragraph 4.4, general rule 13, READ statement
2. Page VII-30, paragraph 3.8, syntax rule 1, RECORD clause
3. Page VII-3, paragraph 1.3.5, I-O status 04
4. Page VII-5, paragraph 1.3.7, file attribute conflict condition

DATE: July 20, 1989

QUESTION:

Which is the I-O status to be set according to reference 1 (and similar for REL and INX)?

The conditions described in general rule 13 are that the actual length of a record is less than the minimum size of the specified record description entries or greater than the maximum. This has nothing to do with the fixed file attributes maximum and minimum record length as defined in the RECORD clause, at least not for formats 1 and 2.

Example 1:

```
FD  X-FILE
        RECORD CONTAINS 100 CHARACTERS
            ...
01  RECAREA PIC X(50).
            ...
    READ X-FILE
```

This is valid according to syntax rule 1 of the RECORD clause (see reference 2). If the actual record has a length of 100, then the condition described in general rule 13 of READ is true, and an I-O status should be set. I-O status 04, however, is intended for the case that the record length "does not conform to the fixed file attributes for that file", that is, is not applicable (see reference 3).

If the actual record length is 50, general rule 13 does not apply, although in this case I-O status 04 would be appropriate. However, such a case should not occur, because, if data management works properly, all records in a file with fixed length records should have the same length; if the length is specified correctly in the RECORD clause then no record can be shorter, if not I-O status 39 should have resulted from the OPEN.

A similar situation exists concerning format 2 of the RECORD clause.

Example 2:

```
FD  X-FILE
        RECORD IS VARYING 10 TO 20 CHARACTERS
            ...
01  RECAREA PIC X(15).
            ...
    READ X-FILE
```

If the actual record has a length of 20, then the condition described in general rule 13 of READ is true, and an I-O status should be set. I-O status 04, however, is intended for the case when the record length "does not conform to the fixed file attributes for the file", that is, is not applicable.

If the actual record length is 5, general rule 13 does apply, and in this case I-O status 04 would be appropriate. However, such a case again should not occur, if the actual minimum and maximum length was correctly specified in the RECORD clause. There may be, however, cases where a violation is not detectable at OPEN time; but in this case the READ should not be regarded as successful, and a permanent error should be indicated, for example, with I-O status 30.

It appears that I-O status 04 is really intended for the general rule 13 cases, not for fixed file attribute conflicts.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that it is undefined when I-O status 04 is returned and it is undefined what I-O status code is generated by general rule 13 of the sequential READ statement.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-88

SUBJECT: RECORD IS VARYING

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VII-30, paragraph 3.8.3, syntax rule 2, RECORD clause

2. Page VII-4, paragraph 1.3.5, item 4d, I-O status 44
   Page VIII-5, paragraph 1.3.4, item 5d, I-O status 44
   Page IX-5, paragraph 1.3.4, item 5d, I-O status 44

3. Page VII-31, paragraph 3.8.4, general rule 10a, RECORD clause

4. Page VII-53, paragraph 3.8.4, general rule 8, WRITE statement
   Page VIII-38, paragraph 4.9.4, general rule 8, WRITE statement
   Page IX-42, paragraph 4.9.4, general rule 8, WRITE statement

5. Page VII-47, paragraph 4.4.4, general rule 13, READ statement
   Page VIII-29, paragraph 4.5.4, general rule 16, READ statement
   Page IX-32, paragraph 4.5.4, general rule 18, READ statement

DATE: April 20, 1989

QUESTION:

Reference 1 states that record descriptions must not describe records that contain a lesser number of character positions than integer-2 or a greater number than integer-3. Therefore, the following example is valid:

```
FD  VFILE  RECORD IS VARYING FROM 10 TO 50 CHARACTERS
       DEPENDING ON RSIZE.
01  RECORD-1  PIC X(20).
01  RECORD-2  PIC X(30).
```

What is expected to happen if the following statements are executed?

```
MOVE 50 TO RSIZE.
WRITE RECORD-2.
```

According to reference 3, the number of characters to be written is determined by the content of the data-name (in the example, 50).

According to reference 4, the number of characters associated with a record description is determined by the sum of all the data items (in the example, 40).

Status 44 (see reference 2) does not apply because the value of the data-name does not exceed the largest or smallest record allowed by the RECORD IS VARYING clause.

Rather than change the definition of status code 44 to encompass this situation at runtime, it seems that syntax rule 2 (see reference 1) should require integer-3 to equal the largest record description. It seems that the VARYING clause (with or without the DEPENDING ON phrase) is most useful and clear when used with a single record description whose size equals integer-3. If used for multiple record descriptions, integer-3 should equal the largest record description and integer-2 should equal the smallest.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that, for the WRITE statement in the given example, 50 characters are written and the WRITE is successful (see reference 3) with an I-O status code of 00 (see reference 4).

X3J4 DOCUMENT A-89

SUBJECT: Fixed File Attribute Logical Record Size

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VII-3, paragraph 1.3.5, item 1b, I-O status 04
2. Page VII-4, paragraph 1.3.5, item 3f, I-O status 39
3. Page II-1, paragraph 2.1, file attributes
4. Page II-3, paragraph 2.1.4.2, variable length records
5. Page VII-21, paragraph 3.1.1, file description entry
6. Page VII-43, paragraph 4.3.4, general rule 25, OPEN statement
7. Page VII-30, paragraph 3.8.4, general rule 1, RECORD clause
8. Page VII-30, paragraph 3.8.1, function, RECORD clause
9. Page VII-31, paragraph 3.8.4, general rule 4, RECORD clause

DATE: March 9, 1989

QUESTION:

When an OPEN statement is executed and the minimum and maximum logical record sizes specified in the file description are checked for conformance with the fixed file attributes of the file being opened, is the check made for equality with the file attribute or for exceeding the bounds of the file attribute?

If equality is required, then all programs in an application which reference that file must always describe the largest and smallest record. This is not consistent with the intent and the spirit of using variable length record files. Certain programs in an application using variable length record files may only require or be permitted to manipulate portions of records.

However, if the program is merely required to specify a file description whose minimum and maximum record sizes are within the bounds of the fixed file attributes of the file, then status code 04 does not adequately describe the circumstances that would result in its being returned.

If ANSI COBOL X3.23-1985 is interpreted as requiring equality in minimum/maximum record sizes between the file description and the fixed file attributes, I would request that consideration be given for a change to a future revision of ANSI COBOL X3.23-1985 to remove such a requirement.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that, for the file referenced by an OPEN statement, (1) the file description specification of the minimum logical record size must be equal to the fixed file attribute minimum logical record size, and (2) the file description specification of the maximum logical record size must be equal to the fixed file attribute maximum logical record size. Otherwise a fixed file attribute conflict condition exists.

When the RECORD IS VARYING clause is specified, the file description minimum and maximum logical record sizes may be specified in that RECORD clause. In this case, it is not necessary to provide record description entries to specify the minimum and maximum logical record sizes (see references 5, 7, 8, and 9).

It is undefined in ANSI X3.23-1985 under what circumstances I-O status value 04 is returned.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-90

SUBJECT: Zero Length Groups

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-26, paragraph 5.8.3, syntax rule 5, OCCURS clause
2. Page VI-28, paragraph 5.8.4, general rule 3, OCCURS clause
3. Page VI-54, paragraph 6.3.1.1, relation condition
4. Page VI-56, paragraph 6.3.1.2, class condition
5. Page II-14, paragraph 4.4, subscripting
6. Page VI-27, paragraph 5.8.4, general rule 2, OCCURS clause
7. Page IV-9, paragraph 4.2.2.2.1, nonnumeric literals
8. Page VI-55, paragraph 6.3.1.1.2, comparison of nonnumeric operands
9. Page IV-16, paragraph 4.3.6, standard alignment rules

DATE: April 26, 1989

QUESTION:

A zero length group item can be declared using **OCCURS 0 TO ... DEPENDING ON**. Is this legal? If so, do two zero length groups equal each other, and is a zero length group both alphabetic and numeric (or neither)?

Reference 1 says the first integer in an OCCURS DEPENDING can be zero. Reference 2 says when a group item containing an item with an OCCURS DEPENDING is referenced its length depends on the current value of the DEPENDING ON item. I can find no rule to prevent the item with the OCCURS DEPENDING clause being the only item subordinate to the group item, so it would seem the group can be zero length.

Reference 3 seems to imply any two zero length groups are equal. Reference 4 seems to imply that every zero length group is both alphabetic and numeric. However, neither really seems to have been written with this case in mind.

X3J4 RESPONSE:

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

1. Zero length groups are legal (see reference 1).

2. Two zero length groups in a relation condition contain no unequal corresponding pairs of characters (see reference 8) and, therefore, are considered equal.

3. If only one operand in a relation condition is a zero length group, it follows the rules for operands of unequal sizes (see reference 8); i.e. it is extended on the right by sufficient spaces to make the operands equal size.

4. A zero length group in a class condition will always return false because it does not consist of characters of the required type (see reference 4).

PROPOSED CORRECTION TO ANSI X3.23-1985:

X3J4 has proposed a correction to ANSI X3.23-1985 as a result of the processing of the question in document A-90. The proposed correction appears in Section 4 of this bulletin as changes to pages IV-11 and VI-57 of ANSI X3.23-1985.

X3J4 DOCUMENT A-91

SUBJECT: ALPHABET Clause

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-13, paragraph 4.5.3, syntax rule 3, ALPHABET literal clause
2. Page VI-16, paragraph 4.5.4, general rule 4d, item 3, ALPHABET clause
3. Page VI-16, paragraph 4.5.4, general rule 4d, item 5, ALSO phrase of ALPHABET clause

DATE: May 5, 1989

QUESTION:

Reference 1 states: "If the literal phrase of the ALPHABET clause is specified, a given character must not be specified more than once in that clause."

Question 1:

This rule does not handle the problem of implied re-specification with the THRU clause. Is the following example valid?

```
ALPHABET alphabet-name IS "A" THRU "M",  "D" THRU "H".
```

According to reference 1, this example is valid, but I don't think that the intent of this rule is to allow it. The same logic problems that present themselves when a given character is repeated, also exist in the above example.

I think the rule should read: "... a given character must not be specified, either explicitly or implicitly, more than once in that clause."

Question 2:

Syntax rule 3 is too restrictive when the ALSO phrase is used. Literal-1 should reasonably be repeated when ALSO is used. Repeating a given character with ALSO seems more useful than introducing new characters with ALSO. To equate ordinal positions with characters that are inside the collating sequence being specified provides more application flexibility than equating ordinal positions of characters outside the collating sequence being specified. Consider the following example:

```
ALPHABET alphabet-name IS  "A" THRU "Z",
                           ".",
                           "." ALSO "," ALSO ";" ALSO ":"
                               ALSO "?".
```

Is the intent of syntax rule 3 to only equate ordinal positions of characters that are not in a previously specified literal-1? I think the rule should be appended with the following: "However, literal-1 (when specified without an ALSO phrase) or literal-2 may be repeated within the same ALPHABET clause if the character is repeated as a literal-1 or literal-3 that is associated with an ALSO phrase."

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example is invalid since **"D" THRU "H"** is implicitly repeating identical characters. ANSI COBOL X3.23-1985 disallows specifying a given character, whether explicitly or implicitly, more than once (see references 1 and 2).

Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the intent of references 1 and 3 is to allow equating ordinal positions only of characters that have not previously been specified in the ALPHABET clause. The example is invalid according to reference 1 since the period character specified as the nonnumeric literal "." appears more than once.

X3J4 DOCUMENT A-92

SUBJECT: BEFORE/AFTER Phrases in the INSPECT Statement

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-95, paragraph 6.18.3, syntax rule 5, INSPECT statement
2. Page VI-95, paragraph 6.18.3, syntax rule 8, INSPECT statement
3. Page I-17, summary of elements entry for INSPECT

DATE: June 6, 1989

QUESTION:

The rules for length of BEFORE/AFTER characters in INSPECT are unclear and appear to be inconsistent between INSPECT TALLYING formats 1 and 3.

For Format 1:

According to syntax rule 5, literal-2 or the data-item referenced by identifier-4 must be one character in length for all formats in level 1. Except as otherwise noted, this restriction does not apply to level 2.

Does syntax rule 5 mean that:

1. When compiling at high level, the length of literal-2 and the data item referenced by identifier-4 in format 1 can be greater than 1? And when compiling at the intermediate level, the length is restricted to 1?

or

2. The length is restricted to 1 in either case because the general format for format 1 is described as level 1? (ref "... all formats in level 1") ??

For Formats 2 and 3:

According to syntax rule 8, literal-2 or the data item referenced by identifier-4 must be one character in length for Formats 2 and 3.

Does this mean that:

1. Syntax rule 8 restricts both level 1 and level 2 to one character, because syntax rule 8 is one of the "otherwise noted in syntax and general rules" mentioned in syntax rule 5?

or

2. Syntax rule 8 restricts only level 1 to one character because syntax rule 8 does not specifically mention level 2? (If this were the case, wouldn't syntax rule 8 be unnecessary since syntax rule 5 is sufficient?)

Conclusion:

Our best guess is that the interpretations of syntax rules 5 and 8 are both #1, giving these results:

	Level 1	**Level 2**
Format 1	1 character	> 1 character
Formats 2 and 3	1 character	1 character

If these are the correct interpretations, then the rules for TALLYING formats 1 and 3 are inconsistent. The summary of elements on page I-17, first 2 entries under INSPECT, indicates consistency.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that the syntax rules for format 1 are less restrictive than the syntax rules for formats 2 and 3 at level 2 of the Nucleus module. Reference 2 restricts the operand of the BEFORE and/or AFTER phrase to one character regardless of level for formats 2 and 3. No such restriction applies for format 1, so reference 1 allows the operand of the BEFORE and/or AFTER phrase to be more than one character in length for level 2.

PROPOSED CORRECTION TO ANSI X3.23-1985:

X3J4 has proposed a correction to ANSI X3.23-1985 as a result of the processing of the question in document A-92. The proposed correction appears in Section 4 of this bulletin as a change to page VI-95 of ANSI X3.23-1985.

X3J4 DOCUMENT A-93

SUBJECT: NOT ON SIZE ERROR Phrase

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-67, paragraph 6.4.2, ON SIZE ERROR phrase

DATE: March 30, 1989

QUESTION:

On page VI-67 the second full paragraph associated with reference 1 states: "If the ON SIZE ERROR phrase is specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers remain unchanged ...".

On page VI-68 the third full paragraph associated with reference 1 states: "If the ON SIZE ERROR phrase is not specified and a size error condition exists ... , the values of the affected resultant identifiers are undefined."

Is the NOT ON SIZE ERROR phrase treated the same as the ON SIZE ERROR phrase with respect to the value of the affected resultant identifiers when a size error condition exists?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the two phrases NOT ON SIZE ERROR and ON SIZE ERROR are distinct phrases when referenced in the rules. Reference 1 states that "If the ON SIZE ERROR phrase is not specified and a size error condition exists ... , the value of the affected resultant identifiers are undefined." The presence of a NOT ON SIZE ERROR phrase alone does not ensure that in the event of a size error condition that the resultant identifiers will remain unchanged. Only the presence of the ON SIZE ERROR phrase will ensure that the values of the resultant identifiers are unchanged in the event of a size error condition.

X3J4 DOCUMENT A-94

SUBJECT: CLOSE REEL/UNIT

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VII-35, paragraph 4.2.4, general rule 1, CLOSE statement
2. Page VII-35, paragraph 4.2.4, first paragraph, CLOSE statement
3. Page VII-37, paragraph 4.2.4, general rule 3f, CLOSE statement
4. Page III-25, definition of unit

DATE: March 3, 1989

QUESTION:

ANSI X3.23-1985 is overly restrictive in its definition of non-reel/unit in the CLOSE statement. Reference 1 classifies non-reel/unit as a medium for which rewind and reels/units have no meaning. For the purpose of CLOSE, reel and unit are synonymous and completely interchangeable (see reference 2).

This leads to a problem that files assigned to multi-unit mass storage do not have the expected behavior of volume switching for CLOSE with the UNIT option. These files have a unit characteristic without having a rewind characteristic. The strict classification of such files as non-reel/unit by CLOSE general rule 2 results in the following:

1. No volume switch (see reference 3)
2. Setting of status code 07 meaning "... the file is on a non/reel/unit medium."

Is my understanding correct?

The problem is the blanket classification of media without rewind as non-reel/unit. A lesser problem is the ambiguity of whether unit means anything other than reel in the general rules of CLOSE.

I suggest that ANSI COBOL X3.23-1985 be modified to classify a file assigned to multi-unit mass storage devices as sequential multi-reel/unit. This can be done by changing general rule 2a to read:

"a. Non-reel/unit. A file whose input or output medium is such that the concept of rewind has no meaning or the concept of multiple volumes or multiple reels has no meaning."

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that files that have a unit characteristic but no rewind characteristic are not classified as non-reel/unit media because reference 1 requires that both of the concepts of rewind and reels/units have no meaning for a file to be classified as non-reel/unit. If either of the concepts has meaning for the file, it is classified as either single-reel/unit or multi-reel/unit. According to the glossary (see reference 4) reel and unit are synonymous. Therefore there is no ambiguity in the rules for the CLOSE statement.

X3J4 DOCUMENT A-95

SUBJECT: Comparing Index-Name to Arithmetic Expression

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-54, paragraph 6.3.1.1, relation condition
2. Page VI-56, paragraph 6.3.1.1.3, comparisons involving index-names and/or index data items
3. Page VI-55, paragraph 6.3.1.1.1, comparison of numeric operands

DATE: April 24, 1989

QUESTION:

Can an index-name (not an index data item) be compared to an arithmetic expression?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that an index-name cannot be compared to an arithmetic expression. According to reference 2, for an index-name, relation tests may be made only between:

1. Two index-names
2. An index-name and a data item (other than an index data item) or a literal
3. An index-name and an index data item.

X3J4 DOCUMENT A-96

SUBJECT: Maximum Length Receiver and Reference Modification

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page IV-23, paragraph 4.3.8.3.4, general rule 4b, reference modification
2. Page VI-28, paragraph 5.8.4, general rule 3b, OCCURS clause

DATE: April 5, 1989

QUESTION:

Given the following example:

```
DATA DIVISION.
01  REC.
    02  NUM  PIC 999.
    02  TABL  OCCURS 0 TO 100 TIMES DEPENDING ON NUM.
        03  PIC X.

PROCEDURE DIVISION.
   MOVE 10 TO NUM.
   MOVE ALL "5" TO REC(1:).
```

When length is unspecified for reference modification, reference 1 says the length extends to include the rightmost character of the data item referenced by data-name-1.

In the last MOVE statement in the above example, what is the length of REC, and therefore the length of the unique elementary item created by reference modification (see reference 1)?

Does REC have the maximum length of the group, 103, as a receiving item per reference 2?
or,
Is the maximum length receiver rule (see reference 2) inapplicable because the receiving item is not the group item REC itself but is instead the unique elementary data item created by reference modification, REC(1:)?

We have interpreted that the length of the data item referenced by data-name-1 in reference modification is determined by the same rules that would apply if data-name-1 were used without reference modification. That is, the length of the group item containing a subordinate item with the OCCURS DEPENDING ON (where the DEPENDING ON variable is within the group) is dependent on whether the item is used in a sending or receiving context (see reference 2). Therefore, the length of the data item defined by REC(1:) would be 103.

Is that interpretation correct?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that reference modification creates a unique data item from the referenced data item. The length of this referenced data item is determined by first applying the rules of reference 2. Subsequently, the rules of reference modification are applied to determine the length of the unique data item created through reference modification.

In the example shown on the previous page, the reference to REC in the last MOVE statement creates a unique data item with a length of 103.

X3J4 DOCUMENT A-97

SUBJECT: Unresolved PERFORM Statements

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-120, paragraph 6.21.4, general rule 14, PERFORM statement
2. Page VI-111, paragraph 6.21.4, general rule 7, PERFORM statement
3. Page I-6, paragraph 1.5.2, definition of a conforming implementation
4. Page I-9, paragraph 1.6, definition of a conforming source program
5. Page I-9, paragraph 1.7, relationship of a conforming program to a conforming implementation
6. Page I-6, paragraph 1.5.1, definition of subsets

DATE: March 30, 1989

QUESTION:

Given the following example:

```
PROCEDURE DIVISION.
A-MAIN.
    DISPLAY "BEFORE A THRU A-EXIT".
    PERFORM A THRU A-EXIT.
    DISPLAY "AFTER A THRU A-EXIT".
    STOP RUN.

A.  DISPLAY "BEFORE B THRU B-EXIT".
    PERFORM B THRU B-EXIT.
    DISPLAY "AFTER B THRU B-EXIT".
A-EXIT.  EXIT.
```

```
B.  ADD +1 TO WORK-VALUE.
    DISPLAY "B BEFORE IF".
    IF WORK-VALUE NOT GREATER THAN 1
        GO TO A-EXIT.
    DISPLAY "B AFTER IF".
B-EXIT.  EXIT.
```

General rule 14 of the PERFORM statement appears to explicitly prohibit the construct shown on the preceding page because the active **PERFORM B THRU B-EXIT** allows control to pass to the exit of the other active PERFORM statement (A-EXIT). However, the existence of this construct in the static program does not mean that it will occur during execution (the condition **WORK-VALUE NOT GREATER THAN 1** may never be true).

Question 1:

a. Should a conforming compiler detect this situation and indicate an error?

b. Should the detection occur statically (at compile time) or dynamically (during execution)?

c. Does the conditional nature of the transfer to A-EXIT affect the answer to question 1b above; that is, should an unconditional transfer cause a compile time error while detection of the error resulting from a conditional transfer be deferred until execution?

Question 2:

If a conforming compiler does not have to detect an error, what should be the execution result of the program? Or, are the results of execution unpredictable? The possible execution results appear to be (assuming the condition is at least initially true):

If execution of paragraph A-EXIT is not recognized as termination of the A THRU A-EXIT range (because B THRU B-EXIT is active), assuming an initial value of zero for WORK-VALUE, the result of the DISPLAY statements would be:

1. BEFORE A THRU A-EXIT
2. BEFORE B THRU B-EXIT
3. B BEFORE IF
4. B BEFORE IF
5. B AFTER IF
6. AFTER B THRU B-EXIT
7. AFTER A THRU A-EXIT

If execution of paragraph A-EXIT terminates the A THRU A-EXIT range, regardless of B THRU B-EXIT being active, the result of the DISPLAY statements would be:

1. BEFORE A THRU A-EXIT
2. BEFORE B THRU B-EXIT
3. B BEFORE IF
4. AFTER A THRU A-EXIT

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that conforming implementations are not required to detect the situation described in the example at compile time or at execution time. The transfer to A-EXIT does not affect the answer to question 1b. Conforming implementations are required to support subsets of ANSI X3.23-1985 defined in reference 6. Reference 4 states that "In order for a source program to conform to Standard COBOL, it must not include any language elements not specified in this standard. The execution of a program, the source text of which conforms to Standard COBOL, is predictable only to the extent defined in this standard. The results of violating the formats or rules of Standard COBOL are undefined unless otherwise specified in this standard." Reference 5 indicates "The translation of a conforming source program by a conforming implementation and the subsequent execution of the resultant object program is defined only to the extent specified in Standard COBOL."

Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the program presented in the example is a nonconforming program because it explicitly violates provisions and specifications of Standard COBOL (see reference 1). The results of violating the formats or rules of Standard COBOL are undefined unless otherwise specified in this standard. Conforming implementations are not required to detect the situation described in the example at compile time or at execution time.

X3J4 DOCUMENT A-98

SUBJECT: ALPHABET Clause and CODE-SET Clause

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-15, paragraph 4.5.4, General rule 4d, ALPHABET literal and CODE-SET

2. Page VI-16, paragraph 4.5.4, general rule 4d, item 5, ALPHABET literal and ALSO

3. Page VI-15, paragraph 4.5.4, general rule 4, ALPHABET clause

DATE: July 21, 1989

QUESTION:

Reference 1 states that if the literal phrase is specified, the alphabet-name may not be referenced in a CODE-SET clause.

Reference 2 states: "If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1 and literal-3 are assigned to the same ordinal position in the collating sequence being specified or in the character code set that is used to represent the data ... ."

If CODE-SET is not allowed when the literal phrase is used with ALPHABET, why is CODE-SET referenced in reference 2? It is contradictory with reference 1. Note that this phrase did not exist in ANSI COBOL X3.23-1974. It does exist in CODASYL COBOL Journal of Development, which unlike ANSI COBOL X3.23-1985, allows CODE-SET for a literal alphabet clause. Is this an editorial problem?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that this is not an editorial problem, and the character code set is referred to by reference 2 because of the possibility of using alphabet-name-1 in a SYMBOLIC CHARACTERS clause (see reference 3).

X3J4 DOCUMENT A-99

SUBJECT: De-Editing and Insertion Character "0"

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page III-7, definition of de-edit
2. Page III-8, definition of editing character
3. Page VI-105, paragraph 6.19.4, general rule 4b, MOVE statement

DATE: June 12, 1989

QUESTION:

During de-editing, we are concerned with the need to differentiate an insertion character "0" from the numeric value 0. Is it intended that the insertion character "0" be de-edited from a character-string when de-editing applies?

```
DATA DIVISION.
01  NUMERIC-EDITED  PIC 990099  VALUE IS "010023".
01  NUMERIC-VAR  PIC 9(9).

PROCEDURE DIVISION.
    MOVE NUMERIC-EDITED TO NUMERIC-VAR.
```

Does NUMERIC-VAR contain 123 or 10023?

There does not appear to be any rules to imply that NUMERIC-VAR will not contain 123. However, we are curious if the rules for de-editing had this case in mind.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the simple insertion character "0" (zero) is removed when de-editing is implied to establish the unedited numeric value of an operand. De-editing is defined as the logical removal of all editing characters from a numeric edited data item (see reference 1) and the character "0" (zero) is listed as one of the editing characters (see reference 2). The MOVE statement in the example implies de-editing (see reference 3). Therefore the value of the data item referenced by NUMERIC-VAR will be 123 following execution of the example MOVE statement if the value of NUMERIC-EDITED is not changed prior to its execution.

X3J4 DOCUMENT A-225

SUBJECT: NOT GREATER OR EQUAL

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-54, Section 6.3.1.1, Relation Condition
2. Page VI-61, Section 6.3.3, Abbreviated Combined Relation Conditions
3. Page IV-3, Section 2.1.7, Use of Special Character Words in Formats

DATE: October 18, 1991

QUESTION:

The standard seems to contain an inconsistency:

Page VI-54 6.3.1.1 shows NOT only preceding the relational operators GREATER, >, LESS, <, EQUAL, and =. It is invalid preceding GREATER OR EQUAL, >=, LESS OR EQUAL, and <=.

Page VI-61 6.3.3 states:

The interpretation ... of NOT in an abbreviated combined relation condition is as follows:

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

This seems to say that the expression

IF A EQUAL B OR NOT GREATER OR EQUAL C OR D

is expanded into

```
IF (A EQUAL B) OR (A NOT GREATER OR EQUAL C)
    OR (A NOT GREATER OR EQUAL D)
```

which is invalid according to VI-54 6.3.1.1. However, the same expression written with the "short" operators is interpreted differently (since the word >= isn't in the list on page VI-61):

```
IF A = B OR NOT >= C OR D
```

expands to

```
IF (A = B) OR (NOT (A >= C)) OR (A >=D)
```

This is valid but certainly not what would be expected. I think the user expectation would be:

1. that >= and <= behave like >, <, = if the NOT allowed, and
2. that the symbols behave like their "long" equivalents

Both will be violated if we strictly apply the rules. The intent of the standard was clearly NOT to allow the "NOT" in combination with the >= etc. It is consistent with this intent that the first example is invalid.

The second example should have been invalid as well. I think X3J4 should pass an interpretation disallowing the second example.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is the following:

Reference 1 and Reference 2 contradict each other. Thus, the expansion of all of your examples (both those with "NOT >=" and those with "NOT GREATER OR EQUAL") are undefined in the Standard.

As you have suggested, the intention of third Standard COBOL was that the COBOL word "NOT" would be invalid before both the COBOL word ">=" and the relational operator "GREATER OR EQUAL" whether in a simple relational expression or in an abbreviated combined relation condition. (Similarly, it would be invalid before all other relational operators equivalent to ">=" and "<="; no matter how they are written.)

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

NOTE:

One result of this interpretation is that the interpretation presented in A-22 (published in CIB-24) has been reversed. X3J4 no longer interprets the examples as valid source code, i.e. the following statements are interpreted by X3J4 as non-standard source code:

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

X3J4 DOCUMENT A-263

SUBJECT: NOT Phrases and USE Procedures

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VII-46, paragraph 4.4.4, general rule 11b, READ statement
   Page VIII-29, paragraph 4.5.4, general rule 11b, READ statement
   Page IX-31, paragraph 4.6.4, general rule 11b, READ statement

2. Page VII-51, paragraph 4.6.4, general rule 6, USE statement
   Page VIII-36, paragraph 4.8.4, general rule 6, USE statement
   Page IX-40, paragraph 4.8.4, general rule 6, USE statement

3. Page VII-2, paragraph 1.3.5, third paragraph
   Page VIII-2, paragraph 1.3.4, third paragraph
   Page IX-2, paragraph 1.3.4, third paragraph

4. Page VII-46, paragraph 4.4.4, general rule 11c, READ statement

5. Page VII-46, paragraph 4.4.4, general rule 10b, READ statement

DATE: September 21, 1990

QUESTION:

The following questions apply to NOT AT END and NOT INVALID KEY for all modules in which they appear, even though NOT AT END in the sequential I-O module is discussed. Considerations are the same.

Problem 1: Non-critical errors and NOT phrases

Reference 1 says "If an exception condition which is not an at end condition exists, control is transferred according to the rules of the USE statement following the execution of any USE AFTER EXCEPTION procedure applicable to file-name ..." and refers to reference 2. Reference 2 describes what happens after execution of the USE procedure.

Reference 1 might mean:

a. whether or not there is an applicable USE procedure, the rules for transfer of control are those described for the USE statement,

b. following the execution of any USE AFTER EXCEPTION procedure applicable ... , the rules for transfer of control are those described for the USE statement.

(that is, the USE statement only specifies the rules for transfer of control in the event there is a USE statement.) ??

If reference 1 means item a above, then transfer of control for a non-zero critical error is to the next executable statement following the statement whose execution caused the exception, both when a USE procedure exists and when one does not. If there is a NOT phrase, it is ignored.

If reference 1 means item b above, then the transfer of control for a non-critical error is to the next executable statement ... when there is a USE procedure. Transfer of control is unspecified if there is no USE procedure, leaving open the question of whether the imperative statement associated with the NOT phrase is executed.

This seems to make a difference only for non-critical 9x conditions since all the other status codes are successful, end of file, invalid key, or critical errors.

Questions for Problem 1:

1. Does reference 1 mean either item a or b above?

2. What is the transfer of control in the case of a non-critical input-output error and in the absence of an applicable USE procedure? Is the imperative statement associated with the NOT phrase executed?

Example:

Assume a non-critical error on the READ statement in the following example; is SUB1 performed?

```
FILE-CONTROL.
     SELECT FILE1 ASSIGN TO SYS015
        ORGANIZATION IS SEQUENTIAL
        ACCESS IS SEQUENTIAL.
        ...
FD FILE1
        ...
* No exception procedure
        ...
     OPEN INPUT FILE1.
     READ FILE1 INTO ...
          NOT AT END PERFORM SUB1.
```

Problem 2: Critical errors and NOT phrases

For a critical error condition, reference 3 says the implementor determines what action is taken after execution of any applicable USE procedure, or if none applies, after completion of IOCS error processing. I believe the intent is to allow the implementor to halt execution or take recovery action in the event of a critical error. In the case where the implementor chooses to continue processing, the Standard does not specify subsequent transfer of control; it seems unlikely that the intent is for the implementor to also determine that.

Questions for Problem 2:

1. Following a critical input-output error, if the implementor allows processing to continue, does the implementor also determine the subsequent transfer of control?

2. If not, is the imperative statement associated with the NOT phrase executed?

X3J4 RESPONSE:

Problem 1, Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the rules for transfer of control are those for the USE statement, whether or not there is an applicable USE procedure.

Problem 1, Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that control is transferred to the next executable statement. The imperative statement associated with the NOT phrase is not executed. SUB1 is not performed. This applies to any exception condition except an at end condition.

Problem 2, Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the implementor does not control subsequent transfer of control. Control would transfer to the next executable statement.

Problem 2, Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the imperative statement associated with the NOT phrase is not executed.

X3J4 DOCUMENT A-283

SUBJECT: Implementor-defined RECORD DELIMITER Clause

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page XII-13, paragraph 2.8.4, general rule 3, RECORD DELIMITER clause
2. Page XVII-91, item 55, implementor-defined language element list
3. Page I-7, paragraph 1.5.2.1, substitute or additional language elements

DATE: August 8, 1989

QUESTION:

Reference 1 discusses an implementor-defined method for determining the length of variable length records through the use of the RECORD DELIMITER clause. The wording of reference 1 implies that a conforming implementation must provide at least one implementor-defined method for determining variable length records.

We believe that the intent of ANSI COBOL X3.23-1985 in providing the implementor-name-1 option was to provide syntax for implementors that supported other (nonstandard) methods for determining the length of variable length records. The intent was to permit (and not to require) support of these nonstandard methods. These nonstandard methods would most commonly be the default implementor methods which preexisted ANSI X3.23-1978 Magnetic Tape Labels and File Structure for Information Interchange.

Is it the intent of X3J4 to require nonstandard record delimiter mechanisms of conforming implementations? If not, we suggest that references 1 and 2 should be changed to indicate that conforming implementations may provide alternate methods.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that a conforming implementation is not required to define an implementor-name for use in the RECORD DELIMITER clause.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-285

SUBJECT: I-O Status 44 and RECORD clause

REFERENCE:

American National Standard COBOL X3.23-1985

1. Page XVII-88, (16) Implementor-Defined Element List, RECORD clause
2. Page VII-4, Section 1.3.5 (3d), I-O Status 44
3. Page VII-32, SR (14), Format 3 of the RECORD Clause

DATE: April 26, 1991

QUESTION:

If a file is described with format 3 of the record clause (RECORD CONTAINS integer to integer), is it required to obtain I-O status 44 on a REWRITE to the file?

I believe that it is not possible because the standard does not require the implementor to write variable length records for RECORD CONTAINS (for 1974 compatibility). If the original length of the record is not known, there is no way to detect a length conflict.

Since variable length records can be described only with format 2 of the RECORD clause (RECORD VARYING) which is a level 2 feature, it appears that all of the description of I-O status 44 as well as General Rule 10 of Sequential REWRITE should have been boxed.

X3J4 RESPONSE:

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

If the implementor has defined that fixed-length records are obtained with format 3 of the record clause (RECORD CONTAINS integer-4 TO integer-5), then I-O status code 44 does not apply. I-O status code 44 applies only to files with a format 2 record clause (RECORD IS VARYING) and to files with a format 3 record clause (RECORD CONTAINS integer-4 TO integer-5) when the implementor defines that variable-length records are obtained.

PROPOSED CORRECTION FOR ANSI X3.23-1985

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-286

SUBJECT: Moving Alphanumerics to Numerics

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-104, paragraph 6.19.4, general rules 3 and 4b-3, MOVE statement
2. Page III-12, definition of integer
3. Page VI-70, paragraph 6.4.7, incompatible data
4. Page VI-34, paragraph 5.9.5, editing rule 7, PICTURE clause
5. Page IV-17, paragraph 4.3.6, standard alignment rule 2
6. Page IV-11, paragraph 4.2.2.2.3, figurative constant values, second item 1

DATE: July 23, 1989

QUESTION:

What should be the result of the following moves:

```
MOVE "123456.789" TO A

MOVE "123.456789" TO A

MOVE ALL "8" TO B

MOVE ALL "8" TO C
```

Where A, B, C are:

```
01  A PIC 999,999.999
01  B PIC $$9.
01  C PIC $$9.9.
```

Are the first two example above valid moves or invalid moves.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that for the examples stated, data is moved as if the sending operand were described as an unsigned numeric integer. (See reference 1.)

Since the literals in the first two examples do not conform to this requirement, (see reference 2) they violate the MOVE statement general rules and therefore, the results of these operations are implicitly undefined.

The last two examples are conforming source code and the result of the operations would be as follows:

- B would contain $88
- C would contain $88.0

(See references 4, 5, and 6.)

NOTE:

With the publication of CIB-26, X3J4 confirms that the intent of the response in interpretation A-44, dated August 20, 1987 and published in CIB-24 dated January 1988 is correct. However, it is also viewed that the following wording would have been more clear,

> "The X3J4 interpretation of ANSI COBOL X3.23-1984 is that since the literals in these examples do not conform to the requirements for sending operands for the MOVE statement, the results of these operations are implicitly undefined."

X3J4 DOCUMENT A-287

SUBJECT: PICTURE IS P$$

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Pages VI-34 and 35, paragraph 5.9.5, editing rule 7, PICTURE clause
2. Page VI-37, paragraph 5.9.6, precedence rules, PICTURE clause

DATE: September 21, 1990

QUESTION:

This is a request for clarification on an apparent contradiction between the rules for floating insertion editing and the table of precedence rules.

Note: Throughout this interpretation request, the symbol $ will be used to represent the currency symbol. However, the same rules should apply to any currency symbol.

The fourth paragraph on page VI-35 (see reference 1) states: "If all numeric character positions in the PICTURE character-string are represented by the insertion character, at least one of the insertion characters must be to the left of the decimal point."

In the table of precedence rules (see reference 2), there is a letter X for a first symbol of P to the right of the decimal point and a second symbol of a floating insertion cs to the right of the decimal point.

Question 1:

The table would appear to support while the rule would appear to prohibit a field with:

```
PICTURE IS P$$
```

Which is correct?

Question 2:

Is it true that the definition for the character P in a PICTURE clause allows

```
PICTURE IS $$P
```

but prohibits

```
PICTURE IS $P$$
```

and

```
PICTURE IS $VP$$
```

Question 3:

So that the interpretation does not introduce any new problems, is it true that

```
PICTURE $V$$
```

is valid while

```
PICTURE V$$
```

is invalid?

Question 4:

Will the same answers hold if these PICTURE character-strings have + or - instead of $, for example P - - ?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that the third and fourth paragraphs of editing rule 7 (see reference 1) state that the second floating character from the left through the rightmost represent the characters which may be replaced by non-zero numeric data while the leftmost represents the leftmost limit of the floating symbols.

Question 1:

The rule adds a restriction to the precedence rule table, and the example is prohibited.

Question 2:

It is true that the first example is allowed and the second and third examples are prohibited, due to the precedence table not allowing a currency symbol to the left of the decimal point to be followed by a P symbol to the right of the decimal point.

Question 3:

The first example is valid and the second example is invalid.

Question 4:

The answers are the same with any floating insertion character.

PROPOSED CORRECTION TO ANSI X3.23-1985:

X3J4 has proposed a correction to ANSI X3.23-1985 as a result of the processing of the question in document A-287. The proposed correction appears in Section 4 of this bulletin as a change to page VI-37 of ANSI X3.23-1985.

X3J4 DOCUMENT A-288

SUBJECT: Scope of Configuration Section

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page X-4, paragraph 1.3.8, scope of names
2. Page VI-9, paragraph 4.2, Configuration Section
3. Page X-6, paragraph 1.3.8.2, conventions for condition-names data-names, file-names, record-names, and report-names
4. Page IV-17, paragraph 4.3.8, uniqueness of reference
5. Page IV-6, paragraph 4.2.2.1.1, user-defined words
6. Page IV-18, paragraph 4.3.8.1, qualification
7. Page X-2, paragraph 1.3.3, global names and local names
8. Page III-10, definition of global names

DATE: August 10, 1990

QUESTION:

There seems to be a conflict between the scoping rules for things declared in the Configuration Section of the top-level program, and the rules for condition-names, data-names, file-names, and record-names.

The rule for Configuration Section items (alphabet-names, class-names, condition-names, mnemonic-names, and symbolic-characters) is that they "may be referenced only by statements and entries either in that program which contains a Configuration Section or in any program contained within that program" (see reference 1).

Also relevant is reference 2: "The entries explicitly or implicitly stated in the Configuration Section of a program which contains other programs apply to each contained program." There would seem to be two possible interpretations of the word "apply".

1. The things declared in the Configuration Section are implicitly given the global attribute, but otherwise obey the "inner-to-outer" name resolution scheme (see reference 3).

2. The things declared in the Configuration Section are implicitly declared locally to all contained programs, so that they are locally available in all of them. This would imply that no other user words by the same name could be defined in any of the contained programs unless they were not used, since any reference to them would be ambiguous (see reference 4). On the other hand, for condition-names, data-names, file-names, record-names, and report-names (which are not declared in the Configuration Section), the rules essentially say that name resolution is from the program of reference outward, and that the reference must be unique within the innermost program containing a matching declaration.

Also relevant is that, according to reference 5, only user-defined words in a given source program, "excluding any contained program", need to belong to the same disjoint set of user-defined words.

This creates an unclear situation when references to names are not context-free and it is not known which name resolution scheme to use. It is also not clear that the context of a reference should be required to determine which type of user-defined word is being referenced.

The following example illustrates the essence of the problem:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. N1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    SYMBOLIC CHARACTERS SC IS 5.
    . . .

        IDENTIFICATION DIVISION.
        PROGRAM-ID. N2.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77 SC GLOBAL PIC X (72).
        . . .
        PROCEDURE DIVISION.
        P2. MOVE SC TO OLINE.
        END PROGRAM N3.
END PROGRAM N2.
END PROGRAM N1.
```

At issue is whether the reference to SC in program N3 should be resolved to the symbolic-character defined in N1 or the data-name defined in N2. The problem is that the inner-to-outer rule does not apply to symbolic-characters, so one cannot say that the declaration in N2 has precedence over that in N1.

Now consider the same example with this version of N3:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. N1.
. . .

    IDENTIFICATION DIVISION.
    PROGRAM-ID. N2.
    . . .
        IDENTIFICATION DIVISION.
        PROGRAM-ID. N3.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77 OLINE PIC X(72).
        . . .
        PROCEDURE DIVISION.
        P2. MOVE ALL SC TO OLINE.
        END PROGRAM N2
    END PROGRAM N2.
```

Everything is the same as before except that the reserved word ALL precedes SC in the reference. The syntax now makes it clear that SC must be a symbolic-character user word, so it would now seem that the definition from N1 could be used, ignoring that in N2 (which is of a data-name). This might seem reasonable.

But now what about this example?

```
IDENTIFICATION DIVISION.
PROGRAM-ID. N1.
. . .
    IDENTIFICATION DIVISION.
    PROGRAM-ID. N2.
    . . .
        IDENTIFICATION DIVISION.
        PROGRAM-ID. N3.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77 OLINE PIC X (72)
        . . .
        PROCEDURE DIVISION.
        P2.  MOVE ALL SC TO OLINE.
        P3.  MOVE SC TO OLINE.
        END PROGRAM N3.
    END PROGRAM N2.
END PROGRAM N2.
```

This is the same as the preceding except that now there are two references to SC, the first of which is clearly to a symbolic-character and the second of which is not clear from the context. On the other hand, there is the rule about the disjoint sets (see reference 5) which would imply that if this program is correct, then both of the references to SC must be to the same disjoint set of user-defined words, hence the second reference must also be to a symbolic-character. This again might seem reasonable.

Now consider this example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. N1
. . .
  IDENTIFICATION DIVISION.
  PROGRAM-ID. N1.
  . . .

      IDENTIFICATION DIVISION.
      PROGRAM-ID. N3
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      77 OLINE PIC X (72).
      . . .
      PROCEDURE DIVISION.
      P2. MOVE SC TO OLINE.
      P3. MOVE ALL SC TO OLINE.
      END PROGRAM N3.
  END PROGRAM N2.
END PROGRAM N2.
```

Now the more ambiguous reference occurs first. If SC is resolved as a data-name, then when it is later discovered that SC should have been a symbolic-character, it will be too late. I think what this example shows is that it is unreasonable that the interpretation of one statement should depend upon the interpretation of another.

The choices seem to be:

1. Always use inner-to-outer scoping, ignoring additional information about the context of the reference. If this then leads to finding a user-defined word of the wrong type, then the program is in error for violating a particular rule in the reference being made (e.g. ALL is not allowed with data-names.)

2. Consider that items declared in the Configuration Section are effectively declared in all contained programs. Then if the user word is also declared another way, the program is in error because the uniqueness of reference rule is violated.

The latter of these two alternatives is the more restrictive, since it does not allow a nested program that does not need the declaration from the Configuration Section to use the same word in a different way locally. This seems to be the intent of the last sentence of the qualification rule (see reference 6) when applied to global data names and non-global data items.

However, the first alternative seems simpler for the user to understand and is more aligned with the general scoping rules for global items.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that alphabet-names, class-names, condition-names, mnemonic-names, and symbolic-characters defined in the Configuration Section are always global names (see references 1, 7, and 8). The rules specified in reference 3 for resolving a name are applied independent of the context in which a name appears and, if the name is not declared with the global attribute in any of the contained programs that contain the program in which the reference appears, then the name may resolve to an alphabet-name, class-name, condition-name, mnemonic-name, or symbolic-character defined in the Configuration Section of the outermost program. Only after the name is resolved can the determination of its appropriateness in the context of the reference be established.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-290

SUBJECT: Record Area

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page III-19, definition of record area
2. Page II-20, paragraph 6.2.2.1.3, record area for files
3. Page VII-30, paragraph 3.8.4, general rule 1, RECORD clause

DATE: May 7, 1990

QUESTION:

The above referenced definitions (see references 1 and 2) for the record area are conflicting.

Reference 1 states: "A storage area allocated ... In the File Section, the current number of character positions in the record area are determined by the explicit or implicit RECORD clause."

However, reference 2 states: "... the storage area is the maximum required to satisfy associated record description entries."

If the RECORD clause indicates a different number of character positions than the associated record description entries, how much storage space is allocated?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that there is a conflict between references 1 and 2: However, it is the intent of ANSI X3.23-1985 that an explicit RECORD clause determine the maximum storage allocation associated with a file. Thus, when the RECORD clause indicates a different number of character positions than the associated record description entry, the size of the storage space allocated is determined by the RECORD clause.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-291

SUBJECT: CLOSE file-name WITH LOCK

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page II-1, paragraph 2, files

2. Page III-9, definition of file

3. Page VII-37, paragraph 4.2.4, general rule 3e, CLOSE statement
   Page VIII-18, paragraph 4.2.4, general rule 2b, CLOSE statement
   Page IX-20, paragraph 4.2.4, general rule 2b, CLOSE statement
   Page XIII-65, paragraph 4.2.4, general rule 3e, CLOSE statement

DATE: July 27, 1989

QUESTION:

Reference 3 states that as a result of executing **CLOSE file-name WITH LOCK**, the "file is locked and cannot be opened again during this execution of this run unit."

Reference 1 states that by convention the term *file* refers to the physical collection of records; while the term *file-name* means the user-defined word used in a COBOL program to reference the file.

If file-name-1 and file-name-2, in the same run unit but perhaps in different programs, refer to the same physical collection of records; and if this run unit successfully executes a **CLOSE file-name-1 WITH LOCK**; must a subsequent attempt to execute **OPEN file-name-2** fail?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that, in the example, the close of file-name-1 would not cause the open of file-name-2 to fail. There is a difference in usage between the definition of the word *file* in the concepts section (see reference 1) and the definition of *file* in the glossary (see reference 2). The definition in the concepts applies "when describing the capabilities of COBOL ...". The definition in the glossary applies to the detailed language specification. Accordingly, within Standard COBOL, the file LOCK option in reference 3 applies to the COBOL file-name, not to the physical collection of records on the external storage medium.

X3J4 DOCUMENT A-292

SUBJECT: PICTURE P and Comparison

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page XVII-56, substantive change number 9
2. Page VI-55, paragraph 6.3.1.1.2, comparison of nonnumeric operands, first paragraph
3. Page VI-31, paragraph 5.9.4, general rule 8, PICTURE clause
4. Page VI-103, paragraph 6.19.4, general rule 4a, MOVE statement

DATE: July 27, 1990

QUESTION:

Paragraph 2 of reference 1 says: "... an item with **PICTURE 9P** and **VALUE 10** will compare equal to an item with **PICTURE XX** and **VALUE "1 "**."

Is this correct?

Reference 2 regarding the comparison of one numeric and one nonnumeric operand says "... the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item ...". In this case, I would expect the move of **PICTURE 9P** and **VALUE 10** to result in an elementary alphanumeric data item with **VALUE 10**. This is the result given in reference 1 for a move of **PIC 9P VALUE 10** to **PIC XX**.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the Standard and the summary of differences are correct. The size of the item with **PICTURE 9P** is 1, and the comparison in paragraph 2 of reference 1 is between "1" with a space appended to it and the value "1 ". This comparison is equal.

While the text of reference 2 says "... the numeric operand is treated as though it were moved to an elementary data item ..." (see reference 4), the rules regarding moves of elementary items with P's in their PICTURE character-string do not apply because this move does not satisfy any of the conditions which explicitly require a numeric sending operand.

X3J4 DOCUMENT A-293

SUBJECT: READ with KEY Phrase

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page I-9, paragraph 1.6, definition of a conforming source program
2. Page III-12, definition of key of reference
3. Page IX-28, paragraph 4.5.3, syntax rule 2, READ statement
4. Page IX-31, paragraph 4.5.4, general rule 15, READ statement
5. Page IX-31, paragraph 4.5.4, general rule 16, READ statement

DATE: August 30, 1989

QUESTION:

Is a READ statement with the KEY phrase disallowed for a file described without the ALTERNATE RECORD KEY clause? Asking the same question in a different way, must an implementor support the use of a READ statement with the KEY phrase for a file that does not have an ALTERNATE RECORD KEY clause? I find no explicit requirement that the ALTERNATE RECORD KEY clause must be specified when the KEY phrase is used, but the requirement may be implicit in the grouping of the features at high level.

Reference 4 says "For an indexed file, if the KEY phrase is specified in a format 2 READ statement, data-name-1 is established as the key of reference ...". Reference 3 requires data-name-1 to be a record key. By reference 2, key of reference can be either a prime record key or an alternate record key. It seems well established that the KEY phrase can be used to specify either a prime key or an alternate key.

Reference 5 provides for reading without the KEY phrase when the prime record key is the key of reference, but this is not the same as saying the KEY phrase *must not* be used unless the file is described with alternate keys.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that a READ statement referencing an indexed file with RANDOM or DYNAMIC access may optionally specify the KEY phrase even though the referenced file is described without the ALTERNATE RECORD KEY clause (see reference 1 and the lack of any explicit prohibition). When there are no alternate record keys, the READ statement KEY phrase, when specified, must reference the name of the prime record key (see reference 3).

X3J4 DOCUMENT A-294

SUBJECT: Size of OCCURS DEPENDING ON Item for CALL BY REFERENCE

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-28, paragraph 5.8.3, general rule 3, OCCURS clause
2. Page X-27, paragraph 5.2.4, general rule 13, CALL statement
3. Page X-25, paragraph 5.1, rule 1, procedure division header
4. Page X-25, paragraph 5.1, rule 2, procedure division header
5. Page X-27, paragraph 5.2.4, general rule 14, CALL statement

COBOL Information Bulletin (CIB) Number 24

6. Interpretation A-27, size of an OCCURS DEPENDING ON item during CALL BY CONTENT

DATE: December 12, 1989

QUESTION:

The size of a group item containing a subordinate item described with OCCURS DEPENDING ON, where data-name-1 of the OCCURS clause is included in the same group, is dependent on whether the group item is referenced as a sending item or a receiving item (see reference 1). If it is a sending item, then reference 1 requires that data-name-1 be set before the group item is referenced.

For a CALL statement, it isn't clear whether an identifier referenced in the USING phrase is a sending or a receiving item. Interpretation A-27 answered the question for CALL BY CONTENT, saying that identifier-2 is a sending item.

Is identifier-2 in the USING phrase of a CALL statement always a sending item?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that identifier-2 in the USING phrase of a CALL statement is not always a sending item. If the parameter specified by identifier-2 is passed with the BY CONTENT phrase, identifier-2 is a sending item (see reference 6). If the parameter specified by identifier-2 is passed with the BY REFERENCE phrase, the object program operates as if the data item in the called program occupied the same storage as the data item in the calling program (see references 2 and 4). The size of this storage area is the maximum size as described by the data description of identifier-2 in the calling program.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-295

SUBJECT: INSPECT LEADING Phrase

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-98, paragraph 6.18.4, general rule 10b, INSPECT statement
2. Page VI-97, paragraph 6.18.4, general rule 6b, INSPECT statement
3. Page VI-97, paragraph 6.18.4, general rule 7a, INSPECT statement

DATE: December 20, 1989

QUESTION:

What is the precise meaning of "eligible to participate" in reference 1?

Consider the following example of the INSPECT statement:

```
01  DATA-STREAM     PIC X(17).
      ...
    INSPECT DATA-STREAM TALLYING COUNT-1
         FOR LEADING  "/"  "A"  "$"  "O"  "C".
```

assuming a value in DATA-STREAM of the following: //AAA$$$OOOCCC

The first comparison cycle results in tallying 2 occurrences of the symbol / (slant). In the second comparison cycle, three occurrences of the letter A might be tallied "... provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate." (see reference 1)

In the first comparison cycle, the letter A was eligible to participate but did not participate because a match was found with the symbol / (slant). The second comparison cycle is not the first cycle in which the letter A is eligible to participate, so the letter A would not be tallied if a strict interpretation is given to the words "eligible to participate". In this case, the increment in the tallying count for the example would be 2.

At least one implementor has interpreted a slightly looser meaning of "eligible to participate" such that both the /'s and A's in the example are tallied, giving a tallying increment of 5. I am inclined to think that this is the intent of the standard; otherwise, for the LEADING phrase, there would be no need for a next comparison cycle after a match is found.

Does the INSPECT statement in the example result in a tallying increment of 2 or 5?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the INSPECT statement in the example results in a tallying increment of 2. Reference 3 states that if neither the BEFORE nor AFTER phrase is specified, "Literal-1 ... is first eligible to participate in matching at the leftmost character position of identifier-1." Thus, in the first comparison cycle, the letter A is eligible to participate but does not because a match is found with the symbol / (see reference 1). Since the second comparison cycle is not the first cycle in which the letter A is eligible to participate, the letter A is not tallied (see reference 1).

X3J4 DOCUMENT A-298

SUBJECT: Simple Insertion Characters as a Part of Floating Strings

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-34, paragraph 5.9.5, editing rule 7, PICTURE clause, paragraph 2
2. Page VI-33, paragraph 5.9.5, editing rule 4, PICTURE clause
3. Page VI-35, paragraph 5.9.5, editing rule 7, PICTURE clause, paragraph 6
4. Page VI-35, paragraph 5.9.5, editing rule 7, PICTURE clause, paragraph 9
5. Page VI-35, paragraph 5.9.5, editing rule 7, PICTURE clause, paragraph 4
6. Page VI-55, paragraph 6.3.1.1.2, comparison of nonnumeric operands

DATE: October 3, 1989

QUESTION:

According to reference 1, simple insertion characters may appear as part of a string of floating insertion strings in a PICTURE clause. Are we correct that the following is conforming COBOL source code?

```
77  A  PIC SV999  VALUE -.123.
77  B  PIC +++0V0+++.
77  C  PIC +++BVB+++.
77  D  PIC +++,V,+++.

PROCEDURE DIVISION.
    MOVE A TO B.
    MOVE A TO C.
    MOVE A TO D.
```

If it is conforming, what value will the receiving fields have after the MOVE statements? (Are the simple insertion characters treated as if they were really coded as the containing floating insertion characters or do they actually maintain their fixed insertion attributes?)

Is it correct that after these moves, a comparison for equality between any two of these fields will fail (unless the answer to the previous question is that all three receiving fields will have the same value)?

If a DISPLAY were done of each of the receiving fields, what would appear?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the PICTURE character-strings on the previous page are conforming according to reference 1. According to reference 3, the floating insertion character will be placed into the character position immediately preceding the decimal point and the preceding character positions will be spaces. The remaining non-zero part of the data is inserted according to the simple insertion editing rule (see reference 2) and the remaining floating insertion editing rules (see reference 5).

Thus, the contents of the items after the MOVE statements are as follows:

B = ␣␣␣-0123
C = ␣␣␣-␣123
D = ␣␣␣-,123

The character ␣ indicates a blank or space character in the contents of the items B, C, and D shown above.

Comparison for equality among any of the fields B, C, and D will fail (see reference 6).

X3J4 DOCUMENT A-299

SUBJECT: Combining Groups

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page IV-14, paragraph 4.3.2, concepts of levels
2. Page V-8, paragraph general format for data division
3. Page III-19, definition of record description entry
4. Page V-13, general format for data description entry
5. Page VI-14, level-numbers

DATE: March 21, 1990

QUESTION:

Reference 1 states that "Groups, in turn, may be combined into groups of two or more groups, etc." This seems to imply that you cannot combine one or more groups into another group. In other words, is the following conforming source code?

```
01  A.
    05  B.
        10  C PIC X.
```

In this example, B is a valid group because "one or more elementary items" may be combined into a group. However, A appears to be an invalid group by the current definition because it is not combining two or more groups.

Is this an error in the standard or was it the intent of third standard COBOL to prohibit this structure?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example COBOL structure is legal and that no changes are needed to the standard.

The description of groups cited by reference 1 is a general description intended to explain the concepts of levels defining record structures, and not an attempt to provide a sentence defining all the possible permutations that the syntax allows. The use of the term "etc." further hints at this.

Reference 5 supports the validity of the example data description.

X3J4 DOCUMENT A-300

SUBJECT: Program-Names as Implementor-Defined or Hardware Dependent

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-54, PROGRAM-ID paragraph
2. Page VII-8, paragraph 2.3.4, general rule 1b, FILE-CONTROL paragraph
3. Pages XVII-87 through 93, implementor-defined language element list
4. Pages XVII-94 through 95, hardware dependent language element list
5. Page X-28, paragraph 5.2.4, general rule 3, CALL statement

DATE: June 11, 1989

QUESTION:

We have been unable to find any statement in either reference 3 or reference 4 which would allow an implementor to place any restrictions on valid program-names beyond those specified for any user-defined word. This would appear to mean that any conforming compiler must be able to support:

1. program names that differ only in the 30th position;
2. program names with character hyphen (—) in any position except the first or the last;
3. program names with leading numerics.

Question 1:

a. Given the fact that most if not all current implementations either place some restrictions on program-names or provide some algorithm for converting valid program-names, was it the intent of the third standard COBOL to include a statement for program-names similar to that in reference 2 for assignment names?

b. Was it also the intent to include program-names in the implementor-defined appendix?

Question 2:

a. If a conforming compiler must provide support for program-names with all possible valid user-defined names, can they provide an algorithm for converting such names to something which meets the rules of their hardware systems?

b. If so, does the statement that a program-name "identifies the object program" mean that there must also be a documented method for associating the object program with the program-name?

Question 3:

a. If an implementation is using an algorithm for converting valid user-defined names to operating system valid program-names, must the implementation insure that the "converted name" can never conflict with a valid program-name?

b. For example, if an implementation's operating system did not support leading numeric program-names, could the implementation provide an algorithm that always converted the last number in a program-name to its corresponding alphabetic character (1 to A, 2 to B, ...) even though the converted name might conflict with a valid program-name for another program?

Question 4:

As some, but not all, implementations make a data item or a file with the EXTERNAL attribute actually an "object" known to the operating system, what, if any, constraints may the implementor place on the user-defined names associated with an EXTERNAL item?

X3J4 RESPONSE:

Question 1a:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that it is not the intent of third Standard COBOL to include a statement for program names similar to that in reference 2 for assignment names.

Question 1b:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that a program-name must conform to the rules for formation of a user-defined word (reference 1). Thus it is not the intent to include program names in the implementor-defined appendix.

Question 2a:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the rules for the resolution at object time of program names and external names are defined by the implementor. Reference 5 states that "the object time resources which must be checked in order to determine the availability of the called program for execution are determined by the implementor." The object time resources include but are not limited to:

1. the resolution of program names;

2. the resolution of external names.

Question 2b:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that it is not within the scope of the Standard to specify how a program-name identifies the object program since this is an environmental issue.

Question 3a:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that since the rules for the resolution of program names at object time are implementor-defined, an implementation that uses an algorithm for converting valid user-defined names to operating system valid program names is not required to insure that there are no conflicts between the "converted name" and the program-name.

Question 3b:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that the example given is valid.

Question 4:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is the same as that for question 2a.

X3J4 DOCUMENT A-301

SUBJECT: Non-Unique User-Defined Words

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page IV-20, Qualification, Rule (1)
2. Page XVII-42, Uniqueness of Reference, Item (8)
3. Page IV-17, Paragraph 4.3.8, Uniqueness of Reference
4. Page IV-18, Paragraph 4.3.8.1, Qualification
5. Page IV-68, Paragraph 6.4.3, The CORRESPONDING Phrase
6. Page VI-92, Paragraph 6.17, The INITIALIZE Statement
7. Page VI-103, Paragraph 6.19.4, General Rule (1)

DATE: August 12, 1991

QUESTION:

Third standard COBOL has introduced the substantive change not affecting existing programs which allows non-unique, incapable of being unique user-defined words which are not referenced. However, we have several questions related to this facility.

Given the following record descriptions:

```
   WORKING-STORAGE SECTION.
(1)    01 X.
(2)       05 B
(3)          10 A      Pic X.
(4)       05 C         Pic X.
(5)       05 A.
(6)          10 A      Pic X.
(7)       05 A         Pic X.
```

```
(8)    01 Z.
(9)       05 B.
(10)      10 A          Pic A.
(11)   05 C.
(12)      10 A.
(13)         15 Y       Pic X.
(14)   05 A.
(15)      10 A          Pic X.
(16)   05 A             Pic X.
```

1. Is this source code conforming COBOL source code (assuming that there are no reference to "A")?

2. Would the statement
   Move Corresponding X to Z
   also be conforming? Does it or does it not "reference" A, therefore, making the entire program non-conforming? If it is a conforming which items, if any would be moved?

3. Would the statement
   Initialize X
   also be conforming? Does it or does it not "reference" A, therefore, making the entire program non-conforming. If it is conforming would the non-unique items be treated like FILLER and not initialized or would they be initialized?

4. Would the statement
   Move Y of A to X
   also be conforming? Does it or does it not "reference" A, therefore, making the entire program non-conforming.?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is that:

1. The example record description is conforming COBOL source code (assuming there are no references to 'A').

2. The statement 'Move Corresponding X to Z', using the example record description, is conforming COBOL source code and there are no references to 'A'. Selected data items within the group item on Line 1 would be moved to the corresponding data items within the group item on Line 8. The transfer of data would be subject to the rules of the CORRESPONDING Phrase. The data name 'A' defined on Lines 5, 7, 14 and 16 cannot be uniquely reference after application of the implied qualifiers. The contents of the data items on Line 3, 4 and 6 would be moved to the data items on Line 10, 11, and 15 respectively.

3. The statement 'Initialize X' is conforming COBOL source. When the INITIALIZE statement is used, the group item is referenced and affected elementary items are initialized in the sequence of their definition within the group. All of the data items contained in 'X' would be initialized to spaces.

4. The statement 'Move Y of A to X' is conforming COBOL source code and there are no references to 'A'.

X3J4 DOCUMENT A-302

SUBJECT: I-O Status Codes 24 and 34

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VII-3, paragraph 1.3.5, I-O status code 34, sequential I-O
2. Page VIII-4, paragraph 1.3.4, I-O status code 24, relative I-O
3. Page IX-4, paragraph 1.3.4, I-O status code 24, indexed I-O

DATE: June 11, 1990

QUESTION:

For I-O status codes 24 and 34 (attempt to write beyond the externally-defined boundaries of the file), the implementor specifies the manner in which these boundaries are defined.

For at least one operating system, the claim is made that files have no externally-defined boundaries. However, the resources available for writing a file are not infinite.

Question 1:

When the resource is exhausted and a write fails, must it be considered an attempt to write beyond the externally-defined boundaries of the file?

Question 2:

Is an implementation standard-conforming in this case if the I-O status for the file is set to an implementor-defined 9x code instead of 24 or 34?

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that when resources fail it need not be considered an attempt to write beyond the externally-defined boundary of the file unless the resource that fails has been defined by the implementor to be an externally-defined boundary.

There are, however, certain types of definitions which must be treated as defining "an external boundary." For example if an implementation provided a method of pre-allocating space for a file that

1. is specific for that file;

2. allocates an explicit amount of storage (either by number of records or by units of storage applicable to that implementation);

3. applies to this running of this program.

Thus, for example, if a system allowed one to pre-allocate a file (or allocate it for this execution of the program) by setting a maximum number of records for the file, then that must be treated as an "external boundary definition." Therefore, a conforming implementation would have to detect an attempt to write more records than specifically allowed for that file and it would have to issue the appropriate 24 or 34 I-O status code.

Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that an implementation may be considered as standard conforming if a file status of 9x is returned for a resource failure that has not been defined by the implementor as an externally defined boundary. Furthermore, a conforming implementation may not have any "externally defined boundaries." In other words, if the implementation only allocates storage for files in general, not specific files, or if it allocates combined program and file storage, these can, but need not, be defined by an implementor as "externally defined boundaries." Therefore, a conforming implementation may not have any way of defining boundaries that are required to be treated as "externally defined boundaries" for a specific file. Such a conforming implementation would never be required to issue either a 24 or 34 I-O status code. (This implementation *could* decide to include its external definitions in a list of "externally defined boundaries" in which case it would be required to issue the appropriate 24 or 34 I-O status code when an attempt was made to write beyond these boundaries.)

X3J4 DOCUMENT A-304

SUBJECT: Function Arguments of Class Alphabetic

REFERENCE:

1. American National Standard COBOL, X3.23a-1989

    a. Page A-28, Section 2.2, Paragraph 2, Subitem (3), Alphanumeric Arguments

    b. Page A-52, Section 2.23.3, The MAX Function, Rule (1)
    Page A-56, Section 2.27.3, The MIN Function, Rule (1)
    Page A-61, Section 2.32.3, The ORD-MAX Function, Rule (1)
    Page A-62, Section 2.33.3, The ORD-MIN Function, Rule (1)

    c. Page A-52, Section 2.23.1, The MAX Functions

    d. Page A-29, Section 2.4, Definition of Functions

2. American National Standard COBOL, X3.23-1985

    a. Page IV-15, Table of Classes and Categories

DATE: November 1, 1991

Question:

There appears to be a contradiction in the intrinsic functions addendum.

In reference 1a, a function argument of alphanumeric type is defined as a data item of class alphabetic or class alphanumeric or a non-numeric literal.

The MAX function and others (reference 1b) state that if more than one argument is specified, all arguments must be of the same class. Reference 2 defines alphanumeric and alphabetic as separate classes.

Reference 1b and 2 say clearly that alphabetic and alphanumeric items must not be mixed in the arguments to MAX, MIN, ORD-MAX, and ORD-MIN.

However, reference 1a indicates an intent that alphabetic items be allowed where arguments of alphanumeric type are allowed. Reference 1c specifies that the arguments to MAX may be type alphanumeric, where type alphanumeric (reference 1a) includes both alphabetic and alphanumeric classes, which is contradictory to reference 1b.

Does the Intrinsic Functions addendum allow the arguments to MAX, MIN, ORD-MAX, and ORD-MIN to be a mixture of class alphabetic and class alphanumeric when there is more than one argument?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is that:

The Intrinsic Functions addendum does allow a mixture of arguments of class alphabetic and class alphanumeric when there is more than one argument.

PROPOSED CORRECTION TO ANSI X3.23A-1989:

X3J4 has proposed a correction to ANSI X3.23a-1989 as a result of the processing of the question in document A-304. The proposed correction appears in Section 4 of this bulletin as changes to A-52, A-56, A-61, and A-62 of ANSI X3.23a-1989.

X3J4 DOCUMENT A-306

SUBJECT: I-O Status 48

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page IX-5, paragraph 1.3.4, 5(g), I-O status 48
2. Pages IX-23 through 25, OPEN statement
3. Page IX-41, paragraph 4.9.4, general rule 3, WRITE statement

DATE: June 11, 1990

QUESTION:

Reference 1 states that an I-O status of 48 should be issued when a WRITE statement is attempted on a file not open in the I-O, output or extend mode. Reference 2 (table 2 for example) states that it is invalid to do a WRITE on an indexed file with sequential access mode that is open in I-O mode. Reference 3 states that a WRITE FROM is equivalent to a MOVE followed by a WRITE.

Question 1:

What I-O status code should be issued when an attempt is made to write a record for an indexed file with access sequential that was opened in I-O mode? (Can an I-O status code of 48 occur for a file that IS opened I-O?)

Question 2:

If no successful OPEN precedes an attempted WRITE FROM statement, must an I-O status of 48 be assigned? (One implementation has stated that as the record is unavailable, the MOVE which is implicit in a WRITE FROM is undefined. Therefore, they abnormally terminate the COBOL program during the WRITE FROM statement and never return to any associated declarative or statement following the WRITE FROM.)

Question 3:

Generally, may an implementation use a standard-defined status code for an undefined situation or must they use a 9x I-O status? For example, if the answer to question 1 is (or were) that there is no defined I-O status for this situation, may an implementation define this as receiving I-O status 48 (as an implementor extension) or must they assign it an I-O status in the 9x range?

X3J4 RESPONSE:

Question 1:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that it is the intent of the Standard that I-O status 48 is issued when a WRITE statement is attempted on a file with sequential access and opened in I-O mode. Reference 2 (table 2) is used in conjunction with the general definition of status code 48.

It is the intention of X3J4 to clarify, in a future standard, the rule that defines when I-O status 48 is issued.

Question 2:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that general rule 2 of reference 2 states that "The successful execution of an OPEN statement makes the record area available ... ." Therefore, if no OPEN statement has occurred, the results of the implied MOVE statement in the WRITE FROM statement are undefined. The implementation may, therefore, terminate the program as the question suggests, or may issue I-O status code 48.

Question 3:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that if there was an error condition that the Standard did not define and the error condition was detected, an implementation may take any or no action. The Standard does not address what happens in an undefined situation.

PROPOSED CORRECTION FOR ANSI COBOL X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-307

SUBJECT: Size Error Condition Without the ON SIZE ERROR Phrase

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-68, paragraph 6.4.2, the ON SIZE ERROR phrase

DATE: May 5, 1990

QUESTION:

Reference 1 states that "If the ON SIZE ERROR phrase is not specified and a size error condition exists after the execution of the arithmetic operation ... control is transferred to the end of the arithmetic statement ... ."

Is it implementor-defined whether or not the arithmetic expression must complete? In other words, one implementation abnormally terminates the entire COBOL program during some (but not all) arithmetic statements if a size error condition occurs but no ON SIZE ERROR phrase is specified. They maintain that the standard specifies what must happen AFTER the execution of the arithmetic statement, should the arithmetic statement be completed, but they also maintain that the standard does not require that the program detect (prevent) an on size error condition unless that phrase is explicitly specified. Therefore, they believe they need not complete the arithmetic statement and therefore need not transfer control to the next (or any) COBOL statement in this situation.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI COBOL X3.23-1985 is that it is nonconforming to abnormally terminate the program when a size error condition occurs and the ON SIZE ERROR phrase is not specified. It is not necessary to complete the evaluation of the arithmetic expression or operation. However, it is required that the "values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored." (See reference 1.) The "end of the arithmetic statement" means the end of the COBOL statement that includes the arithmetic expression or operation.

X3J4 DOCUMENT A-311

SUBJECT: Reserved Word Conflicts with Function-Names

REFERENCE:

1. American National Standard COBOL, ANSI X3.23a-1989
   a. Page A-9, Change to IV-5, Paragraph 4.2.2.1
   b. Page A-18, Section 2.19, The LENGTH Function
   c. Page A-71, Section 2.42, The SUM Function
   d. Page A-75, Section 2.46, The RANDOM Function
2. American National Standard COBOL, ANSI X3.23-1985
   a. Page I-8, Section 1.5.2.3, Reserved Words
   b. Page IV-46, Reserved Words
   c. Page V-18, Format 3 for Report Group Description Entry containing the reserved word SUM
   d. Page V-5, General Format for File Control Entry containing the reserved word RANDOM
   e. Page V-15, General Format for Communication Description Entry containing the reserved word LENGTH
3. Draft Proposed X3.23b-199x, Correction Amendment to ANSI X3.23-1985 and ANSI X3.23a-1989, October 18, 1991 version. Page B-5 and B-6, the second change to IV-5 and change to IV-9.
4. Public Review Document C-102, items 17 and 18.

DATE: March 18, 1992

QUESTION:

Reference 1a says "Within a source program, ... reserved words and function-names form disjoint sets; function-names, system-names, and user-defined words form intersecting sets."

The words SUM, RANDOM, and LENGTH are all both a function-name (Reference 1b) and a reserved word (Reference 2a). The criteria of Reference 1a are not met because:

- Reserved words SUM, RANDOM, and LENGTH are not disjoint from the set of function-names.
- Function-names SUM, RANDOM, and LENGTH do not intersect with the set user-defined words.

The standard is in conflict, and conformance to some part of it must be comprised if the functions module is to be implemented. Which part of the COBOL specification is to be ignored:

- recognition of SUM, RANDOM and LENGTH as reserved words,
- support for the SUM, RANDOM, and LENGTH functions,
- the definition that these function-names and reserved words are disjoint,
- or, for these functions, the definition that function-names and user-defined words intersect?

What is the probable correction of this error?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 and ANSI X3.23a-1989 is that the words LENGTH, RANDOM, and SUM, which are a subset of both function-names and reserved words, are exceptions to the following rules:

- Reserved words and function-names form disjoint sets.
- Function-names, system-names, and user defined words form intersecting sets.

In response to your question concerning a probable correction, see the following section: Draft Proposed X3.23b-199x, Correction Amendment to ANSI X3.23-1985 and ANSI X3.23a-1989, October 18, 1991 version. Page B-5 (A-9) and B-6 (A-10), the second change to IV-5 and the change to IV-9.

PROPOSED CORRECTION TO ANSI X3.23-1985 AND ANSI X3.23A-1989:

X3J4 has proposed a correction to ANSI X3.23-1985 along with ANSI X3.23a-1989 as a result of the processing of the question in document A-311. The proposed correction appears in Section 4 of this bulletin as changes to pages IV-5 (A-9) and IV-9 (A-10) of ANSI X3.23-1985 as updated by ANSI X3.23a-1989.

X3J4 DOCUMENT A-314

SUBJECT: Call By Reference With Occurs Depending On

REFERENCES:

1. American National Standard COBOL X3.23-1985

   (a) Page VI-27 - VI-28, Paragraph 5.8.4, GR(2)b, The OCCURS Clause.
   (b) Page VI-28, Paragraph 5.8.4, GR(3), The OCCURS Clause.

2. Draft COBOL Information Bulletin 26, Interpretation A-294

DATE: August 24, 1990

QUESTION:

During the processing of the Interpretation A-294 concerning the "Size of ODO items in a CALL BY REFERENCE" statement, Reference 1 (a) was mentioned. It states in part:

"At the time the subject of entry is referenced or any data item subordinate or superordinate to the subject of entry is referenced, the value of the data item referenced by data-name-1 must fall within the range integer-1 thru integer-2..."

I thought that during discussion it was felt that this restriction only applied to "sending" items and not to receiving items or items used in a CALL BY REFERENCE statement. Is it a correct interpretation that the value of the data item referenced by data-name-1 is not subject to the above rule when the group which contains the subject of the OCCURS clause is used:

- in a CALL BY REFERENCE statement?

- as a receiving item?

XJ34 RESPONSE:

It is the X3J4 interpretation of X3.23-1985 that the value of the data item referenced by data-name-1 is *not* subject to Reference 1 (a) under the following conditions:

1. when the group being referenced is used in a CALL BY REFERENCE statement, or
2. when the group being referenced is a receiving item AND data-name-1 is inside the group being referenced.

In the above cases the maximum number of occurrences is used to determine the size of the group being referenced.

If the above conditions do not apply, then the data item referenced by data-name-1 is subject to Reference 1 (a) under the following conditions:

1. when the group being referenced is a sending item AND date-name-1 is inside the group, or

2. when data-name-1 is outside the group being referenced (as a sending or receiving item).

In other words, Reference l(b) is applied first to determine if the value of the data item in data-name-1 is needed to determine the part of the table area used. If so, **then,** Reference l(a) applies, which is used solely to determine the number of occurrences. Neither rule applies to CALL BY REFERENCE statements (Reference 2).

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-315

SUBJECT: Reserved Words

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page IV-5, Paragraph 4.2.2.1. COBOL Words
2. Page IV-8, Paragraph 4.2.-2.1.3, Reserved Words
3. Page IV-45, COBOL Reserved words
4. Page IV-45, Paragraph 4.2-2.1.3.1, Required Words Item (2), Special Character Words
5. Page III-21, Reserved Word
6. Page III-4, COBOL Word

DATE: August 12, 1991

QUESTION:

I had previously always thought that "Reserved Words" were a subset of "COBOL Words". However, when I look at pages IV-7 through IV-8 and IV-45, this doesn't seem to be true. Most notably, "+, -, *, **, >, <, =, >=, and <=" are all listed as Reserved words and are even referred to on page IV-9 in 4.2.2.1.3.1(2). However, obviously, these do not meet the requirements for being "COBOL Words" (because of the A-Z and - requirements).

Is there something wrong with either the definition of COBOL words or Reserved words or are these supposed to be intersecting sets rather than Reserved words as a subset of COBOL words?

If these two sets are intersecting rather than one a subset of the other, are System-names or user-defined words also intersecting rather than a subset of COBOL words? If not, why not?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is that reserved words are a subset of COBOL words. There is a problem in the definition of COBOL words (Reference 10). However, it is the intent of third Standard COBOL that each character of a COBOL word, except for special character words, is selected from the set of letters, digits and the hyphen.

PROPOSED CORRECTION TO ANSI X3.23-1985:

X3J4 has proposed a correction to ANSI X3.23-1985 as a result of the processing of the question in document A-315. The proposed correction appears in Section 4 of this bulletin as a change to page IV-5 (2nd change) of ANSI X3.23-1985.

X3J4 DOCUMENT A-316

SUBJECT: Space before Colon in Reference Modification

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page 111-22, 2. Definitions, Definition of Separator
2. Page IV-5, 4.2.1 Separators, Rule 7.
3. Page IV-5, 4.2.1 Separators, Rule S.
4. Page IV-5, 4.2.1 Separators, Rule 9.
5. Pages IV-22 through IV-23, 4.3.8.3 Reference Modification.
6. Page VI-53, 6.2.3 Formation and Evaluation Rules, Arithmetic Operators, Rules 4.
7. Page IV-2, 2.1.6 Format Punctuation.

DATE: August 6, 1990

QUESTION:

I have been trying to find any justification for my belief that a separator space may appear after the "leftmost-character-position" and before the ":" in Reference Modification - as defined on page IV-22. Page IV-5, Rule (9) seems to allow a space after the ":", but not before. Nothing I see in paragraph 2.1.6 starting on page IV-2 seems to support it. Notice that rule (4) on page VI-53 does NOT allow an arithmetic expression to end with a space.

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that for reference modification, spaces may optionally precede the ":" (colon). The ":" (colon) character in the General Format of Reference 5 qualifies as a separator according to Reference 2. Reference 3 states that spaces may optionally precede all separators with three exceptions that do not apply to the separator ":" (colon).

X3J4 DOCUMENT A-319

SUBJECT: Reference Modification of Merge and Sort Key Data Items

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page IV-22, paragraph 4.3.8.3.3, reference modifier
2. Page XI-8, paragraph 4.1.2, general format, MERGE statement
3. Page XI-8, paragraph 4.1.3, syntax rule 4, MERGE statement
4. Page XI-16, paragraph 4.4.2, general format, SORT statement
5. Page XI-16, paragraph 4.4.3, syntax rule 4, SORT statement
6. Page III-7, definition of data-name

DATE: June 11, 1990

QUESTION:

Reference 1 says that, unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of the class alphanumeric is permitted.

In references 2 and 4, merge and sort keys are specified by data-name-1 in the KEY phrase. References 3 and 5 specify that data-name-1 may be qualified, but there is no mention of reference modification.

Does this mean that reference modification may not be specified for the merge and sort key data items, that is, data-name-1, referenced in the KEY phrase of the MERGE and SORT statements?

As a general question, does this mean that reference modification may not be specified wherever a data-name, rather than an identifier, is referenced in a general format unless there is a syntax rule for that format that specifically allows reference modification?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that the merge and sort key data items may not be reference modified. Reference 6 states that a data-name must "not be reference modified ... unless specifically permitted by the rules of the format." The SORT and MERGE statements have no such rules.

X3J4 DOCUMENT A-322

SUBJECT: Floating Insertion Characters in ANSI X3.23-1974

REFERENCES:

American National Standard COBOL, ANSI X3.23-1985

1. Page VI-35, Paragraph 5.9.5 Editing Rules, Rule 7, 7th paragraph, all numeric character positions are floating insertion characters

DRAFT CIB-26

2. Page 57, A-287, PICTURE is P$$

DATE: August 6, 1990

QUESTION:

We currently disagree with our implementor concerning the use of a floating insertion symbol in a PICTURE clause. This PICTURE also uses the implied decimal symbol.

Specifically, I ask if the following combination of floating minus symbols and implied decimal symbol forms a valid ANSI COBOL X3.23-1974 construct: -----V-- (Used in a PICTURE clause)

Several weeks ago, programs using the above editing mask in a PICTURE clause compiled without error. After applying maintenance to our implementor's ANSI '74 COBOL we now get a compiler error. Our implementor contends that they have tightened up adherence to the ANSI COBOL standard." The above construct, they say, is invalid. Even our implementor's COBOL reference manual contains a table which shows that plus or minus floating insertion symbols are valid to the right of the implied decimal point (V)!

Page 133 of Carl Feingold's Fundamentals of Structured COBOL Programming (Third Edition) shows the following as an example of a floating insertion construct within a PICTURE clause:

+(8)V++

Feingold, too, claims to be based on the ANSI COBOL standard.

So, I appeal to the final authority and ask: Is the construct in the second paragraph valid?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that the PICTURE character string ------V-- is a valid construct. (Unlike the PICTURE character string V-- which reference 2 found to be invalid.)

X3J4 is unaware of any difference between ANSI X3.23-1974 and ANSI X3.23-1985 for this syntax.

ANSI X3.23-1985 has superseded ANSI X3.23-1974. This interpretation is based on the ANSI X3.23-1985 COBOL standard. X3J4 no longer provides interpretations of the ANSI X3.23-1974 COBOL standard.

X3J4 DOCUMENT A-323

SUBJECT: NUMVAL-C Currency Sign

REFERENCES:

American National Standard COBOL, X3.23-1985

1. Page III-3, 2. Definitions, COBOL Character Set
2. Page VI-17, 4.5.4 The Special Names Paragraph, General Rule 11.
3. Page VI-55, 6.4.1.1.2 Comparison of Numeric Operands

American National Standard COBOL, X3.23a-1989

4. Page A-59, Section 2.10.3, The NUMVAL-C Function

DATE: March 20, 1992

QUESTION:

1. There is no maximum length specified for cs, the string of one or more characters specified by argument-2. Is the length of cs limited only by an implementor's maximum length for an elementary data item or group item since there is no requirement that argument-2 be an elementary data item? Can a conforming implementation impose a different maximum length for cs?

2. No constraints are specified on the content of argument-2. This leads to ambiguous situations when argument-2 contains some of the characters normally invalid as currency signs. For example, what value is returned for the following:

Function Reference	Returned Value
FUNCTION NUMVAL-C (" - 000123.45 " "-")	-123.45 or 123.45?
FUNCTION NUMVAL-C (" 0 000123.45 " "0")	0 or 123.45, or is it invalid?
FUNCTION NUMVAL-C (" .00123 " ".")	.00123 or 123?

Can a conforming implementation resolve such ambiguities by imposing constraints on the permissible values of argument-2?

3. Are the characters specified in cs case insensitive? In the following examples is "Dm" equivalent to "DM", the "string of one or more characters" specified by argument-2?

FUNCTION NUMVAL-C (" Dm 000123.45 CR ", "DM")

4. Is it correct that the words "the string of one or more characters specified by argument-2" in argument rule 1 refer to one occurrence of a single character string, rather than to one or more repetitions of the characters specified by argument-2? i.e.,

FUNCTION NUMVAL-C (" Lire 100000.50 CR ", "Lire")

rather than

FUNCTION NUMVAL-C (" DMDMDMDM 000123.45 CR ", "DM")

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is that:

Question 1:

The maximum length of cs is limited only by an implementor's maximum lengths. If cs is specified as a nonnumeric literal, then the implementor's maximum length of a nonnumeric literal is the maximum length of cs. If cs is specified as an alphanumeric data item, then the implementor's maximum length of an alphanumeric data item is the maximum length of cs. If a group data item is specified, then the implementor's maximum length of this data item is the maximum length of cs. A conforming implementation cannot impose a different length for cs.

Question 2:

The constraints for the content of argument-2 are limited only by what can occur in an alphanumeric data item and a nonnumeric literal. The three examples cited represent ambiguous situations. For ambiguous situations, the results are undefined. Thus, for the purpose of resolving ambiguities, a conforming implementation may impose constraints on argument-2.

Question 3:

The characters in cs are treated as case sensitive. In the example cited for question 3, "Dm" is not equivalent to "DM" (Reference 4).

Question 4:

The words "the string of one or more characters specified by argument-2" (Reference 4, rule 1) refer to one occurrence of a single character string. The first example for question 4 illustrates the correct specification of cs; the second does not.

PROPOSED CORRECTION TO ANSI X3.23A-1989:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-324

SUBJECT: IF Statement Leveling

REFERENCES:

1. American National Standard COBOL, ANSI X3.23-1985

    a. Page IV-39, paragraph 6.4.2.3.1, Definition of Imperative Statement
    b. Page VI-90, paragraph 6.16.3, syntax rule 1, IF statement
    c. Page I-1, paragraph 1.2, second paragraph describing the Nucleus Module
    d. Page IV-27, paragraph 4.4.4 Explicit and Implicit Scope Terminators, Rule 2.
    e. Page IV-40, paragraph 6.4.3, Scope of Statements
    f. Page I-16, Summary of Elements in the Nucleus Module

2. American National Standard COBOL, ANSI X3.23-1974

    a. Page I-102, paragraph 5.7.2.3.1, Definition of Imperative Statement
    b. Page II-66, paragraph 5.13.3, syntax rule 1, IF statement

DATE: August 2, 1990

QUESTION:

The definition of an imperative statement was changed from the following in ANSI X3.23-1974 (see reference 2a):

> "An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement, nor a compiler directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator."

to the following in X3.23-1985 (see reference 1a):

> "An imperative statement begins with an imperative verb and specifies an unconditional action to be taken by the object program or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator."

Syntax rule 1 of the IF statement was changed from the following in ANSI X3.23-1974 (see reference 2b):

> "(1) Statement-1 and statement-2 represent either an imperative statement or a conditional statement, and either may be followed by a conditional statement."

to the following in X3.23-1985 (see reference 1b):

> "(1) Statement-1 and statement-2 represent either an imperative statement or a conditional statement optionally preceded by an imperative statement. A further description of the rules governing statement-1 and statement-2 is given elsewhere. (See page IV-40, Scope of Statements.)"

With the use of a scope terminator to delimit the scope for an imperative statement, the sequence of statements in example 1 is logically equivalent to the sequence of statements in example 2.

EXAMPLE 1:

```
IF AMOUNT = LOWER-LIMIT
  MOVE AMOUNT TO LL-AMOUNT
ELSE
    IF AMOUNT > LOWER-LIMIT                 >
        MOVE AMOUNT TO OK-AMOUNT            > a conditional statement
    ELSE                                    >
        MOVE AMOUNT TO UNDER-AMOUNT.        >
```

EXAMPLE 2:

```
IF AMOUNT = LOWER-LIMIT
  MOVE AMOUNT TO LL-AMOUNT
ELSE
  IF AMOUNT > LOWER-LIMIT                   >
      MOVE AMOUNT TO OK-AMOUNT              >
  ELSE                                      >
      MOVE AMOUNT TO UNDER-AMOUNT           > an imperative statement
    END-IF.                                 >
```

According to the level 2 boxing in ANSI X3.23-1985, example 1 is a level 2 element while example 2 is a level 1 element.

However the above two examples illustrate that the leveling of the IF statement in ANSI X3.23-1985 (see reference 1b) has no functional purpose in the specifications. Was the presence of the boxing in the IF statement in ANSI X3.23-1985 an editorial error?

X3J4 RESPONSE

The X3J4 interpretation of ANSI X3.23-1985 is that the leveling in the definition of the IF statement and the inclusion of "Imperative and/or conditional statements" as a level 2 only item in the Summary of Elements in the Nucleus Module table (see Reference lf) are correct.

There are many examples where syntax which is valid only in Level 2 of a module can also be coded with a "logically equivalent sequence of statements" from Level 1 of the same module.

In this particular case, when statement-1 of the IF statement is replaced by the conditional form of an IF or SEARCH statement with a NEXT SENTENCE phrase or when statement-2 of the IF statement is replaced by the conditional form of the SEARCH statement including a NEXT SENTENCE phrase, there is no easy way to recode the statements within Level 1 of the NUCLEUS.

However, even in cases such as your example where "logical equivalence" can be obtained using Level 1 syntax, an implementation is only required to support your first example if they claim to support Level 2 of the Nucleus Module while they must support your second example in all defined subsets of the Standard.

X3J4 DOCUMENT A-326

SUBJECT: Implementor Defined Value for CHAR Function

REFERENCE: American National Standard Programming Language COBOL

1. American National Standard COBOL X3.23-1985

   a. Page VI-11, 4.4.3 GR (6), NATIVE collating sequence as default PROGRAM COLLATING SEQUENCE
   b. Page X-5, 1.3.8, Scope of Names

2. American National Standard COBOL X3.23a-1989

   a. Page A-24, Change to Page XVII-93: New Item (83)
   b. Page A-37, 2.9.4, (GR(2) of CHAR function

DATE: June 13, 1991

QUESTION:

1. Why was it made implementor defined as opposed to using the NATIVE collating sequence for evaluating the CHAR function when the current program collating sequence was not specified in the ALPHABET clause?

2. Is there any way that a Standard conforming program compiled with a Standard conforming implementation can have any other current program collating sequence other than NATIVE when the current program collating sequence was not specified in the ALPHABET clause? If so, how?

3. Is an implementation Standard conforming if they do default to the NATIVE collating sequence in evaluating the CHAR function when the current program collating sequence was not specified in the Alphabet clause?

4. If a contained program has a PROGRAM COLLATING SEQUENCE which references an alphabet-name defined in a containing program, is the value returned by the CHAR function implementor defined or defined by the standard? If it is defined by the Standard, what is it?

5.

a. Is this "implementor defined" item required or optional?
b. In other words, is an implementation Standard conforming if it disallows the CHAR function if the current program collating sequence is not specified in the ALPHABET clause?
c. Is an implementation Standard conforming if it leaves undefined (or gives "unpredictable results") when the CHAR function is used in a program for which the current program collating sequence is not specified in the ALPHABET clause?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is that:

Question 1:

The native collating sequence is used if the current program collating sequence was not specified by a PROGRAM COLLATING SEQUENCE clause and an ALPHABET clause. The native collating sequence is implementor defined.

Question 2:

No.

Question 3:

Yes.

Question 4:

It is defined by the Standard, using the collating sequence defined in the ALPHABET clause in the containing program.

Question 5a:

It is required.

Question 5b:

No.

Question 5c:

No.

PROPOSED CORRECTION TO ANSI X3.23A-1989:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-327

SUBJECT: Implementor Defined RANDOM Seed Value

REFERENCE:

1. American National Standard COBOL X3.23-1985

   a. Page A-24, Change to Page XVII-93, New items (84)
   b. Page A-64, Section 2.35.3, Arguments, paragraph 3
   c. Page A-64, Section 2.35.4, Returned Values, paragraph 2

DATE: March 18, 1992

QUESTION:

1. Is this implementor-defined item required or optional? In other words, is an implementation standard conforming if it does not allow the RANDOM function to be used when the first reference to the function does not specify argument-1? Is an implementation standard conforming if it does not define what seed value is used when the first reference to the RANDOM function does not specify argument-1?

2. Is an implementation which defines a value which may vary with every invocation of a program standard conforming? (For example, digits 15 and 16 of the CURRENT-DATE function are used as the seed value.)

3. Is an implementation which defines a value which may vary with every recompilation of a program standard conforming? (For example, digits 15 and 16 of the WHEN-COMPILED function are used as the seed value.)

4. Is an implementation which defines a value which may vary from program to program standard conforming? (For example, it uses the number of characters in the program name as the seed value.)

5. May a standard conforming implementation use different "default seed values" depending on release levels of the implementation? depending on different "standard conforming" compiler options (such as with and without optimization)? Depending on the syntactic environment in which RANDOM function is used?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with X3.23a-1989 is that:

Question 1:

No, in order for an implementation to be standard conforming it must define what seed value is used when the first reference to the RANDOM function does not specify argument-1 (Reference 1b). This definition of the seed value may be either a constant or a technique for determining the seed value.

Questions 2, 3 , and 4:

There is no restriction on how an implementation may vary the implementor-defined seed value. The implementation is standard conforming provided that for a given seed value on a given implementation, the sequence of psuedo-random numbers will always be the same (Reference 1c).

Question 5:

A standard conforming implementation may use different "default seed values" provided that for a given release level, compiler option, syntactic environment, etc., the seed value is defined.

X3J4 DOCUMENT A-334

SUBJECT: Meaning of Integer Value

REFERENCES:

American National Standard COBOL, X3.23-1985

1. Page VI-53, Section 6.2.3, General Rule 6.

American National Standard COBOL, X3.23a-1989

2. Pages A-8 and A-9, Definitions of Integer.

3. Page A-28, Section 2.2, Paragraph (4), Integer.

DATE: March 20, 1992

QUESTION:

In Reference 2, "integer" has 3 definitions. Definitions (1) and (2) exclude digit positions to the right of the decimal point. Definition (3), a numeric function, includes digits to the right of the decimal point provided that all such digits are zero for any possible evaluation of the function.

In Reference 3, an integer is defined as an expression that will always result in an "integer value".

What is the precise meaning of "integer value" in Reference 3? Is an arithmetic expression with zero digits to the right of the decimal point an "integer value"; e.g., where J and K are integer data items, is the expression

```
1.00 * J * K
```

an integer value; thus conforming to the argument type "integer"?

Or, is

```
x - x
```

an integer value, where x is a non-integer data item?

I find nothing that says. This leads me to believe:

1. An expression is an integer value when all its operands meet integer definition (3); and

2. It is implementor-defined by Reference 1 whether other expressions are integer values when the expression always results only in zero digits to the right of the decimal point.

Is this correct?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 along with ANSI X3.23a-1989 is presented in the following paragraphs after a summary of each of the questions.

Question1:

What is the precise meaning of "integer value" in Reference 3?

Answer 1:

The standard does not provide a precise definition of "integer value".

Question 2:

Is an arithmetic expression with zero digits to the right of the decimal point an "integer value"?

Answer 2:

In the case in which the expression is a single element, Yes. An arithmetic expression of a single argument (4) is an "integer value". As soon as the expression contains an operation and multiple operands, Reference 1 applies and the implementor determines if the result is an "integer value".

Question 3:

Where J and K are integer data items, is the expression (1.00 * J * K) an integer value thus conforming to the argument type "integer"?

Answer 3:

Ignoring the possibility of overflow in cases that K and/or J are very large values: the result could be an integer value, depending on the implementation (Reference 1).

Question 4:

Where X is a non-integer value, is the expression (X - X) an integer value?

Answer 4:

Not necessarily. Reference 1 provides that the implementor defines the techniques used in the handling of arithmetic expressions, and therefore which expressions yield an integer result.

Question 5:

Are the following correct?

1. 5A) An expression is an integer value when all its operands meet integer definitions (3); and

2. 5B) It is implementor-defined by Reference 1 whether other expressions are integer values when the expression always results only in zero digits to the right of the decimal point.

Answer 5A) It can be but not always. For 5A, consider the integer arguments in the expression 3/2, which would not yield an integer result.

Answer 5B) Yes. The implementor defines the techniques used in the handling of arithmetic expressions, and therefore which expressions yield an integer result.

PROPOSED CORRECTION TO ANSI X3.23-1985 AND ANSI X3.23A-1989:

X3J4 has proposed a correction to ANSI X3.23-1985 along with ANSI X3.23a-1989 as a result of the processing of the question in document A-334. The proposed correction appears in Section 4 of this bulletin as changes to pages III-12 (A-9) and III-12 (A-8) of ANSI X3.23-1985 as updated by X3.23a-1989.

X3J4 DOCUMENT A-336

SUBJECT: DIVIDE with an Unsigned Quotient

REFERENCES:

American National Standard COBOL X3.23-1985

1. Page VI-82, Divide 6.11.4 GR 6.

DATE: April 3, 1992

QUESTION:

Given the following program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. divtst.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 s-dividend  PIC   S99   VALUE -10
01 s-divisor   PIC   S99   VALUE 3.
01 s-quotient  PIC   S99.
01 quotient    PIC    99.
01 s-remaind   PIC   S99.

PROCEDURE DIVISION.
TEST-DIVIDE.

Div-1 DIVIDE s-dividend BY s-divisor GIVING s-quotient
      REMAINDER s-remain.

Div-2 DIVIDE s-dividend BY s-divisor GIVING quotient
      REMAINDER s-remaind.

   STOP RUN.
```

Consider the two divide statements in the above COBOL program. The only difference between the two is that the quotient in one is signed and in the other unsigned. According to ANSI COBOL the results for Div-1 with the given values in the working-storage section should be -3 for the quotient and -1 for the remainder. When we remove the sign from the quotient, as in Div-2, by following the ANSI COBOL rules the result for the quotient is 3 (the sign removed) and the remainder is -19.

The remainder for Div-2 is calculated in the following way:-

```
Dividend    - (quotient * divisor) = remainder
-10         - (3        *  3     ) =    -19
```

See ANSI X3.23-1985 Nucleus - Divide 6.11.4 General Rule (6).

Could you please let me know if the standard intends that the divide statement should be implemented in such a way that the results shown above are achieved?

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 is that the remainder would be calculated as you have indicated given that the quotient is unsigned. General Rule 6 of the DIVIDE statement, Section 6.11.4, is unambiguous although X3J4 recognizes that the result is mathematically incorrect.

PROPOSED CORRECTION TO ANSI X3.23-1985:

Further clarification for the next full revision is under consideration by Technical Committee X3J4.

X3J4 DOCUMENT A-347

SUBJECT: WITH DEBUGGING MODE clause in the Debug Module

REFERENCES:

FIPS PUB 21-3, COBOL, dated January 12, 1990

CCVS85 Version 2.1, Test Case DB401M

American National Standard COBOL, X3.23-1985

1. Page I-6, Definition of Subsets
2. Pap I-7, Obsolete Language Elements
3. Pap I-8, Extension Language Elements
4. Pap I-40, Summary of Elements By COBOL Division, General Description
5. Pap I-45, Summary of Elements in Environment Division
6. Pap I-58, Summary of Elements in Procedure Division, OPEN statement
7. Page I-59, Summary of Elements in Procedure Division, READ statement
8. Page VI-10, The SOURCE-COMPUTER paragraph, WITH DEBUGGING MODE Clause in the Nucleus Module
9. Page XV-3, The SOURCE-COMPUTER paragraph, WITH DEBUGGING MODE Clause in the Debug module
10. Page XVII-21, Summary of Differences in Environment Division

DATE: July 25, 1991

QUESTION:

Problem:

Test case DB401M of CCVS 85 Version 2.1 expects nonconforming standard flagging for the WITH DEBUGGING MODE clause when flagging Level 1 features of the Debug module. However, the WITH DEBUGGING MODE clause is a conforming standard element in every subset of Standard COBOL and in FIPS 21-3 COBOL, and must never be flagged as nonconforming standard.

The WITH DEBUGGING MODE clause is an element in both the Nucleus module and the Debug module (References 8 and 9). There is no mechanism in Standard COBOL or in FIPS 21-3 for distinguishing a WITH DEBUGGING MODE clause in the Nucleus module from a WITH DEBUGGING MODE clause in the Debug module; in fact, one WITH DEBUGGING MODE clause in a given program enables the debugging features of either or both modules. There are no rules in Standard COBOL or FIPS 21-3 to restrict the clause to either module regardless of which other debugging features appear or do not appear in the program.

Questions for FIPS interpretation and X3J4 interpretation:

1. X3.23-1985 specifies the WITH DEBUGGING MODE clause in both the Nucleus module and the Debug module. Is there any rule in Standard COBOL that classifies the WITH DEBUGGING MODE into one module or the other, based on the presence or absence of any other related debugging feature, or based on anything else?

2. If there is such a rule, is it syntactically distinguishable?

3. Is the WITH DEBUGGING MODE clause always a conforming language element in any subset of Standard COBOL?

Test case DB401M is attached for reference.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.
000300     DB401M.
000400*THE FOLLOWING PROGRAM TESTS THE FLAGGING OF LEVEL 1
000500*FEATURES OF THE DEBUGGING MODULE.
000600  ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SOURCE-COMPUTER.
000900     IBM-370
000910     WITH DEBUGGING MODE.
000950*Message expected for above statement: NON-CONFORMING STANDARD
001000 OBJECT-COMPUTER.
001100         IBM-370.
001200 INPUT-OUTPUT   SECTION.
001300         FILE-CONTROL.
001400         SELECT TFIL ASSIGN
001500         SYS001-DA-3330-S-FIL01
001600         ORGANIZATION IS SEQUENTIAL
001700 ACCESS MODE IS SEQUENTIAL.
001800 DATA DIVISION.
001900 FILE SECTION.
002000 FD  TFIL.
002100 01  FREC.
002200         03 RKEY  PIC  9(8).
002300
```

```
003000  PROCEDURE DIVISION.
003100
003200  DECLARATIVES.
003300
004100
004200  BUGGIM-2 SECTION.
004300
004400
004500  USE FOR DEBUGGING ON ALL PROCEDURES.
004600*Message expected for above statement: NON-CONFORMING STANDARD 004800
004900
005000 BUGGING-3 SECTION.
005100
005200
005400     USE FOR DEBUGGING ON DB301M-CONTROL.
005500*Message expected for above statement: NON-CONFORMING STANDARD
005600
005700
005800  END DECLARATIVES.
005900
006000  DB30IM-FLAGS   SECTION.
006100          DB301M-CONTROL.
006200  DISPLAY 'THIS IS A DUMMY PROCEDURE'.
006300          STOP RUN.
006400
006500
006600*TOTAL NUMBER OF FLAGS EXPECTED = 3.
       *END-OF,DB401M
```

X3J4 RESPONSE:

The X3J4 interpretation of ANSI X3.23-1985 together with ANSI X3.23a-1989 is that:

1. As stated in your question, X3.23-1985 specifies that the DEBUGGING MODE clause is a part of both the NUCLEUS and the DEBUGGING modules. (See References 5, 8, and 9.) In general, when the same element is listed with two or more modules in the Summary of Elements (by Division), then it is classified as a part of both modules regardless of context. When similar elements can be distinguished as to which module they belong, multiple entries appear in the Summary of Elements.

2. There is no such rule (that classifies the WITH DEBUGGING MODE into one module or the other based on the presence or absence of any other related debugging feature or based on anything else).

3. Support for the WITH DEBUGGING MODE clause must be provided in the minimum subset of third Standard COBOL and all other subsets whether or not any optional modules are included. (See Reference 1). Therefore, the WITH DEBUGGING MODE clause is always a conforming language element in all subsets of Standard COBOL.

X3J4 DOCUMENT A-348

SUBJECT: Non-Contiguous Segments with Same Segment Number

REFERENCES:

1. FIPS PUB 21-3, COBOL, dated January 12,1990

2. CCVS85 Version 2.1, Testcase SG401M

3. American National Standard COBOL X3.23-1985

   a. Page XVI-7, Segmentation, paragraph 3.2.4, General Rule I
   b. Page 1-1, Section 1.1, Scope and Purpose
   c. Page 1-9, Section 1.6, Definition of a Conforming Source Program

4. COBOL Interpretation Bulletin (CIB) 24, Interpretation A-11, Extension Language Elements and Conforming Implementations

DATE: June 23, 1991

QUESTION:

Problem:

Testcase SG401M of CCVS 85 Version 2.1 expects nonconforming standard flagging for a noncontiguous segment with a repeated segment number, when flagging level 2 segmentation features. However, the associated rule in Standard COBOL (Reference 3a) is a general rule, not a syntax rule. As a member of the CODASYL COBOL Committee and Technical Committee X3J4, I am aware that both committees practice a convention of specifying syntax rules for requirements that are meant to be distinguished at compile time; the requirements specified as general rules are not meant to be distinguished at compile time, even if it is sometimes possible to do so. This is not stated in X3.23-1985, probably because Standard COBOL does not require flagging of nonconforming syntax other than nonstandard extensions and obsolete language elements.

Reference 4, interpretation document A-11 in CIB 24, responds as follows to a number of questions regarding compile time versus runtime analysis:

> "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."

While X3.23-1985 requires flagging of only nonstandard extensions and not standard extensions, the response in A-11 may provide a direction for consistent interpretation of nonconforming standard flagging requirements in FIPS 21-3. Since X3J4 did not give a definition of "syntactically distinguishable" additional clarification is needed.

Questions for X3J4 interpretation:

1. Is it a convention for Standard COBOL That rules stated as syntax rules syntax checked at compile time, even though Standard COBOL makes no requirement for flagging other than nonstandard extensions and obsolete language elements?

2. Is it a convention for Standard COBOL that rules stated as general rules are not meant to be syntax checked at compile time, even though in some cases it might be possible to do so?

3. In Standard COBOL, must General Rule 1, Page XVI-7, stating "all sections which have the same segment-number must be together in the source program" be syntax checked at compile time?

X3J4 RESPONSE:

The X3J4 interpretation of X3.23-1985 and X3.23a-1989 is that:

1. In general, the General Formats and Syntax Rules in the body of the standard specify the form (or syntax) of programs expressed in COBOL. The standard makes no requirement for syntax checking (Reference 3b), other than for purposes of flagging nonstandard extensions and obsolete language elements; but if an implementation were to syntax check, this checking would, in general, be based on rules dealing with the form of COBOL.

2. In general, the General Rules in the body of the standard specify the interpretation of programs expressed in COBOL. The standard makes no requirement for syntax checking (Reference 3b), other than for purposes of flagging nonstandard extensions and obsolete language elements; but if an implementation were to syntax check, this checking would, in general, not be based on rules dealing with the interpretation of COBOL.

3. In Standard COBOL, syntax checking is not required for General Rule 1, Page XVI-7, or for any other General Rule or Syntax Rule (Reference 3b).

The expression "in general" is used in the answers to questions 1 and 2 because it is unclear whether some of the rules in Standard COBOL relate to the form (syntax) or the interpretation (semantics) of COBOL, and some of the rules are currently misplaced.

Segmentation General Rule 1 may have been mistakenly specified as a general rule rather than a syntax rule, but X3J4 plans no effort to investigate further because the segmentation module is obsolete.

SECTION 3: INDEX OF INTERPRETATIONS

3.1 INTRODUCTION

This section of the COBOL Information Bulletin contains a cross reference index to the interpretations made by the X3J4 COBOL Technical Committee relative to ANSI COBOL X3.23-1985. This index is arranged in order by subject matter within ANSI X3.23-1985 and ANSI X3.23a-1989.

3.2 DEFINITION OF AN IMPLEMENTATION OF STANDARD COBOL

3.3 NUCLEUS MODULE

Language Concepts

Identification Division

Environment Division

Data Division in Nucleus Module

Procedure Division in Nucleus Module

*The interpretation in A-22 has been superseded (reversed) by the interpretation in A-225.

Debugging in Nucleus Module

3.4 SEQUENTIAL I-O MODULE

Language Concepts

Environment Division in Sequential I-O Module

**The interpretation in A-44 has been clarified by the interpretation in A-286.

Data Division in Sequential I-O Module

Procedure Division in the Sequential I-O Module

3.5 RELATIVE I-O MODULE

Language Concepts

Environment Division

Data Division

Procedure Division in the Relative I-O Module

3.6 INDEXED I-O MODULE

Language Concepts

Environment Division

Data Division

Procedure Division in the Indexed I-O Module

3.7 INTER-PROGRAM COMMUNICATION MODULE

Language Concepts

Data Division

Procedure Division

3.8 SORT-MERGE MODULE

Environment Division

Procedure Division

3.9 SOURCE TEXT MANIPULATION MODULE

3.10 REPORT WRITER MODULE

Procedure Division

3.11 COMMUNICATION MODULE

Data Division

Procedure Division in the Communication Module

3.12 SEGMENTATION

3.13 INTRINSIC FUNCTION MODULE

Language Concepts

Arguments

Functions

SECTION 4: DRAFT PROPOSED CORRECTION AMENDMENT

4.1 INTRODUCTION

This document contains a copy of the draft proposed correction amendment to ANSI X3.23-1985 and ANSI X3.23a-1989 as of the September 1993 meeting of X3J4. This draft proposed correction amendment is subject to further modification resulting from future meetings of the X3J4 COBOL Technical Committee.

4.2 DISCLAIMER

It should be noted that the content of this draft proposed correction amendment to ANSI X3.23-1985 and ANSI X3.23a-1989 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 amendment is offered as an information 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 X3.23-1985 or ANSI X3.23a-1989.