An overview of the 1974 COBOL standard

by MARGARET M. COOK, GEORGE N. BAIRD, WILLIAM M. HOLMES, PATRICK M. HOYT, L. ARNOLD JOHNSON and PAUL OLIVER

Department of the Navy
Washington, D.C.

INTRODUCTION

Since 1 July 1972, all COBOL compilers brought into the Federal Government have to be identified as implementing one of the levels of the Federal COBOL Standard. The National Bureau of Standards, which has the responsibility for the development and maintenance of Federal ADP Standards, has delegated to the Department of Defense the responsibility for the operation of a Government-wide COBOL Compiler Testing Service. This responsibility is discharged by the Federal COBOL Compiler Testing Service (FCCTS), an activity of the Department of the Navy's Automatic Data Processing Equipment Selection Office Software Development Division, through the implementation and maintenance of the COBOL Compiler Validation System (CCVS),[1] a comprehensive set of computer programs used to test COBOL compilers for compliance with the Federal COBOL Standard.

In May 1974, the American National Standards Institute approved ANS Programming Language COBOL, X3.23-1974[2] as the national standard for the COBOL language replacing USA Standard COBOL, X3.23-1968.[3] Federal Information Processing Standards Publication 21-1[4] adopts X3.23-1974 (minus the Report Writer module) as the Federal COBOL Standard. As a result of these actions, the Testing Service is engaged in the development of a new COBOL Compiler Validation System, incorporating tests for the revised language. This paper presents an overview of COBOL 74, highlighting the new features in the language, major language deficiencies, and important contributions to the programming discipline. All comments made with regard to COBOL 68 or COBOL 74 will be based on the language as defined in References 3 and 2, respectively.

REVISIONS TO THE COBOL 68 MODULES

The Nucleus, Sequential I-O and Library modules of COBOL 68 have undergone major revisions. The Table Handling and Segmentation modules are essentially the same modules as appeared in the 1968 COBOL Standard. The Random Access module has been replaced by the new modules, Relative I-O and Indexed I-O, and the specifications for the Report Writer module have been rewritten. The Sort module has been expanded into the Sort-Merge module by the addition of the MERGE verb.

The Table Handling module in COBOL 74 consists of two levels instead of the previous three levels, and the punctuation changes in the 74 Standard have relaxed the rigid rules for the use of parentheses and commas in referencing table items. Literals and index-names may be mixed when referencing a table item, and an index may be incremented or decremented by a negative value.

In the remainder of this section we concentrate on the new features of the Nucleus and Library modules. The I-O modules will be covered in a later section, and we ignore the Report Writer as being beyond the scope of the Federal Standard.

Section 2.3 of Appendix B of the X3.23-1974 Standard[2] documents in greater detail the modifications made to the 1968 Standard and the additional language features added for the 1974 Standard.

Nucleus

Three new verbs have been added to the Procedure Division of the Nucleus module; INSPECT, STRING and UNSTRING. The INSPECT verb replaces the old EXAMINE verb and provides expanded editing capabilities. With an INSPECT statement one can count, and replace occurrences of either single characters or groups of characters in a data item. For each of these INSPECT functions, the BEFORE or AFTER phrase allows one to specify that the function begins or ends upon encountering a given character or group of characters.

As an example of the INSPECT statement, consider the following source code. (Coding examples in this paper will contain formatting errors due to the typesetting requirements.)

```
  DATA DIVISION entries
01   ID-1 PIC X(54) VALUE '␣␣DEFEAT␣␣␣␣␣␣
       DEDUCT␣␣␣␣␣␣␣␣␣␣
       DEFENSE␣␣␣␣␣␣␣␣␣␣DETAIL'.

  PROCEDURE DIVISION entries
INSPECT ID-1 REPLACING ALL '␣␣DE' by 'THE␣',
  FIRST 'A' BY 'E',
  FIRST '␣␣␣␣' BY '␣OF␣',
  FIRST 'T' BY 'K' AFTER INITIAL 'UC',
  FIRST '␣␣␣␣␣␣␣␣' BY '␣GO␣OVER␣',
```

```
FIRST 'S' BY 'C',
FIRST '␣␣␣␣␣␣␣␣' BY '␣BEFORE␣' AFTER INITIAL
'EN'.
```

After executing the above INSPECT statement ID-1 would contain

```
'THE␣FEET␣OF␣THE␣DUCK␣GO␣
          OVER␣THE␣FENCE␣BEFORE␣THE␣TAIL'.
```

This example points out one of the limitations on the use of INSPECT; the number of characters being replaced must equal the number of characters by which they are replaced. Thus, 'DE' cannot be replaced by 'THE␣' but the characters '␣␣DE' can be replaced by 'THE␣'.

With the 1974 Standard, the COBOL language has the features necessary for manipulation of character strings without each individual character string beginning in the leftmost character position of a data item. The STRING statement allows a user to build a single data item from two or more data items. The sending data items may be delimited by one or more character(s), or the entire sending item can be part of the receiving item. A user can also indicate the relative starting position in the receiving item for the STRING operation.

The following example illustrates the use of the STRING statement.

DATA DIVISION Entries

```
01   WISH-LIST.
     02   FILLER PIC X(4) VALUE 'GEE␣'.
     02   FILLER PIC X(33) VALUE SPACES.
01   REL-POSITION PIC 99 VALUE 5.
01   STRING-VALUES.
     02   FIELD1 PIC X(13) VALUE 'I␣WISH␣I,␣YOU'.
     02   FIELD2   PIC   X(24)   VALUE
          'WAS␣A␣FORTRAN␣PROGRAMMER'.
```

PROCEDURE DIVISION Entries

```
PARAGRAPH-1.
     STRING FIELD1 DELIMITED BY ', ',
          SPACES, FIELD2 DELIMITED BY SIZE
          INTO WISH-LIST WITH POINTER REL-PO-
          SITION,
          ON OVERFLOW GO TO PARAGRAPH-2.
          ⋮

PARAGRAPH-2.
```

After the execution of the STRING statement, the data item WISH-LIST contains 'GEE␣I␣WISH␣I␣WAS␣A␣FORTRAN␣PROGRAMMER'.

The opposite is achieved by the UNSTRING statement which causes data in a sending field to be separated based on one or more delimiters, and placed into multiple receiving fields. A delimiter may be a single character or a combination of characters.

The SIGN clause allows a user to specify whether a separate character position is used for the sign in numeric items and to indicate whether the position of the sign is leading or trailing.

The PROGRAM COLLATING SEQUENCE clause permits one to specify ASCII, or some other collating sequence. However, changing the collating sequence for a program will necessarily require consideration of all statements which depend upon a character's relative position in a given collating sequence; for example, the SEARCH statement, the SORT/MERGE statement, or alphanumeric comparisons in an IF statement.

Library

Several major changes have been made to the Library module. There are now no restrictions on where a COPY statement may appear in a COBOL program. A COPY sentence may appear anywhere that a COBOL word may appear. As a result, rather strange looking source code can be produced, e.g.,

```
ADD COPY XX. TO B.
```

If the content of the library text XX contains the single identifier A, the results after the COPY takes place are:

```
ADD A TO B.
```

There can be more than one library available at compile time. In this case the COPY sentence must contain the name of the library in which the text resides. The presence of the REPLACING phrase in a COPY sentence causes the library text being copied to be edited prior to being inserted in the program. There are several levels of editing that can be used. Words, literals, identifiers and pseudo-text can be replaced by like or different types of operators.

Pseudo-text is bounded by pseudo-text delimiters, which are matching sets of double equal signs (= =), much like a nonnumeric literal which is delimited by sets of matching quotation marks. The replacement of pseudo-text is based on finding a matching set of character-string(s) contained in the library text. The replacement string may be longer or shorter than the characters replaced.

The following COPY statement illustrates the use of pseudo-text to edit a library entry.

User's COBOL Library LIBRARY-TEXT

```
01   ID1 PIC X(54) VALUE IS
     '␣DEFEAT␣DEDUCT␣DEFENSE␣
     DETAIL␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
     ␣␣␣␣␣␣␣␣␣␣'.
     COBOL Source Statement:
          COPY LIBRARY-TEXT REPLACING
       = =␣DE= = BY = =THE␣= =
       = =AT= = BY = =ET␣OF␣= =
       = =CT= = BY = =CK␣GO␣OVER␣= =
       = =SE= = BY = =CE␣BEFORE␣= =
```

The compiled results of a program with the above COPY statement are the same as a program in which the following source code appeared:

```
01  ID1 PIC X(54) VALUE IS
      "THE␣FEET␣OF␣THE␣DUCK␣GO␣OVER␣
                THE␣FENCE␣BEFORE␣THE␣TAIL".
```

THE DEBUG MODULE

In the 1968 Standard there were no explicit procedures for specifying what debugging actions, if any, would take place during the execution of a COBOL program. The DEBUG module for the 1974 Standard includes language elements which are designed for debugging COBOL programs. Their inclusion in the COBOL language permits a user to consider the debugging of source programs in the design of an application, not as an afterthought when problems are encountered.

The USE FOR DEBUGGING statement identifies the user items that can be monitored by the associated debugging section. The debugging algorithm which is defined in the debugging section can be controlled by both a compile time switch and an execution time switch. Debug lines are source statements whose inclusion or omission from an object program is controlled by a compile time switch. There is a special register called DEBUG-ITEM which can be accessed in debugging sections. This special register contains information relative to the source code which causes the execution of a debugging section.

INTER-PROGRAM COMMUNICATION

The Inter-Program Communication module provides a means for a program to transfer control to one or more subprograms and the sharing of data among these programs. The action which is taken when there is not enough object time memory can be specified and the memory areas occupied by called programs can be released and made available to the operating system.

Features

The CALL statement causes control to be transferred from one object program to another object program in the same run unit. If the subprogram name is known at compile time, the CALL statement operand is a nonnumeric literal. The subprogram name can also be specified dynamically as the contents of a data-name.

The data items which are shared with the called program are specified in the USING phrase of the CALL statement. In the subprogram, the shared items are specified in the USING phrase of the PROCEDURE DIVISION header and defined as data descriptions in the LINKAGE SECTION. There must be a one-to-one correspondence between the operands in the CALL statement and the operands in the PROCEDURE DIVISION header of the called program. The data descriptions for corresponding operands must define an equal number of character positions but the data descriptions do *not* have to be identical. No space is allocated in the called program for items defined in the LINKAGE SECTION, and references to items in the LINKAGE SECTION are resolved at object time.

The CALL statement also permits the user to specify the action to be taken when there is not enough memory available for a subprogram. This is done in the imperative-statement of the ON OVERFLOW phrase of the CALL statement.

The CANCEL statement releases the memory area occupied by the program referred to in the CANCEL statement. A CALL to a program that has been cancelled causes that program to be loaded and executed in its initial state.

The EXIT PROGRAM statement marks the logical end of a called program. Control is returned to the calling program when this statement is executed. More than one EXIT PROGRAM can appear in a subprogram, but the EXIT PROGRAM statement must be the only sentence in a paragraph.

Inadequacies

The INTER-PROGRAM COMMUNICATION module restricts the possible operands which may appear in the USING phrases. The operands must refer to 77 or 01 level-number items and may not be defined in the REPORT SECTION of the calling program.

There are implementor-defined areas in the Inter-Program Communication module which could cause problems in program portability between systems. The relationship between the operand in the CALL/CANCEL statement and the referenced program is implementor-defined. This means that even though the program-name is a user-defined word and thus could be 30 characters, the implementor may limit the actual number of characters which are used to establish linkage between the called and calling programs. If a user has subprograms on a system which recognizes the first 10 characters of the program-name and moves to an implementation which recognizes the first six, then any referenced program-names with the first six characters identical would be treated as referencing the same subprograms.

The action to be taken when there is not enough memory available for a called program can be specified in the ON OVERFLOW phrase of the CALL statement. If this phrase is not specified, the effects of the CALL statement are defined by the implementor.

THE INPUT-OUTPUT MODULES

The major enhancement of COBOL 74 over COBOL 68 is in the revision to the input/output modules. The

Sequential I-O module has been revised, and the Random Access module has been replaced by two new ones; Relative I-O and Indexed I-O, with some degree of functional and syntactic similarity existing between the new Relative I-O module and the old Random Access module. Taken as a group, the three I-O modules provide the COBOL programmer with the file handling capabilities which are well beyond what has been possible previously.

Sequential I-O

The Sequential I-O module for the 1974 Standard has all the features of the 1968 Standard with the exception of user defined labels and declarative label processing sections, which are no longer supported by the COBOL language. The data-name option of the LABEL RECORDS clause and the USE statement option for label record processing were deleted.

A major new feature in the 1974 Standard allows a user to process character code sets other than the system's native character code set on input and output operations. The CODE-SET clause of the File Description entry specifies the character code set which is used to represent the data on the external media. When this clause is included in a File Description entry, characters are converted to the native character set on input or converted from the native set to the code specified on output. The CODE-SET clause can only be used in File Descriptions which are not mass storage files and the external code representations are limited to the ASCII code set, the native character code set or other character code sets supported by the implementation.

The Sequential I-O module provides the capability to add records to the end of an existing sequential file. Execution of an OPEN EXTEND statement positions a file immediately following the last logical record of the file. Subsequent WRITE statements for that file add records to the end of the file as if the file had been opened in the OUTPUT mode.

The FILE STATUS clause of the File-Control entry and the REWRITE statement are also elements of the Relative I-O and Indexed I-O modules and these two new features are discussed later.

In the Sequential I-O module for the 1974 Standard one can describe the logical page format for a printer-destined file through the LINAGE clause in the File Description entry. The LINAGE clause specifies the size of the top and bottom margins for a logical page, the number of lines comprising the page body, and the line number within the page body where the footing area begins. The values given in the LINAGE clause may be specified as integer constants or the contents of data-names. If the values are integer constants each page has the same format throughout execution of the programs. If the values are the contents of data-names, the values at the time the file is opened specify the first logical page. Each time a new page is started, the values of the data items are examined to determine the values for the current page. Thus the logical format and size of a page can be changed for each new page in a file designated for printer output.

New Input-Output Modules

The INDEXED I-O and RELATIVE I-O are new modules in the 1974 Standard. Because the Indexed I-O module is the most complex and important of the two, we will concentrate our remarks on it, mentioning important features of the Relative I-O module where appropriate.

Features

The INDEXED I-O and RELATIVE I-O modules provide the capabilities for accessing a file in a predefined mode. INDEXED I-O also provides the capability of defining several paths of information retrieval.

In a Relative file, records may be stored "randomly", but are identified by a relative record number on which record storage and retrieval is based. The record number, or key, must be unique, and is the only means by which the file may be accessed. The record storage relationship for an Indexed file however is based upon one or more indexes associated with the file. Thus, an Indexed file may be accessed through one or more record keys.

The prime and alternate keys are defined within the SELECT clause of the FILE-CONTROL paragraph for Indexed files. The contents of the prime key must be unique for each record in the file. This is the base key from which the file is constructed, and is used for inserting, updating, and deleting records. The user may specify one or more alternate keys for the file. Unlike the prime record key, alternate keys may be non-unique.

The content of the keys which are used to retrieve records are considerably more flexible in the Indexed I-O module than in the Relative I-O module. Under the Relative I-O file organization the record reference key must be an unsigned integer and is defined outside the record description entries for the file. In the Indexed I-O organization the record reference key must be an alphanumeric data item which can contain any combination of characters in the computer's character set, and must be defined within a record description entry for the file.

The language specification for COBOL 74 allows for sequential, random or dynamic access within both Relative I-O and Indexed I-O modules. The access mode for a given file is indicated by use of the ACCESS MODE IS RANDOM or ACCESS MODE IS SEQUENTIAL clause. When the ACCESS MODE IS DYNAMIC is specified, records may be processed either randomly or sequentially through use of the appropriate I-O statement. The access mode for a file need not be the same as the mode in which the file was created. When accessing the file sequentially, the records are retrieved in ascending order based on the key contents. The START statement provides positioning within an indexed or relative file for subsequent sequential retrieval of records. When processing the file randomly,

records are stored or retrieved based on the data contents of the record key. Records are accessed based on the current key of reference. The key of reference, prime record key or alternate record key, is established at the COBOL instruction level. The default key of reference is the prime record key, but an alternate key of reference may be specified in a random READ statement. Any subsequent sequential read uses the key of reference established by the last random read or START statement.

File maintenance

The means of maintaining mass-storage files, i.e., record insertion, record deletion and record updating is accomplished through the use of the verbs DELETE, WRITE and REWRITE. Only the prime record key associated with the file is used in providing all file maintenance functions, i.e., RELATIVE KEY for Relative I-O and RECORD KEY for Indexed I-O.

The DELETE verb logically removes a record from the file. Once the statement has been executed, the record cannot be accessed again. The WRITE and REWRITE verbs insert and update, respectively, the records in the file. Any file maintenance key associated with the file must be unique within all the records for the file. The previous input-output statement for the file must have been a READ statement. The REWRITE statement causes the last record READ by the program to be logically replaced by the specified record. The number of character positions in the record being rewritten and the record being replaced must be equal. The WRITE statement causes a record to be inserted assuming the key does not already exist.

An important addition to the 1974 language specification for all I-O modules is the FILE STATUS data item which contains information as to the success (or failure) of an I-O operation. The FILE STATUS clause, located in the FILE-CONTROL entry for a file, specifies a two character alphanumeric data item and contains values which indicate the results of every statement which references that file explicitly or implicitly. The operating system moves the values into the file status data item upon the completion of any statement which references the file.

The new I-O modules provide the user greater flexibility and a wider range of functions than have been previously available. The Indexed I-O module in particular gives the user the ability to implement multi-key retrieval functions and provide a closer relationship of the capabilities of data base management systems entirely within the scope of the COBOL Language.

THE COMMUNICATION MODULE

The motivating factor behind including a "Communication Module" in COBOL 1974 was the advent within the past decade of computer systems using remote terminals and the use of these terminals for message processing applications. Heretofore, the COBOL user has been unable to perform this class of interactive operations without resorting to system dependent facilities such as assembly language support routines. Many implementations of COBOL permitted limited degree of access to remote terminals via the ACCEPT and DISPLAY verbs, but obviously these posed serious limitations on capabilities by limiting transmission to an "on demand" basis and in no case could a program be notified of unsolicited input from a terminal.

The Communication module in COBOL 1974 attempts to solve these deficiencies by providing four new I/O verbs (RECEIVE, SEND, ENABLE, DISABLE) and interfacing COBOL programs to any configuration of remote terminals via a set of message queues. The operation of the message queues and the remote terminals is handled by a Message Control System (MCS), a "black box" software package which is largely implementor defined and by its very nature is system-dependent. The MCS must provide the logical interface between the COBOL communication object program and the systems network of communication devices by performing line discipline, including such tasks as dial-up, polling, and synchronization, and by performing device-dependent tasks such as character translation and insertion of control characters. The COBOL programs interface with the MCS through the programs' Communication Descriptions or CD's. CD's are placed in the Communication Section which follows the File, Working-Storage, and Linkage Sections in the Data Division. The CD establishes either an input or an output path for messages, and provides parameter fields for passing information between the program and the MCS.

Features

For the Procedure Division, the Communication Module introduces four new verbs; RECEIVE, SEND, ENABLE, and DISABLE; plus a new variation for an old one, ACCEPT. The following paragraphs summarize these statements as presented in the formal language specification for the Communication module.

"The RECEIVE statement makes available to the COBOL program, a message, message segment, or a portion of a message or segment." Prior to executing a RECEIVE the user must specify in his input CD record area, the name of the queue or subqueue he wishes to address. Executing the RECEIVE then effects dequeuing of a message from the appropriate queue and placing of that message into the field designated by the user. If the addressed queue is empty, then at the user's option, the program can be forced to wait until a message becomes available, or it can be directed to proceed immediately with execution of the next sequential statement. During the RECEIVE operation, all data items in the input CD record are updated by the MCS.

"The SEND statement causes a message, a message segment, or a portion of a message or segment to be

released to one or more output queues maintained by the MCS."

Prior to executing a SEND, the user must specify in his output CD record area, the number and names of the destinations to which the data is to be sent, plus the length of the text in characters. In the SEND statement itself, the user specifies which transmission sentinel is to be used, end-of-message, end-of-group, or end-of-segment. When a SEND is executed, the MCS must enqueue the data on the appropriate queue(s) and return the operation status to the CD for use by the program. Line and page control may be exercised on line oriented devices.

A variation of the ACCEPT statement enables the user to determine the number of messages currently enqueued in any particular queue. Prior to executing the ACCEPT statement, the user must specify in his input CD, the name of the queue or subqueue whose size is to be returned. The MCS will return to the CD in the appropriate parameter fields, the message count and the status of the operation. Only input queues may be measured in this way.

The ENABLE and DISABLE statements direct the MCS to allow and inhibit respectively, data transfer between specified output queues and destinations for output, or between specified sources and input queues for input. The queues or destinations involved must be named in the appropriate CD before execution of the statement. These statements make and break the logical connections between the queues managed by the MCS and the network of communication devices, but they do not affect the logical connections between the various queues and the program itself. The specification of a key or system password is required in both statements "in order to prevent indiscriminate use of the facility by a COBOL user who is not aware of the total network environment, and who may therefore disrupt system functions by the untimely issuance of ENABLE and DISABLE statements".

A special option permits the user to specify the symbolic name of a specific device in his input CD, and then request enabling or disabling of the logical paths between that device and *all* queues and subqueues linked to it.

Finally, there is the added capability to designate in a COBOL communication program that it is to be scheduled for execution automatically by the MCS whenever the MCS determines that there is message processing to be done. This is accomplished by specifying an option in one input CD in the program. Subsequently, when the MCS invokes the program, it will place the name of the queue or subqueue, which prompted the action, in the appropriate parameter fields of the input CD. There are means to test within the program whether the program was invoked by the MCS or scheduled through job control language.

Inadequacies

The basic problem with the Communication module as it now stands is the fact that the MCS and its interface to the network of peripheral devices is so ill-defined. Although the concept of the MCS is never formally introduced except in Appendix C of the Standard, its existence is implied throughout the specification for the Communication module by frequent references to it (the MCS) by name. Unfortunately, the appendices are not considered a part of the formal COBOL language specification; and thus they are in no way binding. At best, the appendix can be considered a suggested guideline for implementation. In other words, it can only serve to enlighten the reader as to what the Programming Language Committee (PLC) of the Conference on Data Systems Language (CODASYL) might have had in mind when they designed the specification for the Communication module.

Contributions

It is not really possible to assess the usefulness of the new Communication module. Such an assessment must await the use of the features in a communication environment. The features themselves are not extensive, and the heart of the system, the MCS, is not well defined. These considerations suggest a degree of skepticism vis-a-vis the degree of applicability of the Communication module. Despite this skepticism, it must be admitted that some capability is now available for message handling.

RECOMMENDATION

COBOL 74 is in many ways a vast improvement over COBOL 68. New features have been added which contribute to the capability of the language (the new I-O modules), enlarge its scope (the Communication Module), and enable the programmer to produce a better product (the Inter-program Communication and Debug modules). Furthermore, old modules have been enlarged and improved. We believe the efforts of the standardizing body, ANSC X3J4 (COBOL) should now be directed toward producing a more complete and precise definition of the language.

The procedures of the American National Standards Institute require that action be taken to reaffirm, review, or withdraw the standard no later than five years following the publication of the current standard. X3J4 should seriously consider a radical rewrite of the standard in an effort to make it more a standard and less a generalized user's manual. A "standard" is supposed to be an authoritative measure by which correctness of other things may be determined (condensed from Websters'), but it is a difficult task to measure correctness against a standard which is ambiguous and subject to interpretation. Both the syntax and semantics of the language should be expressed as formal grammars and defined in an appropriate metalanguage. The efforts of ANSC X3J1 to so define the PL/1

language is an admirable example of what can be done. When a language is well defined, there can be no ambiguities and no doubt as to what is valid and what is not. There is seldom a need for interpretation. Such definitions are also of significant value when a compiler implementor chooses to use a syntax-directed or other automated parsing technique. Finally, a precise definition would make much simpler the task of determining the degree to which COBOL compilers conform, in their translation of COBOL programs, to the Standard.

REFERENCES

1. Baird, G. N., "The DOD Compiler Validation System," *Proc. 1972 FJCC*, AFIPS Press, Volume 41, pp. 819-827.
2. *American National Standard Programming Language COBOL*, X3.23-1974, American National Standards Institute Incorporated, New York 1974.
3. *American National Standard COBOL X3.23—1968*. American National Standard Institute Incorporated, New York 1968.
4. *Federal Information Processing Standards*, Publication 21-1, U. S. Government Printing Office, Washington, D. C., (pending).