CIB 28

ISO/IEC JTC 1/SC22/WG4 **N 094**

Record of Response

Issue 3

ISO/IEC JTC 1/SC22/WG4 Defect Reports 062 through 081

Programming languages — COBOL

ISO/IEC 1989:1985 (ANSI X3.23-1985)
ISO/IEC 1989:1985/Amd.1:1992 (ANSI X3.23a-1989)
ISO/IEC 1989:1985/Amd.2:1994 (ANSI X3.23b-1993)

September 1996

Introduction

This document contains responses to defects reported concerning ISO/IEC 1989:1985, ISO/IEC 1989:1985/Amd.1:1992, and ISO/IEC 1989:1985/Amd.1:1994. The responses are interpretations of Programming Language COBOL prepared by Accredited Standards Committee X3J4 and approved by SC22/WG4. These interpretations, reflecting the technical opinion of the experts maintaining Programming Language COBOL, are informative and have no normative effect on Standard COBOL.

Related normative changes, if any, either have been included in ISO/IEC 1989:1985/Amd.2:1994 or are planned for a future revision of the standard.

The interpretations in this Record of Response will be published nearly simultaneously by X3 in COBOL Information Bulletin 28 (CIB-28). To maintain correspondence between CIB-28 and this Record of Response, each interpretation carries dual identification in the style "DR-nnn / A-nnn" where

DR-nnn is the WG4 Defect Response identifier, which corresponds to the number of the related Defect Report, and

A-nnn is the X3 identifier.

Terminology and references

Second Standard COBOL ISO 1989:1978, Programming languages — COBOL (ANSI X3.23-1974, Programming Language — COBOL)

Third Standard COBOL ISO/IEC 1989:1985, Programming languages — COBOL (ANSI X3.23-1985, Programming Language — COBOL)

Amendment 1 ISO/IEC 1989:1985/Amd.1:1992, Programming languages — COBOL, Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language — Intrinsic Function Module for COBOL)

Amendment 2 ISO/IEC 1989:1985/Amd.2:1994, Programming languages — COBOL Amendment 2: Correction and clarification amendment for COBOL (ANSI X3.23b-1993, Programming Language - Correction Amendment for COBOL).

Defect Report Summary

Defect Report and Response Number	X3 CIB-28 Answer Number	Subject
DR-062	A-296	STRING Statement
DR-063	A-297	Qualification in RENAMES Clause
DR-064	A-305	READ INTO Statement and Elementary Records
DR-065	A-325	Implementor Defined Function Representation and Characteristics
DR-066	A-332	Size of signed numeric when moved to group
DR-067	A-335	Execution of EVALUATE
DR-068	A-337	INSPECT statement and reference modification
DR-069	A-342	When the content of a data item is referenced
DR-070	A-346	Subscripted Datanames in WHEN phrase of SEARCH ALL
DR-071	A-350	Zero-Length Group Items as Conditional Variables
DR-072	A-351	Blank When Zero, Category Numeric Edited, and S in Picture
DR-073	A-353	Arithmetic Statement - Extent of Implementor Latitude
DR-074	A-354	File Status Field Restrictions
DR-075	A-355	CALL in a Multi-Language Case Sensitive Environment
DR-076	A-358	Figurative Constant and CODE Clause
DR-077	A-360	Data in Different Length Records
DR-078	A-362	Overlapping Operands in MOVEs
DR-079	A-363	Incompatible Data
DR-080	A-364	Exponent Overflow
DR-081	A-365	Overlapping Arithmetic

Interpretation A-296

SUBJECT: STRING Statement

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-131. 6.25, The STRING Statement
2. Page VI-132, 6.25.4, The STRING Statement, General Rule 4b

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

In the STRING statement identifier-2 (DELIMITED BY) may be signed since the definition of integer does not preclude signed identifiers. There are no rules concerning the comparison, however. UNSTRING cannot be used as a model since it allows only alphanumeric identifiers. INSPECT specifies that signed identifiers are used as if the sign had been removed. What is the result of using a signed identifier in the DELIMITED BY clause of a STRING statement?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

The comparison of the characters in identifier-1 with those in identifier-2 is performed on a character-by-character basis (Reference 2). A sign participates in the comparison as a separate character or as a character composed of a digit and sign, depending on the presence or absence of the SEPARATE phrase in the sign clause in the data description entry of identifier-2.

Interpretation A-297

SUBJECT: Qualification in RENAMES Clause

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-40, Section 5.11.3, SR 5
2. Page VI-40, Section 5.11.3, SR 4
3. Page VI-40, Section 5.11.3, SR 2
4. Page VI-38, Section 5.11.3, SR 7
5. Page VI-18, Section 4.3.8.1, Qualification

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

According to Reference 1, data-name-2 and data-name-3 in the RENAMES clause may be qualified. Reference 2 requires that data-name-2 and data-name-3 be names of items within the logical record with which the RENAMES clause is associated, and Reference 3 restricts the placements of data-name-2 and data-name-3 in the source program. Considering only the specification of RENAMES, it appears that data-name-2 and data-name-3 require qualification only when that data-name is non-unique within the logical record with which the RENAMES clause is associated.

The REDEFINES clause is similar to RENAMES in that no ambiguity of reference exists for data-name-2 and data-name-3 because of the required placement of the REDEFINES clause within the source program. The uniqueness of reference for REDEFINES is recognized in Reference 4 and further documented under the rules of qualification (Reference 5). However, there is no mention of similar uniqueness for RENAMES.

Does the standard require data-name-2 and data-name-3 in the RENAMES clause to be qualified to uniqueness within the context of the entire data division, or does it require uniqueness only within the context of the logical record with which the RENAMES clause is associated?

RESPONSE:

The standard requires data-name-2 and data-name-3 in the RENAMES clause to be qualified to uniqueness within the context of the entire Data Division.

Interpretation A-305

SUBJECT: READ INTO Statement with Elementary Records

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page IV-15, 4.3.3, Concept of Classes of Data
2. Page VII-45, Sequential I-O, General Rule 7b
3. Page XVII-65 and 66, Substantive Changes (Potentially Affecting), (23) READ Statement
4. Page I-9, 1.6, Definition of a Conforming Source Program

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

According to Reference 2, the second type of RECORD DESCRIPTION that can occur subordinate to a file used in a READ INTO statement is when all record descriptions for a file "describe a group item or an elementary alphanumeric item." However, it does not state whether these items must be of CLASS alphanumeric or CATEGORY alphanumeric. Reference 1 states that CLASS alphanumeric includes the Numeric edited and the Alphanumeric edited categories as well as the alphanumeric category.

Reference 3 would seem to indicate that the intention of Third Standard COBOL was to prevent situations where any conversions would be needed by a READ INTO statement.

Question 1: Does Reference 2 mean class or category alphanumeric?

Question 2: Based on the answer to question 1, is the following conforming or nonconforming source code?

```
FD  File-Name.
01  Rec-1          Pic X(10).
01  Rec-2          Pic Z(09)9.
01  Rec-3          Pic XB X(08).
        ...
    Read File-Name into Working-Storage-Item
```

Question 3: Based on the difference between reference 2 and what appears to be the intent of reference 3, is it true that the following is non-conforming source code, even though no conversion would be needed?

```
FD  File-Name.
01  Rec-1          Pic X(10).
01  Rec-2          Pic A(10).
    ...
    Read File-Name into Working-Storage-Item
```

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. There is an omission in the standard. Reference 2 refers to category alphanumeric.
2. The example in question 2 is nonconforming source code because some of the elementary items are not category alphanumeric.
3. The example in question 3 is nonconforming source code because Rec-2 is category alphabetic rather than category alphanumeric.

CHANGES NEEDED IN STANDARD:

A modification to a future standard may be indicated as a result of this interpretation.
This issue will be pursued for a future standard.

Interpretation A-325

SUBJECT: Implementor Defined Function Representation and Characteristics

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page XVII-87, Introduction to Implementor-Defined Language Element List
2. Page VI-20, 5.3, The Data Description Entry
3. Page IV-16, 4.3.4, Selection of Character Representation and Radix

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL):

4. Page A-24, Change to page XVII-93, New items 81 and 82
5. Page A-10, Second occurrence of a change to page IV-16
6. Page A-29, 2.3, Items (2) and (3)

QUESTION:

It is unclear what exactly is meant by the new "implementor defined" items 81 and 82 documented on page A-24 of the Intrinsic Function Addendum.

1. What is meant by "characteristics of the returned value"? Must this indicate the position of the decimal point and/or total number of digits? What other "characteristics" must be defined instead or in addition?
2. Must the implementor document or otherwise make visible to the users the USAGE, position of the decimal point, and total number of digits available for every possible combination or set of arguments for every FUNCTION for which these are not specified in the Addendum?
3. Must the implementor insure that these implementor-defined items are identical for every compiler option, every order of arguments, every release, and in every allowable syntactic environment? If not, must they insure that all possible characteristics/representations are documented or otherwise explicitly "implementor defined"?
4.
   a. Was the intent of this Addendum to indicate that the Standard places no constraints on the USAGE and "characteristics" of the returned values?

b. If so, why were these placed in the "implementor defined" list as opposed to the "explicitly undefined" list? (or more simply, what does it mean that these items are "implementor defined"?)

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. If the returned value is a COBOL data type, then the characteristics are those defined by the following clauses of a data description entry: PICTURE, USAGE, SIGN, and SYNCHRONIZED (see Reference 2). The total number of digits and the position of the decimal point are included.

   There is no requirement that the returned value be a COBOL data type. For this case, the standard does not define what is meant by the "characteristics of the returned value".

2. No.

3. No to both questions.

4.

   a. Yes.

   b. The Implementor-Defined Element List includes items that the implementor must define and make visible as well as items that the implementor must define and need not make visible. Item 81 is an example of the former while item 82 is an example of the latter. Neither of these fits into the category of "undefined".

CHANGES NEEDED:

The committee intends to clarify in a future standard which implementor-defined items must be documented by the implementor and which need not.

Interpretation A-332

SUBJECT: Size of signed numeric when moved to group

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-55, Section 6.3.1.1.2, Comparison of Nonnumeric Operands
2. Page VI-55, Section 6.3.1.1.2, Comparison of Nonnumeric Operands, Rule 2
3. Page VI-105, Section 6.19.4, The MOVE statement, general rule 5
4. Page VI-30, Section 5.9.3, The PICTURE clause, general rule 7
5. Page VI-30, Section 5.9.3, The PICTURE clause, general rule 3
6. Page VI-31, Section 5.9.3, The PICTURE clause, general rule 8, Picture Symbol 'S'
7. Page VI-42, Section 5.12.4, The SIGN Clause, general rule 4

QUESTION:

1. When comparing a signed numeric item without a SIGN IS SEPARATE clause to a group item, is the sign moved to the group item described in Reference 2?

Reference 2 says the numeric item is moved to a group item of the "same size as the numeric data item (in terms of standard data format characters)".

2. Reference 6 says the "S" is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN IS SEPARATE clause. Does this mean that an overpunched sign must be removed from the signed numeric item before moving it to a group item of the same size as the numeric data item (in terms of standard data format characters)?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) is the following:

1. Yes. The sign is moved to the group item. According to Reference 3 there is no conversion of data for this move.

2. No. Assuming the sign is embedded in the data, the internal representation of the sign is moved to the group item without conversion (Reference 3).

Interpretation A-335

Subject: Execution of EVALUATE

References:

ISO 1989:1985, Programming Languages - COBOL (ANSI X3.23-1985, Programming Language - COBOL)

1. Pages VI-85 and VI-86, Section 6.13.4, General Rule 1
2. Pages VI-86, Sections 6.13.4, General Rule 2

ISO 1989:1985/Amd. 1:1992, Programming Languages - COBOL Amendment 1: Intrinsic function module (ANSI X3a.23-1989, Programming Language - Intrinsic Function Module for COBOL)

3. Page A-65, The RANDOM Function

Question:

Reference 1 specifies that execution of the EVALUATE statement operates as if each selection subject and selection object were evaluated and assigned a value. Reference 2 specifies that execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared ...

Does this mean that the evaluation of every selection subject and selection object is to be executed before the first comparison is executed? Or, are the words "as if" used simply for ease of understanding the operations, without implying actual evaluation of all selection subjects and selection objects before the first comparison? My guess is that the latter is true, because of the inefficiency of evaluating every selection subject and selection object regardless of whether a WHEN phrase needing their values is ever reached.

In general, it seems the effects are the same either way, except when the RANDOM function is specified as an operand. Since each evaluation of RANDOM affects the next random number, the order of evaluation of selection subjects and selection objects (and whether some get evaluated or not) has an effect on the program. Does the specification for the EVALUATE statement require that each occurence of the RANDOM function as a selection subject or selection object be evaluated whether or not its associated WHEN phrase is ever reached? If this is required, then are there any other elements of the language having similar effect?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd. 1:1992 (ANSI X3a.23-1989) is the following:

No, there is no requirement in Standard COBOL that all selection subjects and all selection objects in the EVALUATE statement be evaluated before the first comparison.

No, the specification of the EVALUATE statement does not require that the RANDOM function be evaluated before required for the comparison of the selection subject with the selection object.

CHANGES NEEDED IN STANDARD:

The first two sentences of reference 1 and the word "then" in reference 2 are misleading. This problem will be added to the list of Current Standard Bugs.

Interpretation A-337

SUBJECT: INSPECT statement and reference modification

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Pages IV-22, 4.3.8.3.3., Syntax Rule 3, Reference-modifier
2. Page VI-95, 6.18.3 Syntax Rule 7, INSPECT statement
3. Page VI-95, 6.18.3 Syntax Rule 8, INSPECT statement
4. Page VI-95, 6.18.3 Syntax Rule 9, INSPECT statement

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

The rules for the INSPECT statement do not prohibit reference modification for any of the operands of the INSPECT statement. Thus, reference modification is allowed for all alphanumeric operands (Reference 1). References 2-4 are syntax rules about the sizes of certain such alphanumeric operands. Consider the four examples below:

```
INSPECT ID-1 REPLACING ALL ID-4(OFFSET-1:LENGTH-1)
        BY ID-5(OFFSET-2: LENGTH-2)
INSPECT ID-1 REPLACING ALL ID-4(OFFSET-1:LENGTH-1)
        BY QUOTES
INSPECT ID-1 REPLACING CHARACTERS
        BY ID-5(OFFSET-2: LENGTH-2)
INSPECT ID-1 CONVERTING ID-6 (OFFSET-1: LENGTH-1)
        BY ID-5(OFFSET-2:LENGTH-2)
```

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

The standard allows reference modification for all operands of the INSPECT statement.

CHANGES NEEDED IN STANDARD:

A future standard needs to provide rules for insuring that reference modified items can be used and under what conditions - whether such constraints can be detected only at runtime, or at either runtime or compile-time.

Interpretation A-342

SUBJECT: When the content of a data item is referenced

REFERENCES:

ISO 1989:1985, Programming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL):

1. Page IV-25, 4.4.2, Explicit and Implicit Transfers of Control
2. Page VI-28, 5.8.4, The OCCURS Clause, General Rule (3)
3. Page VI-70, 6.4.7, Incompatible Data

ISO 1989:1985/Amd.1:1992, Programming Languages – COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language – Intrinsic Function Module for COBOL):

4. Page A-18, Change to page VI-70, Incompatible Data

This Record of Response (COBOL Information Bulletin)

5. Interpretation A-363, Incompatible Data

QUESTION:

1. What exactly does amended third Standard COBOL mean by the phrase (in the definition of incompatible data)?

   "when the content of a data item is referenced"

   (Many of the following specific questions try to clarify this general question.)

2. Is it true that any time that the current syntax or general rules describe the use of an identifier (or data item) as or as if it were a "sending item" as referenced in the general rules of the OCCURS clause, that the same use is also always "a reference to the content of a data item"?

3. Are there cases (such as in a conditional statement) when the use of a data item does not qualify it as a "sending item" but does qualify as a "reference to the content of the data item"? If so what are they or how are they distinguished from syntactic uses of a data item which do not reference its content (such as a receiving item in a MOVE, an INITIALIZE, or Format 4 SET)?

4. Are uses within the USING phrase of either a PROCEDURE DIVISION header or a CALL statement considered "references to the content of a data item in the Procedure Division"? Does it matter in a CALL statement if the data item is preceded by either BY CONTENT or BY REFERENCE (explicitly or implicitly) and if so how?

5. Is identifier-1 in any or all formats of the INSPECT statement considered a "reference to the content of a data item"? If so, is it for all formats with any optional phrases or is it only for specific formats or with specific phrases?

6. In order for a Procedure Division statement to "reference the content of a data item" must that data item (or identifier) actually explicitly appear in the Procedure Division statement? For example, does a GENERATE statement "reference the content of a data item" if the specified detail line includes either a SUM or SOURCE statement referencing a data item whose current content does not match its PICTURE specification?

7. If the answer to the previous question is that the data item need not be explicitly mentioned in the Procedure Division statement, what are the rules for when implicit references do and do not "reference the content of a data item"? For example, how does one distinguish between a data item used to store a file status versus a data item used in a RECORD VARYING IN SIZE DEPENDING ON clause when doing a WRITE? (I would assume the former could contain incompatible data before the WRITE statement while the latter could not.)
8. Finally, NIST has indicated that for FIPS 21-3 validation test DB201A, when line 28300 does a MOVE statement where the sending items is defined as numeric but during the execution of that statement contains spaces, that an implementation must continue execution past that statement with the following additional option:

   "A given implementation may, however, issue a message warning of the possible incompatible data condition."

   Amended third Standard COBOL states that when a reference is made to a data item whose content is not compatible with the class specified for that data item, then the "result of such a reference is undefined." Is it true that the fact that this result is "undefined" means that an ANSI conforming implementation need not even continue execution after such a reference or must the implementation insure continued execution (but has an option to issue a warning message as specified by NIST)?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. The standard does not define exactly what is meant by the phrase "when the content of a data item is referenced". This is an omission in the standard. While some simple questions might be answered on the basis of intuition, it would be necessary to develop rules in order to answer these broad questions. Development of rules to correct this omission is planned for a future standard.
2. The response is the same as the response to question 1.
3. The response is the same as the response to question 1.
4. The response is the same as the response to question 1.
5. The response is the same as the response to question 1.
6. The response is the same as the response to question 1.
7. The response is the same as the response to question 1.
8. Yes, it is true that a conforming implementation need not continue when incompatible data is detected.

CHANGES NEEDED:

The committee intends to precisely specify in a future standard the circumstances under which the condition of "incompatible data" is applicable and "when the content of a data item is referenced".

Interpretation A-346

SUBJECT: Subscripted Datanames in WHEN phrase of SEARCH ALL

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-123, The SEARCH statement, Section 6.22.3, Syntax Rule 4
2. Page IV-21, Subscripting, Section 4.3.8.2.2, General Format
3. Page IV-21, Subscripting, Section 4.3.8.2.3, Syntax Rule 4
4. Page IV-21, Subscripting, Section 4.3.8.2.3, Syntax Rules
5. Page IV-124, The SEARCH statement, Section 6.22.4, Syntax Rule 4

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

1. Reference 1 specifies that data-name-1 and data-name-2 in the WHEN phrase of a SEARCH statement containing the ALL phrase must be subscripted by the first index-name associated with identifier-1. Does this exclude relative subscripting (+ or - integer-3) as defined in Reference 2?

2. Reference 1 goes on to say "and other subscripts as required". Is it the case that these "other subscripts" must also be specified as index-name, thus excluding relative indexing? If not, then what are the semantics of relative indexing in these "other subscripts"?

3. Shouldn't there be a syntax rule in Reference 4 similar to Reference 3, stating something like:

"Each table element reference must be subscripted only by index-name when such reference appears in the WHEN phrase of a SEARCH statement containing the ALL phrase."?

Response:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO

1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. Yes, reference 1 does exclude relative subscripting.
2. No, the "other subscripts" may be any form of subscript - normal semantics of relative indexing apply.
3. No, such a rule would be in conflict with this interpretation of the standard.

Changes Needed To Standard

Clarification is needed in syntax rule 4 of the SEARCH statement. This issue will be pursued for a future standard.

Interpretation A-350

SUBJECT: Zero-Length Group Items as Conditional Variables

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page I-9, 1.6, Definition of a Conforming Source Program.
2. Page I-9, 1.6, Definition of a Conforming Source Program, third paragraph.
3. Page VI-27, 5.8.4, GR (2), The OCCURS clause.
4. Page VI-50, 5.15.6, Data Description Entries Other Than Condition-Names, Rule (5), The VALUE clause.
5. Page VI-58, 6.3.1.3, Condition-Name Condition (Conditional Variable).
6. Page VI-127, 6.23, The SET Statement.
7. Page VI-54, 6.3.1.1.2, Comparison of Nonnumeric Operands.

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

ISO 1989:1985/DAM2, Amendment 2: Correction and clarification amendment for COBOL (draft ANSI X3.23b-199x, Correction and clarification amendment for COBOL):

8. Page B-6, change to page IV-11.

SC22 N 1604, Record of Response Issue 1, Programming Languages - COBOL (X3 COBOL Information Bulletin 26):

9. Page 35, A-90, Zero-Length Groups.

QUESTION:

Given the following (partial) sample source code:

```
Working-Storage Section
01  Group-Item.
    05  Num-Field          Pic 9.
    05  Group-ODO.
     88  ODO-Condition     Value All "1".
        10  ODO            occurs 0 to 9 times
                           Depending on Num-Field.
            15  Elem       Pic X.
Procedure Division.
    ...
Length-1.
    Move 1 to Num-Field
    Set ODO-Condition to True
    Evaluate True
        When ODO-Condition
            If ODO-Condition
                Display "IF and Evaluate - Length=1"
            Else
                Display "Evaluate but not IF - Length=1"
            End-IF
        When Other
            If ODO-Condition
                Display "IF but not Evaluate - Length=1"
            Else
                Display "Neither IF nor Evaluate - Length=1"
            End-IF
    End-Evaluate
    If Group-ODO = All "1"
        Display "Figurative Constant Compare True - Length=1"
    Else
        Display "Figurative Constant Compare False - Length=1"
    End-If.
Length-0.
    Move 0 to Num-Field
    Set ODO-Condition to True
    Evaluate True
        When ODO-Condition
            If ODO-Condition
                Display "IF and Evaluate - Length=0"
            Else
                Display "Evaluate but not IF - Length=0"
            End-IF
        When Other
            If ODO-Condition
                Display "IF but not Evaluate - Length=0"
            Else
                Display "Neither IF nor Evaluate - Length=0"
            End-IF
    End-Evaluate
    If Group-ODO = All "1"
        Display "Figurative Constant Compare True - Length=0"
    Else
        Display "Figurative Constant Compare False - Length=0"
    End-If.
```

Please answer the following questions:

1. Assuming the remainder of the program is valid, is there anything in this portion of source code which would make it non-conforming? If so, what and why? Would this be detectable at compile-time, or is this conforming source code with undefined run-time behavior, or is this non-conforming source code which cannot be detected at compile-time?
2. Is it true that the rules of amended third Standard COBOL require that when executed, that the LENGTH-1 paragraph will display the following two literals:

   ```
   "IF and Evaluate - Length=1"
   "Figurative Constant Compare True - Length=1"
   ```

   If these two literals will NOT be displayed, please explain what is displayed and why.
3. I believe that the current interpretation represented in Reference 9 requires that the second display in the LENGTH-0 paragraph would be:

   ```
   "Figurative Constant Compare False - Length=0"
   ```

   However, I am unable to tell which literal would be displayed first in this paragraph. Please:

   a. State if the first literal to be displayed in the LENGTH-0 paragraph is defined in amended third Standard COBOL.

   b. Specify, if it is defined, which literal would be displayed.

   c. Explain the logic behind the specification of which literal is to be displayed (if defined in amended third Standard COBOL).

   d. Explain how amended third Standard COBOL can require a condition-name to be evaluated as FALSE immediately after that condition-name is set to TRUE.

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) and ISO 1989:1985/Amd.2:199x (ANSI X3.23b-198x[1] is the following:

Question 1: Assuming the rest of the program is valid, the illustrated portion of code is conforming source code. Standard COBOL does not specify the semantics of a SET statement when the associated conditional variable is a zero-length group item (Reference 6), therefore this is conforming source code with run-time behavior that is undefined (Reference 2).

Question 2: Yes (Reference 7). the 'Length-1' paragraph in the sample code would display the two literals:

```
"IF and Evaluate - Length=1"
"Figurative Constant Compare True - Length=1"
```

---

[1] **This answer assumes approval of ISO 1989:1985/DAM2, Amendment 2: Correction and clarification amendment for COBOL, and draft ANSI X3.23b-199x, Correction and clarification amendment for COBOL.**

Question 3:

a. Yes.

b. "Neither IF nor Evaluate - Length=0" would be displayed.

c. ODO-Condition is a zero-length group item. In accordance with the rules for a relation test (Reference 7), comparison proceeds as though ODO-Condition were extended on the right by sufficient spaces to give ODO-Condition a length of 1, the length of the ALL literal (Reference 9). Therefore, the result of the comparison is false.

d. Regardless of the statements executed previously, ODO-Condition is a zero-length group item at the time of its evaluation. Evaluation proceeds in accordance with the rules for a relation test, as described in the response to question 3c.

CHANGES NEEDED IN STANDARD:

Correction of the standard is under investigation. There is a need to define the semantics of a SET statement when the conditional variable is a variable-length group item.

Interpretation A-351

Subject: Blank When Zero, Category Numeric Edited, and S in Picture

References:

ISO 1989:1985, Programming Languages - COBOL (ANSI X3.23-1985, Programming Language - COBOL)

1. Page VI-22,BLANK WHEN ZERO Clause, Syntax Rule #1
2. Page VI-22,BLANK WHEN ZERO Clause, General Rule #2
3. Page VI-30,PICTURE Clause, General Rule #6

ISO 1989:1985/Amd. 1:1992, Programming Languages - COBOL Amendment 1: Intrinsic function module (ANSI X3a.23-1989, Programming Language - Intrinsic Function Module for COBOL)

Question:

Reference #1 states that the BLANK WHEN ZERO clause can be specified only for an elementary item whose PICTURE is specified as numeric or numeric edited; Reference #2 states that an item with the BLANK WHEN ZERO clause is considered as category numeric edited; and Reference #3 lists the valid characters and rules for a PICTURE clause of an item to be defined as category numeric edited.

1. Is the following a valid data item definition for an item to be considered numeric edited?

```
05 Num-Edited   Pic 9V9  Blank When Zero.
```

NOTE: Sub-item b in Reference #3 requires that at least one of a specified list(none of which are in this PICTURE clause) must appear in the PICTURE of a numeric edited item. Therefore, does this mean that the above definition is invalid or that it is not for a numeric edited item?

2. Is the following a valid data item definition for an item and if so what category should it be considered?

```
05 Num-Edited-2   Pic S9V9  Blank When Zero.
```

NOTE: Sub-item b in Reference #3 lists the valid characters for the PICTURE clause of a numeric edited item and this list excludes "S" but includes "V".

3. If neither of the previous definitions are valid, are there any valid data item definitions which would be category numeric without the BLANK WHEN ZERO clause but which are considered numeric edited with this clause? If there are none, why does Reference #1 say "numeric OR numeric edited"? If there are some, please provide an example that does not violate the rules of Reference #3.

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd. 1:1992 (ANSI X3a.23-1989) is the following:

1. Yes, this picture is a valid numeric PICTURE string; adding the BLANK WHEN ZERO clause makes this item a valid numeric edited data item.

2. No, the specification for the data item is not valid. "S" is not in the list of permissible picture symbols for a numeric-edited data item (reference 3) and therefore a BLANK WHEN ZERO clause is not permissible when "S" has been included in the PICTURE.

3. The first definition is valid.

CHANGES NEEDED IN STANDARD:

On Page VI-22, General Rule 2, delete "considered to be".

Interpretation A-353

SUBJECT: Arithmetic Statement - Extent of Implementor Latitude

REFERENCES:

ISO 1989:1985, P ɔgramming Languages – COBOL (ANSI X3.23-1985, Programming Language – COBOL):

1. Page VI-53, Paragraph 6.2.3, Rule 6, Formation and Evaluation Rules (of Arithmetic Expressions).
2. Page VI-69, Paragraph 6.4.6, Multiple Results in Arithmetic Statements.
3. Page VI-76, Paragraph 6.8, The Compute Statement.
4. Page VI-76, Paragraph 6.8.4, The Compute Statement, General Rule 1.
5. Page VI-76, Paragraph 6.8.4, The Compute Statement, General Rule 2, Developing the arithmetic expression with multiple receiving items.
6. Page VI-52, Paragraph 6.2.3, Rule 1, Formation and Evaluation Rules (of Arithmetic Expressions).

ISO 1989:1985/Amd.1:1992, Programming Languages – COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language – Intrinsic Function Module for COBOL):

7. Page A-69, The SQRT Function, Returned Values (1), Approximation of Square Root.

QUESTION:

Amended third Standard COBOL discusses the use of a temporary data item when an arithmetic statement includes multiple receiving items. It also states,

"Each implementor will indicate the techniques used in handling arithmetic expressions."

However, it seems unclear about the extent of implementor latitude in defining the techniques for resolving arithmetic statements (as opposed to arithmetic expressions) and the handling of multiple receiving items in the same COMPUTE statement.

It seems to me that users expect and many implementors have produced COBOL compilers which

- Always evaluate the exact same arithmetic expressions (same identifiers and same values in the same order) as the same value - regardless of context and receiving items (if any).
- Always evaluate arithmetic expressions as a "good approximation" of the algebraic value represented by the expression.

However, I have difficulty knowing if this is what Amended third Standard COBOL intends much less requires.

I have the following questions:

1. Although there is no statement for general arithmetic expressions like that for some intrinsic functions which requires the "evaluated arithmetic expression" to be treated as an "approximation" of the algebraic evaluation of the expression, is this the intent of amended third Standard COBOL? (Compare Reference 7).

2. Does Reference 1 place any constraints on the implementor handling of arithmetic expressions? For example, are all of the following valid implementor arithmetic expression handling rules? (If any are invalid, please explain why.)

   a. Arithmetic expressions beginning on an even line of source code are always evaluated as zero while those beginning on odd lines are evaluated as -1.

   b. All arithmetic expressions are evaluated as +1 when the program is run on Saturday or Sunday but are evaluated as -1 when run on other days of the week.

   c. Each subsequent arithmetic expression which is executed in a program's logic is evaluated as an integer one greater than the preceding arithmetic expression with the first expression being evaluated as the number of source lines in the program.

3. Reference 1 states that each implementor defines the techniques for handling arithmetic expressions. However, there is no such statement for arithmetic statements. Does this mean that implementors may *provide any* defined technique for handling arithmetic expressions, but when an arithmetic expression is included in an arithmetic statement (the COMPUTE verb), that the "implementor-defined" techniques must take into consideration *only* those factors which are a part of the expression? In other words, does amended third Standard COBOL require that the same arithmetic expression must be evaluated as the same value independent of the statement(s) in which it occurs (Assuming that all the operands in the arithmetic expression itself - but not the arithmetic statement - are the same and have the same value)?

4. Does the *answer* to the *preceding* statement along with the paragraph on "Multiple Results in Arithmetic Statements" on page VI-69, mean that the value of Integer-1 is or is not guaranteed to be the same after each of the three compute statements in the following example:

```
01  Integer-1      Pic 9.
01  Integer-2      Pic 9.
01  Decimal-1      Pic 9V9.
    ...
Compute-1.
    Move Zero to Integer-1
    Compute Integer-1        = (7 / 4) * 4.
Compute-2.
    Move Zero to Integer-1
                 Integer-2
    Compute Integer-1
            Integer-2 Rounded = (7 / 4) * 4.
Compute-3.
    Move Zero to Integer-1
                 Decimal-1
    Compute Integer-1
            Decimal-1        = (7 / 4) * 4.
```

5. If two but not three of the above examples should give the same answer, please explain which and why (and why the other is not). If none of them are required to give the same answers please explain what justification there is for only stating that "arithmetic expressions" are implementor-defined but not stating this for "arithmetic statements" (or at least the COMPUTE verb).

   Note: General Rule 3 of the COMPUTE verb on page VI-76 does reference the fact that the techniques used in handling the arithmetic expression are implementor-defined, but says nothing about the arithmetic statement itself. Furthermore, General Rule 4 of the COMPUTE verb explicitly references "Multiple Results in Arithmetic Statements" and that paragraph on page VI-69 does not seem to allow any arithmetic statement (rather than expression) to use the characteristics of one or more receiving items in changing the value in other receiving items.

6. If the answers to the previous questions indicated that there are no restrictions on the latitude allowed implementors when evaluating arithmetic expressions, please comment on the following example code:

   Move Field-1 (1:) to Field-2

   a. Is it true that the implementor may evaluate the "1" (which must be an arithmetic expression as the "leftmost-character-position" in reference modification) as *any* integer value and may, therefore, create a temporary data item from any character position in Field-1 through the end of field-1?

   b. Is it true, therefore, that there is no way in which reference modification may be specified in order to create "portable" applications (even when both the starting position and the length are specified as integer literals or identifiers)?

c. Is it true, therefore, that if an implementation defines its technique for evaluating the starting position arithmetic expression as always 1 greater than the length of the data item being referenced and thereby it always violates rule 4a on page IV-23, that such an implementation could always issue a syntax error against any reference modification and still be ANSI conforming (for all defined subsets of the Standard)?

RESPONSE

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. No, there were mathematical reasons for specifying for some of the intrinsic functions that the value returned is an approximation.
2. Yes, Reference 1 places the following constraints on the implementor's handling of arithmetic expressions:

   a. that the value of arithmetic expressions be derived by application of the arithmetic operations to the operands denoted in the expression, in accordance with operator precedence rules in Reference 6, and

   b. that the techniques used in handling arithmetic expressions be indicated by the implementor.

   The techniques to be indicated by the implementor include factors such as radix conversions and precision of intermediate results, which may affect the accuracy of the resulting value.

   The examples you give do not meet the first constraint and, therefore, are not valid examples of expression handling.
3. Reference 1 places no requirements or limits on the factors that an implementor may consider in specifying the "implementor-defined" techniques used in handling arithmetic expressions. Reference 1 makes no requirement that the same arithmetic expression be evaluated as the same value, whether executing the same statement or another statement. Reference 5, for multiple receiving items, requires that an arithmetic expression be *first* evaluated and *then* stored as the new value of *each* of the resultant identifiers; i.e., the arithmetic expression is to be evaluated once.
4. The value of integer-1 is not guaranteed to be the same after each of the COMPUTE statements in the example.
5. We are unable to answer this question. We do not have the information necessary to justify the decisions of past language designers.
6. No, it is false in all cases.

   It was the intent of Standard COBOL that, for reference modification, an arithmetic expression consisting of a single identifier or a single literal result in a value equal to the single literal or the value of the data item referenced by the single identifier. The COMPUTE statement (Reference 4) has such a rule, and X3J4 believes there was an unintentional omission of this rule for arithmetic expressions used in places other than in the COMPUTE statement.

CHANGES NEEDED

A change to correct the problem associated with question 6 is under investigation for a future standard. The recommended change is to delete General Rule 1 under the COMPUTE statement and to add a similar rule under arithmetic expressions addressing an arithmetic expression consisting of a single identifier or a single literal.

Interpretation A-354

Subject: File Status Field Restrictions

References:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VII-10, The File Status Clause
2. Page VII-12, Padding Character Clause, General Rule 6
3. Page VII-29, The Linage Clause, General Rule 11
4. Page VIII-9, File Control Entry, General Rule 1f
5. Page X-2, External Objects and Internal Objects
6. Page X-23, The External Clause

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic Function Module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL):

CODASYL COBOL Journal of Development 1988

7. Page III-8.4-6, Paragraph 8.4.8, I-O Status

Question

1. If a physical file is associated with an External File Connector, there are explicit rules indicating that all but one data-name or identifier which can be specified in a file control entry (Select/Assign Clause) if present, must reference External data items. For example, an identifier in a Padding Character clause (Reference 2); the relative key data item (Reference 4); and the same external Linage identifiers (Reference 3). However, I can find no rules requiring that the data item referenced in the File Status clause for an External file also needs to be external. Is there any such restriction and if so where is it documented? If not, why must all the other data-names or identifiers (which can be specified in a File Control Entry) be External if the file is External, but the file status data item need not be?

   Note: It is possible that this was an historical accident/error due to the fact that the File Status clause was still a part of the FD (not in a Select/Assign statement) in the JOD when the ANS'85 Standard was under development. Therefore, it would already have been under an External clause (for an External File) if left in the FD, but when it was moved to the File-Control Paragraph, it appears that this "restriction" was lost or forgotten. If you look at the JOD today, it still states (Reference 7) that the file status data item is associated with the file connector which could only happen if External file connectors required External file status code data items.

2. One implementation has indicated that (for some file organizations, but not all) that the data item referenced in the File Status clause may not be defined in the Linkage Section (or at least not in the Linkage Section if the actual referenced storage need not be the same from invocation to invocation of the subprogram doing I-O). This seems to conflict with Syntax

Rule 2 on page VII-10 which only prohibits File Section, Report Section, and Communication Section items for reference in a File Status clause.

a. Are there any rules in amended third Standard COBOL which would allow the implementor to make such a restriction either as a default or as an "extension"?

b. If so, what are they?

c. If there is no such restriction in general, does it matter if the Linkage Section item referenced in the File Status clause was actually passed by Content as opposed to by Reference?

d. If amended third Standard COBOL requires a conforming implementation to allow a file status clause to reference a Linkage Section data item, is it true that each update of this file status must be in the currently referenced storage rather than in the storage referenced by the Linkage Section item at the time of the OPEN statement?

Response

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) along with ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. There is no requirement that data-name-1 in the FILE STATUS clause be described with the external attribute for a file associated with an external file connector. The other data-names describe fixed file attributes which need to be the same for each I/O operation. The file status data item is always a receiving field so it need not be the same for all I/O operations.

2.

a. No, this is not allowed.

b. N/A

c. No, it may be passed either by Content or by Reference.

d. Yes, the storage referenced when the I/O statement is executed must be used.

Changes Needed in Standard

Delete "always" and change "whenever" to "when" in VII-10 2.5.4 General Rule 1, The File Status Clause.

Interpretation A-355

SUBJECT: CALL in a Multi-Language Case Sensitive Environment

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page X-27, CALL Statement, General Rule 1
2. Page X-28, CALL Statement, General Rule 3
3. Page X-28, CALL Statement, General Rule 4
4. Page IV-5, paragraph 4.2.2.1, COBOL words
5. Page IV-6, paragraph 4.2.2.1.1, user-defined rules
6. Page VI-7, paragraph 3.3, PROGRAM-ID paragraph
7. Page X-5, 1.3.8.1, Conventions for Program-names
8. Page I-9, 1.6, Definition of a Conforming Source

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

There have already been several interpretation requests concerning uppercase and lowercase equivalence in CALL statements, but it seems that there is still some uncertainty as to what is implementor-defined and what is not.

1. If an application is compiled and run in an operating system which does distinguish between object programs stored with uppercase characters in their names from those with lowercase characters, and there is a COBOL program stored with a name of "AA" and a non-COBOL program stored with a name of "aa"

a. If the following CALL statement occurs in a COBOL program, is it defined whether the COBOL or non-COBOL program is invoked?

```
Call "aa"
```

If it is not implementor defined, please explain why. (If the Standard defines which is invoked, please state which it is and why.)

b. If the following data item and CALL statement occur in a COBOL program, is it defined whether the COBOL or non-COBOL program is invoked?

```
05  Pgm-Name Pic X(02) Value "aa".
 ...
Call Pgm-Name
```

If it is not implementor defined, please explain why. (If the Standard defines which is invoked, please state which it is and why.)

2. If an application is compiled and run in an operating system which does distinguish between object programs stored with uppercase characters in their names from those with lowercase characters, and there is a COBOL program stored with a name of "AA" and another COBOL program stored with a name of "aa"

a. If the following CALL statement occurs in a COBOL program, is it defined which COBOL program is invoked (or is this a non-conforming application)?

```
Call "aa"
```

If it is not implementor defined, please explain why. (If the Standard defines which is invoked, please state which it is and why.)

b. If the following data item and CALL statement occur in a COBOL program, is it defined which COBOL program is invoked (or is this a non-conforming application)?

```
05  Pgm-Name Pic X(02) Value "aa".
 ...
Call Pgm-Name
```

If it is not implementor defined, please explain why. (If the Standard defines which is invoked, please state which it is and why.)

3. For either of the preceding questions, would the answer be different if the literal or identifier had a value of "Aa" or "AA"? If so, why?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. It is implementor-defined (Reference 1).

2. COBOL program-ids "AA" and "aa" are the same program-name. Combining two or more separately-compiled COBOL programs with the same program-name into a single run unit violates the rules of Standard COBOL (Reference 7); the result is undefined (Reference 8). This is a non-conforming application.

3. No.

CHANGES NEEDED:

Enhancement to address case-sensitive program-names in interlanguage calls are being considered.

Interpretation A-358

SUBJECT: Figurative Constant and CODE Clause

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Pages IV-10 to IV-12, Figurative Constant Values
2. Page XIII-14, Syntax Rule 1, The CODE Clause
3. Page XIII-14, General Rule 1 and 2, The CODE Clause

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

Page IV-12 states that a figurative constant can be used whenever "literal" appears in a format and the format for the CODE Clause does use "literal".

The second rule (2) on page IV-11 states,

> "When a figurative constant ... is not associated with another data item ..., the length of the string is one character."

Similarly, the first rule (3) on page IV-12 states,

> "When the figurative constant ALL literal is not associated with another data item, the length of the string is the length of the literal."

However, syntax rule (1) of the CODE Clause on page XIII-14 states,

> "Literal-1 must be a two-character nonnumeric literal."

1. Is it conforming or nonconforming source code to use a figurative constant composed of the word ALL and a two-character nonnumeric literal in the CODE Clause?
2. Is it conforming or nonconforming source code to use a figurative constant composed of the word ALL and a one-character nonnumeric literal in the CODE Clause? If it is conforming, what is the result of this reference?
3. Is it conforming or nonconforming source code to use a figurative constant composed of the word ALL and a three-character nonnumeric literal in the CODE clause? If it is conforming, what is the result of this reference?

4. Is it conforming or nonconforming source code to use a figurative constant without the word ALL (and other than Zero) in the CODE clause? If it is conforming, what is the result of this reference?
5. Is it conforming or conconforming source code to use the figurative constant Zero in the CODE clause? If it is conforming, what is the result of this reference?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

Question 1: It is conforming. The CODE clause implies the association of a two-character data item (Reference 3) with any figurative constant specified in the CODE Clause.

Question 2: It is conforming. The result is the concatenation of 2 occurrences of the nonnumeric literal.

Question 3: It is conforming. The result is the first 2 characters of the nonnumeric literal.

Question 4: It is conforming. The result is 2 characters composed of the concatenation of 2 occurences of the figurative constant.

Question 5: It is conforming. The result is "00".

Interpretation A-360

SUBJECT: Data in Different Length Records

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-21, The Data Description Entry, General Rule 3
2. Page VII-30, The RECORD Clause
3. Page VII-52, The WRITE Statement

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

A file defined with format 1 of the RECORD clause produces fixed length records. It appears to be valid to include multiple record definitions of different length for such fixed format files. When multiple 01 levels appear under the same FD, they are implicit redefinitions of the same storage. However, it is unclear what happens to the entire record area associated with a file when data is moved to a record-description which is smaller than the entire record-area.

Given the program

```
identification division.
program-id.  fixed.
environment division.
input-output section.
file-control.
select f1 assign fred.
data division.
file section.
fd f1 record contains 10.
01   r1 pic x(10).
01   r2 pic x(5).
working-storage section.
01   x1 pic x(5).
procedure division.
 mainline.
    open output f1
    move all "X" to r1
    move all "Y" to x1
    write r2 from x1
    close f1
    open input f1
    read f1
    close f1
    stop run.
```

Is this conforming source code for a conforming application? If so, what are the contents of record area r1 following the READ statement and why?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

The example is conforming source code. The content of record area r1 following the READ statement is "YYYYY" followed by 5 character positions with undefined content. The content of character positions beyond the length of record area r2 are not defined for the WRITE statement.

CHANGES NEEDED IN STANDARD:

A correction to the Standard is under investigation. A rule is needed for the WRITE statement indicating that the content of the record area beyond the length of the record written is undefined.

Interpretation A-362

Subject: Overlapping Operands in MOVEs

References:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page IV-23, paragraph 4.3.8.3.4, Reference Modification
2. Page VI-69, paragraph 6.4.5, Overlapping Operands
3. Page VI-93, paragraph 6.17.3, General Rule (7), The INITIALIZE Statement
4. Page VI-100, paragraph 6.18.3, General Rule (17), The INSPECT Statement
5. Page VI-103, paragraph 6.19, The MOVE Statement
6. Page VI-133, paragraph 6.25.4, General Rule (12), The STRING Statement
7. Page VI-140, paragraph 6.27.4, General Rule (21), The UNSTRING Statement
8. Page III-7, Glossary Definition of "Data Description Entry"
9. Page VI-20, DATA DESCRIPTION ENTRY
10. Page XII-12, PAGE-COUNTER Rules

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

ISO 1989:1978, Programming Languages -- COBOL (ANSI X3.23-1974, Programming Language -- COBOL):

11. Page II-51, paragraph 5.3.5, Overlapping Operands

CODASYL COBOL Committee Journal of Development (JOD), 1988, with updates to November 1990:

12. Page III-8.26-1, paragraph 8.26.4, The MOVE Statement

SC22 N 1517, US Contribution on COBOL: COBOL Information Bulletin Number 24

13. Page 37, A-23, Overlapping Parameters in CALL statement

Question:

1. "Does Third Standard COBOL permit moves of overlapping operands?"

2. The sending and receiving items in the following MOVE statement share a part of their storage areas and are defined by the same data description entry. Is the MOVE statement defined by amended Third Standard COBOL?

```
01   X      PIC X(3).
01   LEN    PIC S9 VALUE 2.

MOVE X(1:LEN) TO X(2:2).
```

3. If the answer to Question 2 is yes, is the MOVE statement in Question 2 equivalent to:

```
MOVE X(1:LEN) TO TEMP.
MOVE TEMP     TO X(2:2).
```

4. If the answer to Question 2 is no, is the MOVE statement in Question 2 defined if LEN has a value of 1?

Introduction:

It is clear that the wording in the paragraphs on "Overlapping Operands" in second Standard COBOL and amended third Standard COBOL are different. The purpose of this interpretation request is to determine:

- If this wording difference also causes a "currently undocumented" substantive change potentially affecting existing programs.
- If those situations that this paragraph claims are defined are actually (well) defined.
- If the rules for Overlapping Moves and the MOVE statement are actually complete and non-contradictory or if they are conflicting or incomplete rules that make some situations "implicitly undefined".

Note: X3J4 has previously discussed (in interpretation responses and within committee meetings) the "as if" rules that currently exist within amended third Standard COBOL. It is not the intent of this request to require additional discussion of this topic. Within this document, when I use this term, I mean that given ***complex-statement-1*** is defined as working "as if" it were ***compound-statements-2***, then this means:

a. ***complex-statement-1*** is considered to be logically (semantically) equivalent to ***compound-statements-2*** and that the COBOL Standard considers that portability would be enhanced by defining the required semantics in terms of ***compound-statements-2***.

b. This definition makes no requirement on the implementation technique(s) for ***complex-statement-1***. (The "object code" for ***complex-statement-1*** and ***compound-statements-2*** may or may not be identical.)

c. If two separate implementations give the same run-time results (semantics) for ***compound-statements-2***, then they must also yield the same results for ***complex-statement-1***.

d. If two separate implementations give different run-time results (semantics) for ***compound-statements-2***, then they must also yield different results for ***complex-statement-1***. However, each individual implementation must yield the same results for ***complex-statement-1*** and ***compound-statements-2***.

If the previous rules are not a correct interpretation of X3J4's view of amended third Standard COBOL, the response to this request need not (but may) explain this; however, the following questions should all be read as meaning the above when they use the concept of statement-1 working "as if" it were statement-2.

Question:

5. Is it true (assuming "ELEM" is an elementary data-item) that the results of executing the following COBOL code are explicitly undefined in second Standard COBOL (because of its rules for Overlapping Operands) but are explicitly defined in amended third Standard COBOL (because

the sending and receiving items are "defined in the same data description" as referenced in its rules for Overlapping Operands)?

```
MOVE ELEM TO ELEM.
```

Or is there an explicit General Rule for the MOVE statement as referenced in the last sentence of paragraph 6.4.5 on page VI-69 of X3.23-1985 that makes this undefined in amended third Standard COBOL (and if so, what is it)?

6. If the answer to the previous question is "yes", isn't this a "Substantive Change Potentially Affecting Existing Programs" that should have been included in Appendix B but wasn't?

7. If the answer to the first question was yes, then is it true that the intent of General Rule 2 on page VI-102, is that

```
MOVE ELEM TO ELEM.
```

should work "as if" it were coded

```
MOVE ELEM TO temp.
MOVE temp TO ELEM.
```

where temp is "an intermediate result item provided by the implementor" that has all the same explicit and implicit attributes as the data item, ELEM.

Note: There are some (many?) software and hardware environments for which any overlapping MOVE such as the first example might "modify" the "contents of ELEM" during the middle of an attempt to move its contents back "on itself". This was not an issue in second Standard COBOL where the results of such a MOVE would have been explicitly undefined. If, however, amended third Standard COBOL makes this explicitly defined, it is at least questionable what GR(2) means when it states,

> "... or the content of the data item referenced by identifier-1 is moved to the data item referenced by each identifier-2 ..."

8. Depending on the answer to the previous question, is it the intent of GR(2) of the MOVE statement that for any elementary MOVE statement

```
MOVE elem-1 TO elem-2
```

that this statement work "as if" it were coded

```
MOVE elem-1 TO temp
MOVE temp TO elem-2
```

where temp is "an intermediate result item provided by the implementor" that has all the same explicit and implicit attributes as the data item, elem-1 (but not necessarily elem-2)?

9. Do the rules for Overlapping Operands in amended third Standard COBOL allow for an (apparently undefined) concept of "implicit data description entries"? For example, do the rules on page VI-69 mean that the results of executing the following example COBOL code would be undefined (because the sending and receiving items are not defined by the same explicitly coded data description entry) or would the results be defined (because the sending and receiving items are defined by the same implicitly defined data description entry)? (For either option, assume the remainder of the program is an ANSI conforming Report Writer program.)

```
Move Page-Counter to Page-Counter
```

10. Given the following example COBOL code,

```
01  Elem-Item       Pic X(5)    Value "12345".
01  Start-Pos-1     Pic 9       Value 2
01  Start-Pos-2     Pic 9       Value 3.
01  Length-1        Pic 9       Value 3.
01  Length-2        Pic 9       Value 3.

    ...

Move Elem-Item (Start-Pos-1 : Length-1) to
     Elem-Item (Start-Pos-2 : Length-2).
Move Page-Counter to Page-Counter
```

a. Should the term "item ... defined by the same data description entry" from paragraph 6.4.5 on page VI-69 apply? In other words, does

"Elem-Item (Start-Pos-1 : Length-1)"

reference 1 or 3 "data description entries"?

b. If the answer to the previous question is that "Elem-Item (Start-Pos-1 : Length-1)" references a single "data description entry", is it the same data description entry as "Elem-Item" or is it (depending on a previous answer) an "implicitly defined data description entry" associated with the "unique data item" created by reference modification as defined in GR(4) on page IV-23?

c. Depending on the answers to the previous two questions, are the sending and receiving items in this MOVE statement "item<s> defined by the same data description entry" or not? In other words, given the response to the questions above, are the results of the execution of the previous example defined or undefined in amended third Standard COBOL?

d. If the response to the previous questions are that the results of executing such a statement are defined, is it true that an ANSI conforming implementation must insure that the contents of Elem-Item would be "12234" after execution of this statement (assuming that the initial values of the fields had not been modified) or is it implementor-defined what can happen during the MOVE of the contents of the sending item to the receiving item?

11. If the results of executing the MOVE statement in the previous example are defined in amended third Standard COBOL, is it true that the results of executing the first MOVE statement in the following example are defined while the results of executing the second and third are undefined (even though most COBOL programmers would consider them to be logically equivalent)?

```
05  Group-1.
    10  filler      Pic X.
    10  Middle-3    Pic X(3).
    10  filler      Pic X.
05  Group-Red redefines Group-1.
    10  filler      Pic X(2).
    10  Last-3      Pic X(3).
...
Move Group-1 (2:3)  to Group-1 (3:3)
Move Middle-3       to Group-1 (3:3)
Move Middle-3       to Last-3
```

12. Given the following example COBOL code,

```
01  GROUP-1.
    05  ODO-OBJ     PIC 9.
    05  FULL-TABLE.
        10  TABL
            OCCURS 1 TO 9 TIMES DEPENDING ON ODO-OBJ.
            15  ELEM    PIC X.

    ...

    MOVE GROUP-1 TO GROUP-1.
```

a. Is it true that in second Standard COBOL that the results of executing this MOVE statement were explicitly undefined (because of the Overlapping Operand rules - regardless of any OCCURS ambiguities) while in amended third Standard COBOL the result of executing this MOVE statement are defined (because of the changes in Overlapping Operand rules)?

b. If the response to the previous question is that the results of this MOVE are now defined,

is it true that size of GROUP-1 would depend on the current value of ODO-OBJ when it is evaluated as a sending item while the size would be the maximum possible when evaluating it as a receiving item?

Response:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. The answer to Question 1 is "yes".
2. The answer to Question 2 is "no".
3. Question 3 requires no answer.
4. The answer to Question 4 is "yes".

Questions 5 through 12 cannot be answered pending further investigation of the issue of overlapping MOVEs.

Changes Needed in Standard:

A modification to a future standard is indicated as a result of this interpretation. Further investigation of overlapping operands in MOVEs will be pursued.

Interpretation A-363

SUBJECT: Incompatible Data

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-70, 6.4.7, Incompatible Data
2. Page VI-105, 6.19.4, MOVE Statement, General Rule 4b
3. Page VI-105, 6.19.4, MOVE Statement, General Rule 4b3

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL):

QUESTION:

The following has been coded:

```
...
File Section.
FD  any-file.
01  any-recordname.
    03  B  pic 999V99.
Working-Storage Section.
01  A pic X(5) value "MMMMM".
...
    Move A to B
    Write any-recordname
```

Question 1:

Is it true that item B after the execution of the MOVE statement contains MMM00?

I think that the answer is YES. Reference 1 states that the result of a reference is undefined when the content of a data item is not compatible with its class. However:

- this cannot be examined at the beginning of execution of the MOVE statement when item B has still to get its new content;
- after the execution of the MOVE statement there is no longer a "reference" to item B.

So, I think that the term "reference" as stated in Reference 1 cannot be applied to the receiving item of a MOVE statement.

Question 2:

If the answer to Question 1 is YES, is it true that the execution of the WRITE statement causes the content of a record in "any-file" to be MMM00?

RESPONSE:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. No, the results are undefined.

   The MOVE Statement, in general rule 4.b.3, states that "When the sending operand is described as being alphanumeric, data is moved as if the sending operand were described as an unsigned integer". The value MMMMM is not an integer and thus violates the general rules of the MOVE statement; the result is implicitly undefined.

   You are correct that the "content" of the receiving operand of a MOVE statement is not "referenced", and the incompatible data condition is not applicable to the receiving operand.

2. Since the answer to Question 1 is NO, this question is not answered.

CHANGES NEEDED IN STANDARD:

The standard does not precisely define when the content of a data item is referenced. This needs to be defined, and may depend on complete and precise definitions of sending and receiving fields. This will be addressed in a future standard.

Interpretation A-364

SUBJECT: Exponent Overflow

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page VI-76, General Rule 3, COMPUTE statement
2. Page VI-67, 6.4.2, ON SIZE ERROR phrase

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

COBOL Information Bulletin - 26
A-307

QUESTION:

Consider the following statements:

```
01  HOLD-FUT-AMOUNT PIC S9(09)V9(08).
01  AMOUNT          PIC S9(09)V9(08) VALUE 1000.
01  EFFECT-RATE     PIC S9(03)V9(14) VALUE 1.
01  YEARS           PIC S9(06)V9(05) VALUE 1000.
    COMPUTE HOLD-FUT-AMOUNT = AMOUNT
                * (( (1 + EFFECT-RATE) ** YEARS) - 1)
                / EFFECT-RATE
        ON SIZE ERROR
                PERFORM ERROR-RTN
    END-COMPUTE
```

The result of this operation will be approximately 10**303, much larger than can be expressed in Cobol. I would naturally expect the ON SIZE ERROR statement to kick in. However, my vendor seems to be distinguishing between two different events. If YEARS were to have the value 100 instead of 1000, the result would be approximately 10**33, too large for COBOL but large enough for the vendor to represent it internally, and the SIZE ERROR clause is properly executed. However, 10**303 is too large for any internal representation on the vendor's machine, and, when YEARS has the value 1000, my program terminates with an "Exponent overflow" condition.

1) Is there any basis at all for the vendor to claim that terminating in this manner is ANSI compliant? (I'll note that it specifically is not part of the Hardware Dependent

Language Element List on page XVII-94.) Please note that I am not particularly concerned with what happens if the SIZE ERROR clause is omitted.

2) Can the next revision of the standard eliminate any possibility of this being a legitimate action on the vendor's part?

RESPONSE:

The interpretation of ISO 1989:1985(ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. No, terminating in this manner is not conforming. For this example, a conforming implementation must either evaluate the expression as a value that will fit in the resultant identifier or transfer control to the imperative statement specified in the ON SIZE ERROR phrase.

2. Your suggestion will be considered for a future revision.

CHANGES NEEDED IN THE STANDARD:

The relationship between SIZE ERROR and intermediate results is being explored for a future standard.

Interpretation A-365

SUBJECT: Overlapping Arithmetic

REFERENCES:

ISO 1989:1985, Programming Languages -- COBOL (ANSI X3.23-1985, Programming Language -- COBOL):

1. Page I-7, Paragraph 1, Section 5.2.3
2. Page I-8, Section 1.5.2.52
3. Page VI-69, Section 6.4.5

ISO 1989:1985/Amd.1:1992, Programming Languages -- COBOL Amendment 1: Intrinsic function module (ANSI X3.23a-1989, Programming Language -- Intrinsic Function Module for COBOL)

QUESTION:

This topic was dismissed rather too shortly I thought in the discussion of overlapping moves.

Question 1: Does the following code conform to the Standard?

```
01  A PIC 9(3).
01  B REDEFINES A PIC 9(1).
    ADD B TO A.
```

Please do not dwell on the advisability of coding in such a manner - I can easily come up with a more complex statement for which this type of coding is a reasonable solution.

The discussion on overlapping operands (section 6.4.5 page VI-69) omits the definition of "sending" and "receiving" fields. While I think it's clear that MOVEs have sending and receiving fields, I'm not so sure about arithmetic statements. Furthermore, an overlapping MOVE could be implemented in any of several non-compatible ways, but I can't see any such ambiguity in the handling of arithmetic statements. However, the general rules for the arithmetic statements clearly state that the rules about overlapping operands apply (e.g. 6.6.4(5) on page VI-74).

Question 2: Neither of our COBOL vendors flags the statement above as non-compliant.
Are these conforming implementations?

Question 3: Is there any reason for the overlapping operands rule to apply to

arithmetic statements?

Question 4: Can the next revision of the standard eliminate this restriction, which would certainly lead to greater clarity in the standard, and which is not being recognized by a number of vendors anyhow?

Response:

The interpretation of ISO 1989:1985 (ANSI X3.23-1985) amended by ISO 1989:1985/Amd.1:1992 (ANSI X3.23a-1989) is the following:

1. Yes, this is conforming source code; however, "the result of the execution of such a statement is undefined". (Reference 3).
2. Yes, these are conforming implementations. There is no requirement in standard COBOL for flagging syntax that can result in undefined behavior.
3. We are unable to answer your question. We seldom have the information necessary to justify the decisions of past language designers.
4. Your suggestion will be considered for a future revision.

Changes Needed in standard:

Three areas for investigation have been identified:
1) allow overlapping operands in arithmetic statements
2) more clearly define rules for overlapping operands
3) define additional operand concepts; sending and receiving fields are not sufficient to classify all operands for all statements.

INDEX OF INTERPRETATIONS

INTRODUCTION

This index presents a summary of COBOL interpretations arranged by subject matter. The reference indicates the identification number of the interpretation and the X3 COBOL Information Bulletin (CIB) in which it is contained. A corresponding SC22 document exists for each X3 document, as follows:

CIB 24 SC22 N 1517, US Contribution on COBOL: COBOL Information Bulletin Number 24
CIB 25 SC22 N 1518, US Contribution on COBOL: COBOL Information Bulletin Number 25
CIB 26 SC22 N 1604, Record of Response Issue 1, Programming Languages - COBOL
CIB 27 SC22 N 1634, Record of Response Issue 2, Programming Languages - COBOL
CIB 28 This document

COMPLIANCE WITH THE STANDARD

COMPILER DIRECTING FACILITY

LANGUAGE FUNDAMENTALS

FILES

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

PROGRAM STRUCTURE

IDENTIFICATION DIVISION

ENVIRONMENT DIVISION

DATA DIVISION

PROCEDURE DIVISION

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

INTRINSIC FUNCTIONS

COMMUNICATIONS FACILITY

REPORT WRITER

PORTABILITY GUIDELINES