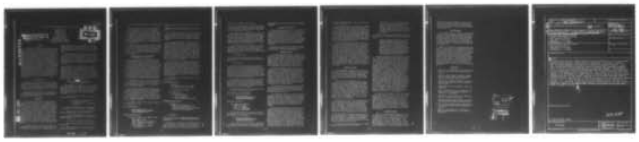
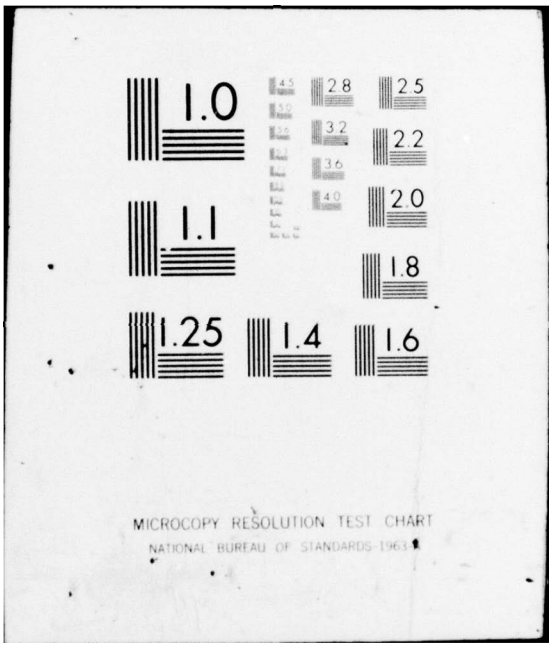
AD-A039 739 FEDERAL COBOL COMPILER TESTING SERVICE WASHINGTON D C F/G 9/2
THE STANDARDIZATION AND EVOLUTION OF THE COBOL LANGUAGE,(U)
MAY 77 P M HOYT, G N BAIRD, M M COOK
UNCLASSIFIED FCCTS/TR-77/11 NL

1 OF 1
AD
A039739



END

DATE
FILMED
6-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

# THE STANDARDIZATION AND EVOLUTION OF THE COBOL LANGUAGE

Patrick M. Hoyt
George N. Baird
Margaret M. Cook
William M. Holmes
L. Arnold Johnson
Paul Oliver
Department of the Navy
ADPE Selection Office
Software Development Division
Washington, D. C. 20376





## ABSTRACT

Government-wide testing of COBOL Compilers is the responsibility of the Federal COBOL Compiler Testing Service, an activity of the Department of the Navy's Automatic Data Processing Equipment Selection Office, Software Development Division. In May 1974, the American National Standards Institute approved ANS Programming Language COBOL, X3.23-1974 as the national standard for the COBOL language. Specifications for this revision were drawn from USA Standard COBOL X3.23-1968 and the CODASYL COBOL Journal of Development, dated 31 December 1971. As a result, the Testing Service is engaged in the development of a COBOL Compiler Validation System for this new Standard, incorporating tests for the revised language into the existing 1968 COBOL Compiler Validation System. The first part of the paper focuses on the rationale behind changes from the '68 to '74 COBOL Standard as well as highlighting the new features in the language. The second part of the paper discusses evolution within the COBOL language itself with respect to the CODASYL COBOL Journal of Development and several other CODASYL language activities.

## INTRODUCTION

The first part of this paper will focus on the rationale behind changes from the '68 to the '74 COBOL Standard as well as highlight the new features in the language. 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.

The second part of the paper will discuss evolution within the COBOL language itself with respect to the CODASYL COBOL Journal of Development and several other CODASYL language activities.

## BACKGROUND

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] (COBOL '74) as the national standard for the COBOL language replacing USA Standard COBOL, X3.23-1968[3] (COBOL '68). Language specifications for this revision were drawn from the COBOL '68 and the CODASYL COBOL Journal of Development[4] (JOD) dated 31 December 1971.

Federal Information Processing Standards Publication 21-1[5] 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 '74 COBOL Compiler Validation System, incorporating tests for the revised language into the existing operational '68 COBOL Compiler Validation System.

Appendix A of the X3.23-1974 Standard presents a history of COBOL. As one follows the history of changes to the Standard, one looks for a planned direction and growth of the COBOL language. Do we see evolution toward a "foreseeable" future language specification and standard? If so, can we state the final specifications to see where we stand along the road of progress? If it appears that we cannot, then where is COBOL headed with respect to the revision of COBOL '74 and the CODASYL Journal of Development?

## CRITERIA FOR CHANGES IN THE STANDARD (COBOL '63 TO COBOL '74)

When considering all changes made in the revision process from COBOL '68 to COBOL '74, one must consider the eight criteria against which each proposed change was measured. These were chosen and published by the X3J4 committee in Section 2.1 of Appendix B of the X3.23-1974 Standard, and are shown here for later reference:

(1) The general usefulness of an element or function in terms of:

a. The degree of implementation;
b. Acceptance by users;
c. The degree to which a function was required.

(2) The overall functional capability of the language, considering such things as redundancy.

(3) The state-of-the-art technology with regard to implementing the language feature.

(4) The usefulness, in terms of application requirements, of language capabilities withia each level of a module.

(5) Compatibility with other standards.

(6) The cost of implementing versus advantages of use.

(7) Overall language consistency within a defined level or module.

(8) Upward compatibility of levels within a module.

One must carefully examine each of these eight criteria and ask whether the impact of the proposed change to the language is measurable as a quantity (dollars and cents), or is measurable only in a qualitative sense (relatively good or bad). When a vendor makes a qualitative judgment with respect to a standard, how objective can his representative be? We all must realize that vendors have a large investment in existing software products which may include certain language elements and features that are extensions to the current standard.

As the changes in the Standard for COBOL are presented, it will be left to the reader to decide whether each change is truly necessary and follows the published policy for change (the aforementioned eight criteria). This is because most DP managers will face the decision as to whether to implement COBOL '74 at their installations.

## REVISIONS TO THE COBOL '68 MODULES AND NEW FEATURES OF COBOL '74

Nucleus, Sequential I-O and Library modules of COBOL '68 have undergone major revisions. The Segmentation module is essentially the same as in the 1968 COBOL Standard. The Random Access module has been replaced by two new modules, Relative I-O and Indexed I-O, and the specifications for the Report Writer module have been rewritten. Addition of the MERGE verb has expanded the SORT module into the Sort-Merge module, and the Table Handling module consists of two levels instead of the previous three, but is otherwise little changed. Section 2.3 of Appendix B of the X3.23-1974 Standard documents in greater detail the modifications made to the 1968 Standard and the additional language features added for the 1974 Standard.

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 since it is beyond the scope of the Federal Standard.

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 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 string of characters.

As an example of the INSPECT statement, consider the following source code:

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␣'.

By removing EXAMINE from the COBOL reserved word list in the COBOL '74 compilers, both vendors and users must transform all uses of the EXAMINE to the new expanded syntax of INSPECT. TALLY is no longer a special register provided by the compiler and would need to be added to the DATA DIVISION of every application program which references it. Although translators could change simple occurrences of EXAMINE to INSPECT, more complex EXAMINE statements would have to be manually translated. The cost for this process will be considerable; however, no data for the total number of occurrences of EXAMINE in all applications for all COBOL users is available.

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 option.

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(14)
        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-POSITION,
     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, and alphanumeric comparisons in an IF statement.

Several major changes have been made to the Library module. There are no restrictions now 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 result, 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 (==). 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'.
```

Here we have a definite expansion of capabilities of an existing language element. This should aid in establishing and constructively using production program libraries for structured programming development techniques.

## 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 modules; 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 file handling capabilities which are well beyond what has been possible previously.

The features of the three I-O modules as well as file maintenance capabilities are described in detail in "An Overview of the 1974 COBOL Standard"[6].

## OTHER NEW MODULES

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.

The increased size of source and object programs will add to the cost to compile and execute application programs using DEBUG; this must be carefully measured and compared to the expected savings in the program development process.

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

There are implementor-defined areas in the Inter-Program Communication module which could cause problems in program portability between systems. For instance, 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 only the first six, then any referenced program-names with the first six

characters identical would be treated as referencing the same subprograms.

The motivating factor behind including the COMMUNICATION module in COBOL '74, was the advent within the past decade of computer systems using remote terminals and the use of these terminals for message processing applications. The Communication module provides four new I/O verbs (RECEIVE, SEND, ENABLE, DISABLE) and the capability of 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 an implementor-defined Message Control System (MCS).

The basic problem with the Communication module, as it now stands, is 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 Languages (CODASYL) might have had in mind when it designed the specification for the Communication module.

## EVOLUTION OF COBOL

Having presented an overview of changes and new features from COBOL '68 to COBOL '74, one can explore the question of growth in the COBOL language with respect to Operating System Command Languages (OSCL), data base efforts in Data Definition Language (DDL) and Data Manipulation Language (DML), and mini-COBOL efforts.

There is current international interest and activity in the area of a standardized, user oriented Operating System Command Language (reference CODASYL OSCL Task Group and IFIPS TC-2 Working Conference on Command Languages). The file manipulating features, Inter-Program Communication and Communication modules of COBOL '74 relate to this subject area. There has so far been no formal coordination between CODASYL'S PLC, ANSC X3J4, and the OSCL groups on possible standard features of operating systems, status and conditions, and error responses to the user. It is essential that future coordination between these and similar groups be accomplished.

Currently we see a host of new implementations of data base management systems. How these systems will relate to the work of the Data Base Language Task Group of CODASYL'S Programming Language Committee is yet to be determined. The stated purpose of PLC's DBLTG is to prepare a proposal[7] to include in COBOL a data base facility which is based on the April 1971 DBTG Report. ANSC X3J4, user, and vendor acceptance of this effort will be determined in the near future.

Since the CODASYL Programming Language Committee (PLC) is responsible for the continuing development of the COBOL language it is important to understand how this group functions with respect to accepting proposals for changes to COBOL and publishing the JOD when these changes have been agreed upon.

In Section 2.1.4 of the JOD, one finds the stated purpose and objectives of PLC as follows:

> "The objectives are to make possible: compatible, uniform, source programs and object results, with continued reduction in the number of changes necessary for conversion or interchange of source programs and data. The PLC concentrates its efforts in the area of tools, techniques, and ideas aimed at the programmer."

Evolution of COBOL is a dynamic process as noted in Section 2.2.1 of Appendix B of the X3.23-1974 Standard:

> "Since the language, and hence the JOD, is constantly changing, it was necessary to select the JOD of a given date to serve as a base document for the revision process. This date, known as the cutoff date, was December 31, 1971. Changes to the language after that date were considered for inclusion only where they were in response to X3J4 proposals or where they affected items whose final disposition had been deferred by X3J4 pending specific PLC action."

Evolution in COBOL is shown in Section 2.2 of the JOD. Each modification began as a written proposal whose format and content are shown in the Acknowledgment on Page iii of the JOD. Every written proposal to PLC is discussed and voted upon. The acceptance of a proposal is determined by a rather sophisticated voting procedure based upon a percentage of members present at the meeting. In contrast to the ANSC X3J4 committee's eight criteria for accepting a proposed change (whether it be a minor or a radical change to the language), no such published criteria exist for CODASYL's PLC. This points out the contrasting philosophy in the COBOL language development committee vs. the COBOL language standards committee. Whereas the development group feels a need to be unconstrained in keeping COBOL a growing "living" language, the standards group has an obligation to maintain a standard COBOL specification against which a particular implementation can be measured (validated) for conformance. With this, a user, such as the Federal Government, can insure transportability across different computer systems. The effective lifespan of a COBOL application program is a very important aspect in judging the cost effectiveness of that application program written in the COBOL language as opposed to other possible computer languages.

In answer to the question of "foreseeable" future language specification for COBOL, as long as the COBOL language will continue to evolve to try to meet the changing needs of the business data processing community (as determined by CODASYL's PLC), no "final" specification is possible. As for the revision to COBOL '74, since the Standard COBOL is based largely upon a reference JOD, rather significant changes can be expected again as with the revision of COBOL '68. These changes require a certain amount of conversion for both the vendor's compilers and the user's application programs.

## CONVERSION IMPACTS

COBOL '74 will be expensive to implement for both the vendors and users. Exactly how expensive is yet to be determined. Actual cost should be calculated, at least after the fact, and given to ANSC X3J4 as reference data for the next COBOL Standard. Translators need to be developed and provided to the whole COBOL community as a tool to aid the overall conversion. The users must see that this product is shared by all, especially in the Federal Community, since all

taxpayers will pay for the higher conversion cost if they are not willing to cooperate with each other. One solution to generalized conversion can be seen in the paper describing the "System for Efficient Program Portability"[9].

## RECOMMENDATIONS

There remains a need to define the COBOL language formally. Both the syntax and semantics should be expressed as formal grammars and defined in an appropriate metalanguage. This would aid in pinpointing the ambiguities in the language and leave no doubt as to what is valid and what is not. This would be a great help in measuring conformance to the COBOL language (as described in the JOD) and to the COBOL Standard (COBOL '74).

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-Progam Communication and Debug modules). Furthermore, old modules have been enlarged and improved. CODASYL PLC and ANSC X3J4 deserve tremendous praise for their efforts. This paper suggests that users and vendors unite in helping to honestly evaluate and improve COBOL '74. When problems arise in attempting to implement COBOL '74, those problems should be brought to the attention of the entire COBOL community.

## REFERENCES


1. Baird, G. N., "The DOD Compiler Validation System," Proc. 1972 FJCC, AFIPS Press, Volume 41, Pages 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 Standards Institute Incorporated, New York, 1968.

4. CODASYL COBOL Journal of Development - 1970, published 1971.

5. Federal Information Processing Standards Publication 21-1, U. S. Government Printing Office, Washington, D. C., (pending).

6. Cook, et al, "An Overview of the 1974 COBOL Standard," Proc. 1975 NCC, AFIPS Press.

7. CODASYL Programming Language Committee Item 73007, The COBOL Data Base Facility, November 25, 1974.

8. *Aids for COBOL Programs Conversion*, NBS Handbook, U. S. Government Printing Office, Washington, D. C. (pending).

9. Baird, G. N. and Johnson, L. A., "System for Efficient Program Portability," Proc. 1974 NCC, AFIPS Press.

| BIBLIOGRAPHIC DATA SHEET | 1. Report No. FCCTS/TR-77/11 | 2. | 3. Recipient's Accession No. |
|---|---|---|---|
| 4. Title and Subtitle: The Standardization and Evolution of the COBOL Language | | | 5. Report Date: 9 May 1977 |
| | | | 6. |
| 7. Author(s): Patrick M. Hoyt, George N. Baird, Margaret M. Cook, William M. Holmes, L. Arnold Johnson, and Paul Oliver | | | 8. Performing Organization Rept. No. |
| 9. Performing Organization Name and Address: Software Development Division, ADPE Selection Office, Department of the Navy, Washington, D. C. 20376 | | | 10. Project/Task/Work Unit No. |
| | | | 11. Contract/Grant No. |
| 12. Sponsoring Organization Name and Address: ADPE Selection Office, Department of the Navy, Washington, D. C. 20376 | | | 13. Type of Report & Period Covered |
| | | | 14. |

15. Supplementary Notes

16. Abstracts

Government-wide testing of COBOL compilers is the responsibility of the Federal COBOL Compiler Testing Service, an activity of the Department of the Navy's Automatic Data Processing Equipment Selection Office, Software Development Division. In May 1974, the American National Standards Institute approved ANS Programming Language COBOL, X3.23-1974 as the national standard for the COBOL language. Specifications for this revision were drawn from USA Standard COBOL X3.23-1968 and the CODASYL COBOL Journal of Development, dated 31 December 1971. As a result, the Testing Service is engaged in the development of a COBOL Compiler Validation System for this new Standard, incorporating tests for the revised language into the existing 1968 COBOL Compiler Validation System. The first part of the paper focuses on the rationale behind changes from the '68 to '74 COBOL Standard as well as highlighting the new features in the language. The second part of the paper discusses evolution within the COBOL language itself with respect to the CODASYL COBOL Journal of Development and several other CODASYL language activities.

17b. Identifiers/Open-Ended Terms

408 438

17c. COSATI Field/Group 09/02

| 18. Availability Statement | 19. Security Class (This Report) | 21. No. of Pages |
|---|---|---|
| Unlimited | UNCLASSIFIED | 5 |
| | 20. Security Class (This Page) UNCLASSIFIED | 22. Price |