Apple III

COBOL

Language Reference Manual Volume 2

Customer Satisfaction

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable diskettes and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

Limitation on Warranties and Liability

Even though Apple has tested the software described in this manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is", and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

This manual and the software (computer programs) described in it are copyrighted by Apple or by Apple's software suppliers, with all rights reserved. Under the copyright laws, this manual or the programs may not be copied, in whole or part, without the written consent of Apple, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given or loaned to another person. Under the law, copying includes translating into another language.

You may use the software on any computer owned by you but extra copies cannot be made for this purpose. For some products, a multi-use license may be purchased to allow the software to be used on more than one computer owned by the purchaser, including a shared-disk system. (Contact your authorized Apple dealer for information on multi-use licenses.)

Product Revisions

Apple cannot guarantee that you will receive notice of a revision to the software described in this manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple Dealer.

© Micro Focus, Inc. 1978, 1982
1860 Embarcadero Road
Palo Alto, CA 94303

© Apple Computer, Inc. 1982
20525 Mariani Avenue
Cupertino, California 95014

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the U.S.A and Canada.

Reorder Apple Product #A3D0021

Apple III COBOL

Language Reference Manual

Volume 2

ACKNOWLEDGEMENTS

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection herewith.

The authors and copyright holders of the copyrighted material used herein:

FLOW-MATIC (Trademark for Sperry Rand Corporation) Programming for the Univac(R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Contents

Volume 1

Chapter 3

The Nucleus 47

Chapter 6

Relative Input and Output 197

Chapter 7

Indexed Input and Output

Chapter 10

Library 25

Chapter 11

Debug and Debugging with the Animation Option 27

Chapter 12

Interprogram Communication 33

Chapter 13

Communication 41

Chapter 8

Sort-Merge

INTRODUCTION TO THE SORT-MERGE MODULE

The Sort-Merge module provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each record. Optionally, a user may apply some special processing to each of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the SORT or after the records have been combined by the MERGE.

RELATIONSHIP WITH SEQUENTIAL I-O MODULE

The files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described implicitly or explicity in the FILE-CONTROL paragraph as having sequential organization. No input-output statement may be executed for the file named in the sort-merge file description.

ENVIRONMENT DIVISION IN THE SORT-MERGE MODULE

INPUT-OUTPUT SECTION

The FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format

```
FILE CONTROL. {file-control-entry}      ...
```

The File Control Entry

Function

The file control entry names a sort or merge file and specifies the association of the file to a storage medium.

General Format

```
SELECT file-name

    ASSIGN TO implementor-name-1   [, implementor-name-2] ... .
```

Syntax Rules

1. Each sort or merge file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph. Each sort or merge file specified in the file control entry must have a sort-merge file description entry in the Data Division.

2. Since file-name represents a sort or merge file, only the ASSIGN clause is permitted to follow file-name in the FILE-CONTROL paragraph.

General Rule

The ASSIGN clause specifies the association of the sort or merge file referenced by file-name to a storage medium.

The I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph specifies the memory area which is to be shared by different files.

General Format

```
I-O-CONTROL
[; SAME { RECORD     } AREA FOR file-name-1
        { SORT       }
        { SORT-MERGE }
                , file-name-2 ... ]  ... .
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

3. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.

4. The three formats of the SAME clause (SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are considered separately in the following:

   More than one SAME clause may be included in a program, however:

   a. A file-name must not appear in more than one SAME RECORD AREA clause.

b. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

c. If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE AREA clauses(s).

5. The files referenced in the SAME SORT AREA, SAME SORT-MERGE AREA, or SAME RECORD AREA clause need not all have the same organization or access.

General Rules

1. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to implicit redefinition of the area, i.e., records are aligned on the leftmost character position.

2. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause. This clause specifies that storage is shared as follows:

   a. The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies a memory area which will be made available for use in sorting or merging each sort or merge file named. Thus any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

   b. In addition, storage areas assigned to files that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in the SAME SORT AREA or SAME SORT-MERGE AREA clause. The extent of such allocation will be specified by the implementor.

   c. Files other than sort or merge files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, he must also include in the program a SAME AREA or SAME RECORD AREA clause naming these files.

   d. During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any non sort-merge files named in this clause must not be open.

DATA DIVISION IN THE SORT-MERGE MODULE

FILE SECTION

An SF file description gives information about the size and the names of the data records associated with the file to be sorted or merged. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements.

THE SORT-MERGE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

Function

The sort-merge file description furnishes information concerning the physical structure, identification and record names of the file to be sorted or merged.

General Format

```
SD   file-name

     [; RECORD CONTAINS  [integer-1 TO]  integer-2 CHARACTERS]
     [; DATA {RECORD IS  } data-name-1  [, data-name-2] ... ]  .
             {RECORDS ARE}
```

Syntax Rules

1. The level indicator SD identifies the beginning of the sort-merge file description and must precede the file-name.

2. The clauses which follow the name of the file are optional and their order of appearance is immaterial.

3. One or more record description entries must follow the sort-merge file description entry, however, no input-output statements may be executed for this file.

THE DATA RECORDS CLAUSE

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

General Format

```
DATA {RECORD IS  }  data-name-1  [, data-name-2]  ...
     {RECORDS ARE}
```

Syntax Rule

Data-name-1 and data-name-2 are the names of data records and must have 01 level-number record descriptions, with the same names, associated with them.

General Rules

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

THE RECORD CONTAINS CLAUSE

Function

The RECORD CONTAINS clause specifies the size of data records.

General Format

RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

General Rule

The size of each data record is completely defined within the record description entry, therefore this clause is never required. When present, however, the following notes apply:

a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data records and the maximum number of characters in the largest size data records, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record; see Selection of Character Representation and Radix in Chapter 2; and THE SYNCHRONIZED CLAUSE and THE USAGE CLAUSE in Chapter 3.

PROCEDURE DIVISION IN THE SORT-MERGE MODULE

THE MERGE STATEMENT

Function

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merge order, to an output procedure or to an output file.

General Format

```
MERGE file-name-1 ON { ASCENDING  } KEY data-name-1 [, data-name-2] ...
                     { DESCENDING }

    [ON { ASCENDING  } KEY data-name-3 [, data-name-4] ...] ...
        { DESCENDING }

    [COLLATING SEQUENCE IS alphabet-name]

    USING file-name-2, file-name-3 [, file-name-4] ...

    { OUTPUT PROCEDURE IS section-name-1 [{ THROUGH } section-name-2] }
    {                                    { THRU    }                  }
    { GIVING file-name-5                                              }
```

Syntax Rules

1. File-name-1 must be described in a sort-merge file description entry in the Data Divison.

2. Section-name-1 represents the name of an output procedure.

3. File-name-2, file-name-3, file-name-4, and file-name-5 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, file-name-3, file-name-4, and file-name-5 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause an equal number of character positions to be allocated for the corresponding records.

4. The words THRU and THROUGH are equivalent.

5. Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable length items.

   d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

   e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

6. No more than one file-name from a multiple file reel can appear in the MERGE statement.

7. File-names must not be repeated within the MERGE statement.

8. MERGE statements may appear anywhere except in the declaratives portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.

General Rules

1. The MERGE statement will merge all records contained on file-name-2, file-name-3, and file-name-4. The files referenced in the MERGE statement must not be open at the time the MERGE statement is executed. These files are automatically opened and closed by the merge operation with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file.

2. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, etc.

   a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rule for comparison of operands in a relation condition.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

   b. Second, the collating sequence established as the program collating sequence.

4. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form part of any other procedure. In order to make merged records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time in merge order, from file-name-1. The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must not contain any transfers of control to points outside the output procedure; ALTER, GO TO and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

   b. The output procedures must not contain any SORT or MERGE statements.

   c. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedures; ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedures.

5. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable procedure. The merge procedure has then reached a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

6. Segmentation, as defined in Chapter 9, can be applied to programs containing the MERGE statement. However, the following restrictions apply:

   a. If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

      * Totally within non-independent segments, or

      * Wholly contained in a single independent segment.

   b. If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

      * Totally within non-independent segments, or

      * Wholly within the same independent segment as that MERGE statement.

7. If the GIVING phrase is specified, all the merged records in file-name-1 are automatically written on file-name-5 as the implied output procedure for this MERGE statement.

8. In the case of equal compare, according to the rules for comparison of operands in a relation condition, on the contents of the data items identified by all the KEY data-names between records from two or more input files (file-name-2, file-name-3, file-name-4, ...), the records are written on file-name-5 or returned to the output procedure, depending on the phrase specified, in the order that the associated input files are specified in the MERGE statement.

9. The results of the merge operation are predictable only when the records in the files referenced by file-name-2, file-name-3, ..., are ordered as described in the ASCENDING or DESCENDING KEY clause associated with the MERGE statement.

THE RELEASE STATEMENT

Function

The RELEASE statement transfers records to the initial phase of a SORT operation.

General Format

RELEASE record-name [FROM identifier]

Syntax Rules

1. A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement for a file whose sort-merge file description entry contains record-name. (See The SORT Statement.)

2. Record-name must be the name of a logical record in the associated sort-merge file description entry and may be qualified.

3. Record-name and identifier must not refer to the same storage area.

General Rules

1. The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.

2. If the FROM phrase is used, the contents of the identifier data area are moved to record-name, then the contents of record-name are released to the sort file. Moving files takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The information in the record area is no longer available, but the information in the data area associated with identifier is available.

3. After the execution of the RELEASE statement, the logical record is no longer available in the record area unless the associated sort-merge file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated sort-merge file, as well as to the file associated with record-name. When control passes from the input procedure, the file consists of all those records which were placed in it by the execution of RELEASE statements.

THE RETURN STATEMENT

Function

The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records during a MERGE operation.

General Format

```
RETURN file-name RECORD [INTO identifier]
                ; AT END imperative-statement
```

Syntax Rules

1. File-name must be described by a sort-merge file description entry in the Data Division.

2. A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for file-name.

3. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

General Rules

1. When the logical records of a file are described with more than one record descritpion, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

2. The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT or MERGE statement, to be made available for processing in the record areas associated with the sort or merge file.

3. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

4. When the INTO phrase is used, the data is available in both the input record area and the data area associated with identifier.

5. If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs. The contents of the record areas associated with the file when the AT END condition occurs are undefined. After the execution of the imperative-statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

THE SORT STATEMENT

Function

The SORT statement creates a sort file by executing input procedures or by transferring records from another file, sorts the records in the sort file on a set of specified keys, and in the final phase of the sort operation, makes available each record from the sort file, in sorted order to some output procedures or to an output file.

General Format

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1  [, data-name-2] ...
                    { DESCENDING }

                  [ON { ASCENDING  } KEY data-name-3,  [, data-name-4] ...] ...
                      { DESCENDING }

[COLLATING SEQUENCE IS alphabet-name]

{ INPUT PROCEDURE IS section-name-1 [{ THROUGH } section-name-2] }
{                                   { THRU    }                  }
{ USING file-name-2  [, file-name-3] ...                         }
{ OUTPUT PROCEDURE IS section-name-3 [{ THROUGH } section-name-4] }
{                                    { THRU    }                  }
{ GIVING file-name-4                                             }
```

Syntax Rules

1. File-name-1 must be described in a sort-merge file description entry in the Data Division.

2. Section-name-1 represents the name of an input procedure. Section-name-3 represents the name of an output procedure.

3. File-name-2, file-name-3 and file-name-4 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, file-name-3 and file-name-4 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause equal amounts of character positions to be allocated for the corresponding records.

4. Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable length items.

   d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

   e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

5. The words THRU and THROUGH are equivalent.

6. SORT statements may appear anywhere except in the declaratives portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.

7. No more than one file-name from a multiple file reel can appear in the SORT statement.

General Rules

1. The Procedure Division may contain more than one SORT statement appearing anywhere except:

   a. In the declaratives portion, or

   b. In the input and output procedures associated with a SORT or MERGE statement.

2. The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, etc.

   a. When the ASCENDING phrase is specified, the sorted sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

   b. When the DESCENDING phrase is specified, the sorted sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

   b. Second, the collating sequence established as the program collating sequence.

4. The input procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. In order to transfer records to the file referenced by file-name-1, the input procedure must include the execution of at least one RELEASE statement. Control must not be passed to the input procedure when a related SORT statement is being executed. The input procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

   a. The input procedure must not contain any SORT or MERGE statements.

   b. The input procedure must not contain any explicit transfers of control to points outside the input procedure; ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

   c. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

5. If an input procedure is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

6. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form part of any input procedure. In order to make sorted records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure may consist of any procedures needed to select, modify or copy the records that are being returned, one at a time in sorted order, from the sort file. The restrictions on the procedural statements within the output procedure are as follows:

a. The output procedure must not contain any SORT or MERGE statements.

b. The output procedure must not contain any explicit transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

c. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

7. If an output procedure is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

8. Segmentation as defined in Chapter 9 can be applied to programs containing the SORT statement. However, the following restrictions apply:

   a. If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

      * Totally within non-independent segments, or

      * Wholly contained in a single independent segment.

   b. If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

      * Totally within non-independent segments, or

      * Wholly within the same independent segment as that SORT statement.

9. If the USING phrase is specified, all the records in file-name-2 and file-name-3 are transferred automatically to file-name-1. At the times of execution of the SORT statement, file-name-2 and file-name-3 must

not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2 and file-name-3. These implicit functions are performed such that any associated USE procedures are executed. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 and file-name-3 to the file area for file-name-1 and the release of records to the initial phase of the sort operation.

10. If the GIVING phrase is specified, all the sorted records in file-name-1 are automatically written on file-name-4 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-4 must not be open. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of file-name-4. These implicit functions are performed such that any associated USE procedures are executed. The terminating function is performed as if a CLOSE statement, without optional phrases, had been executed for the file. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and the moving of the records from the file area for file-name-1 to the file area for file-name-4.

Chapter 9
Segmentation

INTRODUCTION TO THE SEGMENTATION MODULE

The Segmentation module provides a capability to specify object program overlay requirements.

Segmentation provides a facility for specifying permanent and independent segments. All sections with the same segment-number must be contiguous in the source program. All segments specified as permanent segments must be contiguous in the source program.

GENERAL DESCRIPTION OF SEGMENTATION

COBOL segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements.

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division is considered in determining segmentation requirements for an object program.

ORGANIZATION

Program Segments

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

Fixed Portion

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of fixed permanent segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by another independent segment. An independent

segment is in its initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

1. Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

2. Control is transferred explicitly to that segment from a segment with a different segment-number (with the exception noted in paragraph 2 below).

3. Control is transferred to that segment as the result of the implicit transfer of control between a SORT or MERGE statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

On subsequent transfer of control to the segment, an independent segment is in its last-used state when:

1. Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in paragraph 1).

2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

SEGMENTATION CLASSIFICATION

Sections which are to be segmented are classified, using a system of segment-numbers and the following criteria:

1. Logic Requirements - Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections which are used less frequently are normally classified as belonging to one of the independent segments, depending on logic requirements.

2. Frequency of Use - Generally, the more frequently a section is referred to, the lower its segment-number, the less frequently it is referred to, the higher its segment-number.

3. Relationship to Other Sections - Sections which frequently communicate with one another should be given the same segment-numbers

SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

STRUCTURE OF PROGRAM SEGMENTS

SEGMENT-NUMBERS

Section classification is accomplished by means of a system of segment-numbers. The segment-number is included in the section header.

GENERAL FORMAT

```
section-name SECTION [segment-number]
```

SYNTAX RULES

1. The segment-number must be an integer ranging in value from 0 through 99.

2. If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

3. Sections in the declaratives must contain segment-numbers less than 50.

GENERAL RULES

1. All sections which have the same segment-number constitute a program segment. All sections which have the same segment-number must be together in the source program.

2. Segments with segment-number 0 through 49 belong to the fixed portion of the object program.

3. Segments with segment-number 50 through 99 are independent segments.

RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER and PERFORM statement.

THE ALTER STATEMENT

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

THE PERFORM STATEMENT

A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

* Sections and/or paragraphs wholly contained in one or more non-independent segments.

* Sections and/or paragraph wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

EXTRA INTERMEDIATE CODE FILES

When segmentation is used, extra intermediate code files are generated by the compiler as follows:

filename.Inn	-	Intermediate code files one for each independent segment
filename.ISR	-	Inter-Segment Reference table one per segmented program
filename.Dnn	-	Dictionary files one for each independent segment except the last

where:

filename is the name without the extension of the principal intermediate code file

nn is a segment number that identifies the particular segment

NOTE:

The filename.Dnn files are written and used solely by the compiler, and need not be retained after compilation. The filename.Inn files and the filename.ISR file must be retained as part of the object program and must also be copied when the program is copied.

THE MERGE STATEMENT

If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

a. Totally within non-independent segments, or

b. Wholly contained in a single independent segment.

If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

a. Totally within non-independent segments, of

b. Wholly within the same independent segment as that MERGE statement.

THE SORT STATEMENT

If a SORT statement appears in a section that is not an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

a. Totally within non-independent segments, or

b. Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

a. Totally within non-independent segments, or

b. Wholly within the same independent segment as that SORT statement.

Chapter 10
Library

INTRODUCTION TO THE LIBRARY MODULE

The Library module provides a capability for specifying text that is to be copied from a source user-library file. This is usually created using any suitable source text editor

Apple III COBOL libraries consist of disk files that contain source to be made available to the compiler. The effect of the interpretation of the COPY statement is to insert text into the source program, where it will be treated by the compiler as part of the source program.

THE COPY STATEMENT

FUNCTION

The COPY statement incorporates text into an Apple III COBOL source program.

GENERAL FORMAT

```
COPY "text-name".
```

SYNTAX RULES

1. Text-name must be a unique standard operating system file name.

2. The COPY statement must be preceded by a space and terminated by the separator period.

3. A COPY statement may occur in the source program anywhere a character-string or a separator may occur except that a COPY statement must not occur within a COPY statement.

GENERAL RULES

1. The compilation of a source program containing COPY statement is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.

2. The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive.

3. The library text is copied unchanged.

4. If the unit identifier is not explicitly specified, default is to the drive from which the compiler is loaded.

Chapter 11

Debug and Debugging with the Animation Option

INTRODUCTION

Standard ANSI COBOL debugging provides a means by which the user can describe the conditions under which procedures are to be monitored during the execution of the object program.

The Apple III COBOL Animation Option is an extension to ANSI COBOL that provides interactive debugging facilities in the user's program at the COBOL level. Programs may be run from the start with or without the Animation Option invoked until a specified break-point is reached, when control is passed back to the user. At this point, data areas and program procedures may be inspected or changed.

APPLE III COBOL ANIMATION OPTION

The Animation Option is entered as an option by the user and the user program is then monitored and tested line by line, paragraph by paragraph and so on as required. The commands to the package can reference procedure statements and data areas. Powerful macros of commands can be used to give very sophisticated debugging facilities. The precise details for using the Animation Option are contained in the Apple III COBOL Introduction and Operating System Manual.

STANDARD ANSI COBOL DEBUG

The decisions of what to monitor and what information to display are explicitly in the domain of the user. The COBOL Debug facility simply provides a convenient access to pertinent information.

The features of the language that support the COBOL Debug module are:

* A compile time switch -- WITH DEBUGGING MODE.
* An object time switch.
* A USE FOR DEBUGGING statement.
* A special register -- DEBUG-ITEM.
* Debugging lines.

The reserved word DEBUG-ITEM is the name for a special register generated automatically by the compiler that supports the debugging facility. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DUBUG-ITEM are also reserved words.

COMPILE-TIME SWITCH

The DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph in the Environment Division. It serves as a compile-time switch over debugging statements written in the program.

When DEBUGGING MODE is not specified in a program, all the debugging lines are compiled as if they were comment lines and their syntax is not checked.

COBOL DEBUG OBJECT TIME SWITCH

An object time switch dynamically activates the debugging code inserted by the compiler. This switch cannot be addressed in the program; it is controlled outside the COBOL environment. If the switch is 'on', the effects of any USE FOR DEBUGGING statements written in the source program are permitted. If the switch is 'off', all the effects described in the USE FOR DEBUGGING Statement, are inhibited. Recompilation of the source program is not required to provide or take away this facility.

The object time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program at compile time.

The switch is described in the Apple III COBOL Introduction and Operating System Manual.

ENVIRONMENT DIVISION IN COBOL DEBUG

The WITH DEBUGGING MODE Clause

Function

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be compiled. If this clause is not specified, all debugging lines and sections are compiled as if they were comment lines.

General Format

SOURCE-COMPUTER. computer-name [WITH DEBUGGING MODE].

General Rules

1. If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, all USE FOR DEBUGGING statements and all debugging lines are compiled.

2. If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, any USE FOR DEBUGGING statements and all associated debugging sections, and any debugging lines are compiled as if they were comment lines.

PROCEDURE DIVISION IN COBOL DEBUG

The USE FOR DEBUGGING Statement

Function

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

General Format

```
section-name SECTION [segment number].

USE FOR DEBUGGING ON { procedure-name-1  }
                     { ALL PROCEDURES    }

   [ procedure-name-2  ]
   [ 'ALL PROCEDURES   ]
```

Syntax Rules

1. Debugging section(s), if specified, must appear together immediately after the DECLARATIVES header.

2. Except in the USE FOR DEBUGGING statement itself, there must be no reference to any non-declarative procedure within the debugging section.

3. Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections.

4. Except for the USE FOR DEBUGGING statement itself, statements appearing within a given debugging section may reference procedure-names defined within a different USE procedure only with a PERFORM statement.

5. Procedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.

6. Any given procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.

7. The ALL PROCEDURES phrase can appear only once in a program.

8. When the ALL PROCEDURES phrase is specified, procedure-name-1, procedure-name-2, ... must not be specified in any USE FOR DEBUGGING statement.

9. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

General Rules

1. In the following general rules all references to procedure-name-1 apply equally to procedure-name-2.

2. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

3. When procedure-name-1 is specified in a USE FOR DEBUGGING statement that debugging section is executed:

   a. Immediately before each execution of the named procedure;
   b. Immediately after the execution of an ALTER statement which references procedure-name-1.

4. The ALL PROCEDURES phrase causes the effects described in general rule 3 to occur for every procedure-name in the program, except those appearing within a debugging section.

5. The associated debugging section is not executed for a specific operand more than once as a result of the execution of a single statement, regardless of the number of times that operand is explicitly specified. In the case of a PERFORM statement which caused iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

   Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

6. A reference to procedure-name-1 as a qualifier does not constitute reference to that item for the debugging described in the general rules above.

7. Associated with each execution of a debugging section is the special register DEBUG-ITEM, which provides information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```
01  DEBUG-ITEM.
    02  DEBUG-LINE      PICTURE IS X(6).
    02  FILLER          PICTURE IS X VALUE SPACE.
    02  DEBUG-NAME      PICTURE IS X(30).
    02  FILLER          PICTURE IS X VALUE SPACE.
    02  DEBUG-SUB-1     PICTURE IS S9999 SIGN IS LEADING SEPARATE
                         CHARACTER.
    02  FILLER          PICTURE IS X VALUE SPACE
    02  DEBUG-SUB-2     PICTURE IS S9999 SIGN IS LEADING SEPARATE
                         CHARACTER.
    02  FILLER          PICTURE IS X VALUE SPACE.
    02  DEBUG-SUB-3     PICTURE IS S9999 SIGN IS LEADING SEPARATE
                         CHARACTER.
    02  FILLER          PICTURE IS X VALUE SPACE.
    02  DEBUG-CONTENTS  PICTURE IS X(n).
```

8. Prior to each execution of a debugging section, the contents of the data item referenced by DEBUG-ITEM are space-filled. The contents of data items subordinate to DEBUG-ITEM are then updated, according to the following general rules, immediately before control is passed to that debugging section. The contents of any data item not specified in the following general rules remain spaces.

   Updating is accomplished in accordance with the rules for the MOVE statement, the sole exception being the move to DEBUG-CONTENTS, when the move is treated exactly as if it was an alphanumeric to alphanumeric elementary move with no conversion of data from one form of internal representation to another.

9. The contents of DEBUG-LINE is the relevant COBOL source line number. This provides the means of identifying a particular source statement.

10. DEBUG-NAME contains the first 30 characters of the name that caused the debugging section to be executed.

    All qualifiers of the name are separated in DEBUG-NAME by the word 'IN' or 'OF'. Subscripts/indices, if any, are not entered into DEBUG-NAME.

11. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following general rules.

12. If the first execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the first statement of that procedure.

    b. DEBUG-NAME contains the name of that procedure.

    c. DEBUG-CONTENTS contains 'START PROGRAM'.

13. If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.

14. If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

15. If the transfer to control from the control mechanism associated with a PERFORM statement causes the debugging section associated with procedure-name-1 to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the PERFORM statement that references procedure-name 1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'PERFORM LOOP'.

16. If procedure-name-1 is a USE procedure that is to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the statement that causes execution of the USE procedure.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'USE PROCEDURE'.

17. If an implicit transfer of control from the previous sequential paragraph to procedure-name-1 causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the previous statement.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'FALL THROUGH'.

DEBUGGING LINES

A debugging line is any line with a 'D' in the indicator area of the line. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.

The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

A debugging line will be considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain a 'D' in the indicator area, and character-strings may not be broken across two lines.

A debugging line is only permitted in the program after the OBJECT-COMPUTER paragraph.

Chapter 12
Interprogram Communication

INTRODUCTION TO THE INTER-PROGRAM COMMUNICATION MODULE

The Inter-Program Communication module provides a facility by which a program can communicate with one or more programs. This provides a programmer with a modular programming capability. Each module when CALLed is loaded dynamically by the Run Time System. Communication is provided by:

* The ability to transfer control from one program to another within a run unit

* The ability for both programs to have access to the same data items.

DATA DIVISION IN THE INTER-PROGRAM COMMUNICATION MODULE

LINKAGE SECTION

The Linkage Section in a program is meaningful if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. No space is allocated in the program for data items referenced by data-names in the Linkage Section of that program. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of the called program only if they are specified as operands of the USING phrase of the Procedure Division header or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous data items and/or record description entries.

Each Linkage Section record-name and noncontiguous item name must be unique within the called program since it cannot be qualified. Data items defined in the Linkage Section of the called program must not be associated with data items defined in the Report Section of the calling program.

Of those items defined in the Linkage Section only data-name-1, data-name-2, ... in the USING phrase of the Procedure Division header, data items subordinate to these data-names, and condition-names and/or index-names associated with such data-names and/or subordinate data items, may be referenced in the Procedure Division.

Noncontiguous Linkage Storage

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these data items is defined in a separate data description entry which begins with the special level-number 77.

The following data clauses are required in each data description entry:

* Level-number 77
* Data-name
* The PICTURE clause or the USAGE IS INDEX clause

Other data description clauses are optional and can be used to complete the description of the item if necessary.

Linkage Records

Data elements in the Linkage Section which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause which is used in an input or output record description can be used in a Linkage Section.

Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

PROCEDURE DIVISION IN THE INTER-PROGRAM COMMUNICATION MODULE

THE PROCEDURE DIVISION HEADER

The Procedure Division is identified by and must begin with the following header:

PROCEDURE DIVISION [USING data-name-1 [, data-name-2] ...]

The USING phrase is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

Each of the operands in the USING phrase of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

Within a called program, Linkage Section data items are processed according to their data descriptions given in the called program.

When the USING phrase is present, the object program operates as if data-name-1 of the Procedure Division header in the called program and data-name-1 in the USING phrase of the CALL statement in the calling program refer to a single set of data that is equally available to both the called and calling programs. Their descriptions must define an equal number of character positions; however they need not be the same name. In like manner, there is an equivalent relationship between data-name-2, ..., in the USING phrase of the called program and data-name-2, ..., in the USING phrase of the CALL statement in the calling program. A data-name must not appear more than once in the USING phrase in the Procedure Division header of the called program; however, a given data-name may appear more than once in the same USING phrase of a CALL statement.

If the USING phrase is specified, the INITIAL clause must not be present in any CD entry.

THE CALL STATEMENT

Function

The CALL statement causes control to be transferred from one object program to another, within the run unit.

General Format

Format 1

```
CALL    {identifier-1}  [USING data-name-1    [, data-name-2] ...]
        {literal-1   }
     [ON OVERFLOW imperative-statement]
```

Format 2

```
CALL    {literal-2   }  [USING data-name-3    [, data-name-4] ...]
        {identifier-2}
```

Syntax Rules

1. Literal-1 must be a nonnumeric literal.

2. Identifier-1 must be defined as an alphanumeric data item usage display.

3. The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program and the number of operands in each USING phrase must be identical.

4. Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, Communication Section,or Linkage Section, and must have a level-number of 01 or 77. Data-name-1, data-name-2, ..., may be qualified when they reference data items defined in the File Section or the Communication Section.

5. Literal-2 must be defined as a non-numeric literal.

6. Identifier-2 must be defined as an alphanumeric data item with a numeric value, e.g. CALL "3" or CALL D-NAM where D-NAM is defined as class alphanumeric, and usage display, containing a numeric value.

General Rules

1. The program whose name is specified by the value of literal-1 or identifier-1 is a called intermediate code module, literal-2 is a called run time subroutine; the program in which the CALL statement appears is the calling program.

2. The execution of a CALL statement causes control to pass to the called program.

3. In format 1, a called intermediate code module is loaded from disk the first time it is called within a run-unit and the first time it is called after a CANCEL to the called program.

   On all other entries into the called program, the state of the program remains unchanged from its state when last exited. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

4. In format 2, a called run time subroutine is always in the state in which it last exited.

5. If during the execution of a CALL statement, it is determined that the available portion of run-time memory is incapable of accomodating the program specified in the CALL statement, the next sequential instruction is executed. If ON OVERFLOW has been specified, the associated imperative statement is executed before the next instruction is executed.

6. Called programs may contain CALL statements. However, a called program must not contain a call statement that directly or indirectly calls the calling program.

7. The data-names, specified by the USING phrase of the CALL statement, indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the data-names in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Corresponding data-names refer to a single set of data which is available to the called and calling program. The correspondence is positional, not by name. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

8. The CALL statement may appear anywhere within a segmented program. Therefore, when a CALL statement appears in a section with a segment-number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement returns control to the calling program.

THE CANCEL STATEMENT

Function

The CANCEL statement releases the memory areas occupied by the referred to program.

General Format

```
CANCEL  { identifier-1 }  [ { , identifier-2 } ]
        { literal-1    }  [ {   literal-2    } ]
```

Syntax Rules

1. Literal-1, literal-2, ..., must each be a nonnumeric literal.

2. Identifier-1, identifier-2, ..., must each be defined as an alphanumeric data item such that its value can be a program name.

General Rules

1. Subsequent to the execution of a CANCEL statement, the program referred to therein ceases to have any logical relationship to the run unit in which the CANCEL statement appears. A subsequently executed CALL statement naming the same program will result in that program being initiated in its initial state. The memory areas associated with the named programs are released so as to be made available for disposition by the operating system.

2. A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.

3. A logical relationship to a cancelled subprogram is established only by execution of a subsequent call statement.

4. A called program is cancelled either by being referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

5. No action is taken when a CANCEL statement is executed naming a program that has not been called in this run unit or has been called and is at present cancelled. Control passes to the next statement.

THE EXIT PROGRAM STATEMENT

Function

The EXIT PROGRAM statement marks the logical end of a called program.

General Format

```
EXIT PROGRAM
```

Syntax Rules

1. The EXIT PROGRAM statement must appear in a sentence by itself.

2. The EXIT PROGRAM sentence must be the only sentence in the paragraph.

General Rule

An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program which is not called behaves as if the statement were an EXIT statement. (See THE EXIT STATEMENT in Chapter 3.)

Chapter 13
Communication

INTRODUCTION TO THE COMMUNICATION MODULE

FUNCTION

The Communication module provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System (MCS) with local and remote communication devices.

DATA DIVISION IN THE COMMUNICATION MODULE

COMMUNICATION SECTION

In a COBOL program the communication description entries (CD) represent the highest level of organization in the Communication Section. The Communication Section header is followed by a communication description entry consisting of a level indicator (CD), a data-name and a series of independent clauses. These clauses indicate the queues and sub-queues, the message date and time, the source, the text length, the status and end keys, and message count of input. These clauses specify the destination count, the text length, the status and error keys, and destinations for output. The entry itself is terminated by a period. These record areas may be implicitly redefined by user-specified record description entries following the various communication description clauses.

THE COMMUNICATION DESCRIPTION - COMPLETE ENTRY SKELETON

Function

The communication description specifies the interface area between the MCS and a COBOL program.

General Format

Format 1:

```
CD  cd-name;

                                  [; SYMBOLIC QUEUE IS data-name-1]

                                       [; SYMBOLIC SUB-QUEUE-1 IS data-name-2]
                                       [; SYMBOLIC SUB-QUEUE-2 IS data-name-3
                                       [; SYMBOLIC SUB-QUEUE-3 IS data-name-4]
                                       [; MESSAGE DATE IS data-name-5]
FOR  [INITIAL]  INPUT                  [; MESSAGE TIME IS data-name-6]
                                       [; SYMBOLIC SOURCE IS data-name-7]
                                       [; TEXT LENGTH IS data-name-8]
                                       [; END KEY IS data-name-9]
                                       [; STATUS KEY IS data-name-10]
                                       [; MESSAGE COUNT IS data-name-11]

                                  [data-name-1, data-name-2, ..., data-name-11]
```

Format 2:

```
CD  cd-name; FOR OUTPUT

     [; DESTINATION COUNT IS  data-name-1]
     [; TEXT LENGTH IS data-name-2]
     [; STATUS KEY IS data-name-3]
     [; DESTINATION TABLE OCCURS integer-2 TIMES
          [; INDEXED BY index-name-2] ...
     [; ERROR KEY IS data-name-4]
     [; SYMBOLIC DESTINATION IS data-name-5].
```

Syntax Rules

Format 1:

1. A CD must appear only in the Communication Section.

2. Within a single program, the INITIAL clause may be specified in only one CD. The INITIAL clause must not be used in a program that specifies the USING phrase of the Procedure Division Header. (See The Procedure Division Header.)

3. Except for the INITIAL clause, the optional clauses may be written in any order.

4. If neither option in the format is specified, a level 01 data description entry must follow the CD description entry. Either option may be followed by a level 01 data description entry.

5. For each input CD, a record area of 87 contiguous standard data format characters is allocated. This record area is defined to the MCS as follows:

   a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1-12 in the record.

   b. The SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13-24 in the record.

   c. The SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25-36 in the record.

   d. The SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37-48 in the record.

   e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 49-54 in the record.

   f. The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of 8 digits without an operational sign occupying character positions 55-62 in the record.

   g. The SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63-74 in the record.

   h. The TEXT LENGTH clause defines data-name-8 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 75-78 in the record.

   i. The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of 1 character occupying position 79 in the record.

   j. The STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of 2 characters occupying positions 80-81 in the record.

   k. The MESSAGE COUNT clause defines data-name-11 as the name of an elementary data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 82-87 in the record.

The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clauses.

Use of either option results in a record whose implicit description is equivalent to the following:

```
                IMPLICIT DESCRIPTION                        COMMENT

01   data-name-0.

     02   data-name-1    PICTURE X(12).      SYMBOLIC QUEUE
     02   data-name-2    PICTURE X(12).      SYMBOLIC SUB-QUEUE-1
     02   data-name-3    PICTURE X(12).      SYMBOLIC SUB-QUEUE-2
     02   data-name-4    PICTURE X(12).      SYMBOLIC SUB-QUEUE-3
     02   data-name-5    PICTURE 9(06).      MESSAGE DATE
     02   data-name-6    PICTURE 9(08).      MESSAGE TIME
     02   data-name-7    PICTURE X(12).      SYMBOLIC SOURCE
     02   data-name-8    PICTURE 9(04).      TEXT LENGTH
     02   data-name-9    PICTURE X.          END KEY
     02   data-name-10   PICTURE XX.         STATUS KEY
     02   data-name-11   PICTURE 9(06).      MESSAGE COUNT
```

NOTE:

In the above, the information under 'COMMENT' is for clarification and is not part of the description.

6. Record description entries following an input CD implicitly redefine this record and must describe a record of exactly 87 characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. However, the MCS will always reference the record according to the data descriptions defined in syntax rule 5.

7. Data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

Format 2:

8. A CD must appear only in the Communication Section.

9. If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.

10. For each output CD, a record area of contiguous standard data format characters is allocated according to the following formula: (10 plus 13 times integer-2).

a. The DESTINATION COUNT clause defines data-name-1 as the name of a data item whose implicit description is that of an integer of four digits without an operational sign, occupying character positions 1-4 in the record.

b. The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of four digits without an operational sign, occupying character positions 5-8 in the record.

c. The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of 2 characters occupying positions 9-10 in the record.

d. Character positions 11-23 and every set of 13 characters thereafter will form table items of the following description:

   * The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of 1 character.

   * The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

Use of the above clauses results in a record whose implicit description is equivalent to the following:

```
     IMPLICIT DESCRIPTION                                  COMMENT

01   data-name-0.
     02   data-name-1     PICTURE 9(04).          DESTINATION COUNT
     02   data-name-2     PICTURE 9(04).          TEXT LENGTH
     02   data-name-3     PICTURE XX.             STATUS KEY
     02   data-name  OCCURS integer-2 TIMES.      DESTINATION TABLE

          03   data-name-4     PICTURE X.          ERROR KEY
          03   data-name-5     PICTURE X(12).      SYMBOLIC DESTINATION
```

NOTE:

In the above, the information under 'COMMENT' is for clarification and is not part of the description.

11. Record descriptions following an output CD implictly redefine this record. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. Note that the MCS will always reference the record according to the data descriptions defined in syntax rule 10.

12. Data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.

13. If the DESTINATION TABLE OCCURS clause is not specified, one ERROR KEY and one SYMBOLIC DESTINATION area is assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.

14. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and dataname-5 may only be referred to by subscripting or indexing.

15. There is no restriction on the value of the data item referenced by data-name-1 and integer-2.

General Rules

Format 1:

1. The input CD information constitutes the communication between the MCS and the program as information about the message being handled. This information does not come from the terminal as part of the message.

2. The contents of the data items referenced by data-name-2, data-name-3, and data-name-4, when not being used, must contain spaces.

3. The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 contain symbolic names designating queues, sub-queues, ..., respectively. All symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the MCS.

4. A RECEIVE statement causes the serial return of the 'next' message or a portion of a message from the queue as specified by the entries in the CD.

   If during the execution of a RECEIVE statement, a message from a more specific source is needed, the contents of the data item referenced by data-name-1 can be made more specific by the use of the contents of the data items referenced by data-name-2, data-name-3, and in turn data-name-4. When a given level of the queue structure is specified, all higher levels must also be specified.

   If fewer than all the levels of the queue hierarchy are specified, the MCD determines the 'next' message or portion of a message to be accessed.

   After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 will contain the symbolic names of all the levels of the queue structure.

5. Whenever a program is scheduled by the MCS to process a message, that program establishes a run unit and the symbolic names of the queue structure that demanded this activity will be placed in the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause as applicable. In all other cases, the contents of the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause are initialized to spaces.

   The symbolic names are inserted or the initialization to spaces is completed prior to the execution of the first Procedure Division statement.

   The execution of a subsequent RECEIVE statement naming the same contents of the data items referenced by data-name-1 through data-name-4 will return the actual message that caused the program to be scheduled. Only at that time will the remainder of the CD be updated.

6. If the MCS attempts to schedule a program lacking an INITIAL clause, the results are undefined.

7. Data-name-5 has the format 'YYMMDD' (year, month, day). Its contents represent the date on which the MCS recognizes that the message is complete.

   The contents of the data item referenced by data-name-5 are only updated by the MCS as part of the execution of a RECEIVE statement.

8. The contents of data-name-6 have the format 'HHMMSSTT' (hours, minutes, seconds, hundredths of a second) and its contents represent the time at which the MCS recognizes that the message is complete.

   The contents of the data item referenced by data-name-6 are only updated by the MCS as part of the execution of the RECEIVE statement.

9. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by data-name-7, the symbolic name of the communications terminal that is the source of the message being transferred. This symbolic name must follow the rules for the formation of system names. However, if the symbolic name of the communication terminal is not known to the MCS, the contents of the data item referenced by data-name-7 will contain spaces.

10. The MCS indicates via the contents of the data item referenced by data-name-8 the number of character positions filled as a result of the execution of the RECEIVE statement. (See THE RECEIVE STATEMENT later in this Chapter.)

11. The contents of the data item referenced by data-name-9 are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

a. When the RECEIVE MESSAGE phrase is specified, then data-name-9 is set to one of the following:

   * If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;

   * If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

   * If less than a message has been detected, the contents of the data item referenced by data-name-9 are set to 0.

b. When the RECEIVE SEGMENT phrase is specified, data-name-9 is set to one of the following:

   * If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;

   * If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

   * If an end of segment has been detected, the contents of the data item referenced by data-name-9 are set to 1;

   * If less than a message segment is transferred, the contents of the data item referenced by data-name-9 are set to 0.

c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by data-name-9.

12. The contents of the data item referenced by data-name-10 indicate the status condition of the previously executed RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT, or DISABLE INPUT statements.

    The actual association between the contents of the data item referenced by data-name-10 and the status condition itself is defined in Table 13-1.

13. The contents of the data item referenced by data-name-11 indicate the number of messages that exist in a queue, sub-queue-1, ... The MCS updates the contents of the data item referenced by data-name-11 only as part of the execution of an ACCEPT statement with the COUNT phrase.

Format 2:

14. The nature of the output CD information is such that it is not sent to the terminal, but constitutes the communication between the program and the MCS as information about the message being handled.

15. During the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the contents of the data item referenced by data-name-1 will indicate to the MCS the number of symbolic destinations that are to be used from the area referenced by data-name-5.

    The MCS finds the first symbolic destination in the first occurrence of the area referenced by data-name-5, the second symbolic destination in the second occurrence of the area referenced by data-name-5 ... ,m up to and including the occurrence of the area referenced by data-name-5 indicated by the contents of data-name-1.

    If during the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement the value of the data item referenced by data-name-1 is outside the range of 1 through integer-2, an error condition is indicated and the execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is terminated.

16. It is the responsibility of the user to insure that the value of the data item referenced by data-name-1 is valid at the time of execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

17. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 to be the user's indication of the number of leftmost character positions of the data item referenced by the associated SEND identififer from which data is to be transferred. (See THE SEND STATEMENT later in this Chapter.)

18. Each occurrence of the date item referenced by data-name-5 contains a symbolic destination previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.

19. The contents of the data item referenced by data-name-3 indicate the status condition of the previously executed SEND, ENABLE OUTPUT or DISABLE OUTPUT statement.

    The actual association between the contents of the data item referenced by data-name-3 and the status condition itself is defined in Table 13-1.

20. If, during the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the MCS determines that any specified destination is unknown, the contents of the data item referenced by data-name-3 and all occurrences of the data items referenced by data-name-4 are updated.

    The contents of the data item referenced by data-name-4 when equal to 1 indicate that the associated value in the area referenced by data-name-5 has not been previously defined to the MCS. Otherwise, the contents of the data item referenced by data-name-4 are set to zero.

All Formats:

21. Table 13-1 indicates the possible contents of the data items referenced by data-name-10 for Format 1 and by data-name-3 for Format 2 at the completion of each statement shown. An 'X' on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

Table 13-1. Communication Status Key Condition

RECEIVE	SEND	ACCEPT MESSAGE COUNT	ENABLE INPUT (without terminal)	ENABLE INPUT (with terminal)	ENABLE OUTPUT	DISABLE OUTPUT (without terminal)	DISABLE OUTPUT (with terminal)	DISABLE OUTPUT	STATUS KEY CODE	
X	X	X	X	X	X	X	X	X	00	No error detected. Action completed.
	X								10	One or more destinations are disabled. Action completed.
	X				X			X	20	One or more destinations unknown. action completed for known destinations. No action taken for unknown destinations. Data-name-4 (ERROR KEY) indicates known or unknown.
X		X	X			X			20	One or more queues or sub-queues unknown. No action taken.
				X			X		20	The source is unknown. No action taken.
	X				X			X	30	Content of DESTINATION COUNT invalid. No action taken.
			X	X	X	X	X	X	40	Password invalid. No enabling/disabling action taken.
	X								50	Character count greater than length of sending field. No action taken.
	X								60	Partial segment with either zero character count or no sending area specified. No action taken.

PROCEDURE DIVISION IN THE COMMUNICATION MODULE

THE ACCEPT MESSAGE COUNT STATEMENT

Function

The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available.

General Format

```
ACCEPT cd-name MESSAGE COUNT
```

Syntax Rule

CD-name must reference an input CD.

General Rules

1. The ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in a queue, sub-queue-1, ...

2. Upon execution of the ACCEPT MESSAGE COUNT statement, the contents of the area specified by a communication description entry must contain at least the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-10 (STATUS KEY) and data-name-1 (MESSAGE COUNT) of the area associated with the communication entry to be appropriately updated. (See THE COMMUNICATION DESCRIPTION - COMPLETE ENTRY SKELETON.)

THE DISABLE STATEMENT

Function

The DISABLE statement notifies the MCS to inhibit data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

General Format

```
DISABLE { INPUT  [TERMINAL] } cd-name WITH KEY { identifier-1 }
        { OUTPUT            }                  { literal-1    }
```

Syntax Rules

1. Cd-name must reference an input CD when the INPUT phrase is specified.

2. Cd-name must reference an output CD when the OUTPUT phrase is specified.

3. Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric.

General Rules

1. The DISABLE statement provides a logical disconnection between the MCS and the specified sources or destinations. When this logical disconnection is already in existence, or is to be handled by some other means external to this program, the DISABLE statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the DISABLE statement.

2. When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all queues and sub-queues is deactivated. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful.

3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queues and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are deactivated.

4. When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are deactivated.

5. Literal-1 or the contents of the data-name referenced by identifier-1 will be matched with a password built into the system. The DISABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

   The MCS must be capable of handling a password of from one to ten characters inclusive.

6. The MCS will insure that the execution of a DISABLE statement will cause the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement will never cause the remaining portion of the message to be terminated during transmission to or from a terminal.

THE ENABLE STATEMENT

Function

The ENABLE statement notifies the MCS to allow date transfer between specified output queues and destinations for output or between specified sources and input queues for input.

General Format

```
ENABLE  {INPUT  [TERMINAL]}  cd-name WITH KEY  {identifier-1}
        {OUTPUT            }                  {literal-1   }
```

Syntax Rules

1. cd-name must reference an input CD when the INPUT phrase is specified.

2. Cd-name must reference an output CD when the OUTPUT phrase is specified.

3. Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric.

General Rules

1. The ENABLE statement provides a logical connection between the MCS and the specified sources or destinations. When this logical connection is already in existence, or is to be handled by some other means external to this program, the ENABLE statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the ENABLE statement.

2. When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all associated queues and sub-queues which are already enabled is activated. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful to the MCS.

3. When the INPUT phrase without the optional word TERMINAL is specified, the logcal paths for all of the sources associated with the queue and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are activated.

4. When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are activated.

5. Literal-1 or the contents of the data item referenced by identifier-1 will be matched with a password built into the system. The ENABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

   The MCS must be capable of handling a password of from one to ten characters inclusive.

THE RECEIVE STATEMENT

Function

The RECEIVE statement makes available to the COBOL program a message, message segment, or a portion of a message or segment, and pertinent information about that data, from a queue maintained by the Message Control System. The RECEIVE statement allows for a specific imperative statement when no data is available.

General Format

```
RECEIVE  cd-name { MESSAGE } { INTO } identifier-1
                 { SEGMENT }

        [; NO DATA imperative-statement]
```

Syntax Rule

Cd-name must reference an input CD.

General Rules

1. The contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name designate the queue structure containing the message. (See THE COMMUNICATION DESCRIPTION - COMPLETE ENTRY SKELETON.)

2. The message, message segment, or portion of a message or segment is transferred to the receiving character positions of the area referenced by identifier-1 aligned to the left without space fill.

3. When during the execution of a RECEIVE statement, the MCS makes data available in the data item referenced by identifier-1, control is transferred to the next executable statement, whether or not the NO DATA phrase is specified.

4. When, during the execution of a RECEIVE statement, the MCS does not make data available in the data item referenced by identifier-1:

   a. If the NO DATA phrase is specified, the RECEIVE operation is terminated with the indication that action is complete (see general rule 5), and the imperative statement in the NO DATA phrase is executed.

   b. If the NO DATA phrase is not specified, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.

c. If one or more queues or sub-queues are unknown to the MCS, control passes to the next executable statement, whether or not the NO DATA phrase is specified. (See Table 13-1 for Status.)

5. The data items identified by the input CD are appropriately updated by the Message Control System at each execution of a RECEIVE statement.

6. A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used). However, the MCS does not pass any portion of a message to the object program until the entire message is available in the input queue, even if the SEGMENT phrase of the RECEIVE statement is specified.

7. When the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to the data transfer:

   a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.

   b. If a message size is less than the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

   c. If a message size is greater than the area referenced by identifier-1, the message fills the area referenced by identifier-1 left to right starting with the leftmost character of the message. The remainder of the message can be transferred to the area referenced by identififer-1 with subsequent RECEIVE statements referring to the same queue, sub-queue ... . The remainder of the message, for the purposes of applying rules 7a, 7b, and 7c, is treated as a new message.

8. When the SEGMENT phrase is used, the following rules apply:

   a. If a segment is the same size as the area referenced by identifier-1, the segment is stored in the area referenced by identifier-1.

   b. If the segment size is less than the area referenced by identifier-1, the segment is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

   c. If a segment size is greater than the area referenced by identifier-1, the segment fills the area referenced by identifier-1 left to right starting with the leftmost character of the segment. The remainder of the segment can be transferred to

the area referenced by identifier-1 with subsequent RECEIVE statements calling out the same queue, sub-queue ... . The remainder of the segment, for the purposes of applying rules 8a, 8b and 8c, is treated as a new segment.

d. If the text to be accessed by the RECEIVE statement has associated with it an end of message indicator or end of group indicator, the existence of an end of segment indicator associated with the text is implied and the text is treated as a message segment according to general rule 8.

9. Once the execution of a RECEIVE statement has returned a portion of a message, only subsequent execution of RECEIVE statements in that run unit can cause the remaining portion of the message to be returned.

10. After the execution of a STOP RUN statement, the disposition of a remaining portion of a message partially obtained in that run unit is defined by the implementor. (See THE STOP STATEMENT in Chapter 3.)

THE SEND STATEMENT

Function

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 Message Control System.

General Format

Format 1:

```
SEND  cd-name FROM identifier-1
```

Format 2:

```
SEND cd-name   [FROM identifier-1] { WITH identifier-2 }
                                   { WITH ESI          }
                                   { WITH EMI          }
                                   { WITH EGI          }

[ { BEFORE } ADVANCING { { identifier-3 }      [ LINE  ] }  ]
  { AFTER  }           { { integer      }      [ LINES ] }
                       { { mnemonic-name }                }
                       { { PAGE          }                }
```

Syntax Rules

1. CD-name must reference an output CD.

2. Identifier-2 must reference a one character integer without an operational sign.

3. When identifier-3 is used in the ADVANCING phrase, it must be the name of an elementary integer item.

4. When the mnemonic-name phrase is used, the name is identified with a particular feature specified by the implementor. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the Environment Division.

5. Integer or the value of the data item referenced by identifier-3 may be zero.

General Rules

All Formats:

1. When a receiving communication device (printer, display screen, card punch, etc.) is oriented to a fixed line size:

   a. Each message or message segment will begin at the leftmost character position of the physical line.

   b. A message or message segment that is smaller than the physical line size is released so as to appear space-filled to the right.

   c. Excess characters of a message or message segment will not be truncated. Characters will be packed to a size equal to that of the physical line and then transmitted to the device. The process continues on the next line with the excess characters.

2. When a receiving communication device (paper tape punch, another computer, etc.) is oriented to handle variable length messages, each message or message segment will begin on the next available character position of the communications device.

3. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name to the user's indication of the number of leftmost character positions of the data item referenced by identifier-1 from which data is to be transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are zero, no characters of the data item referenced by identifier-1 are transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred. (See Table 13-1 for Status.)

4. As part of the execution of a SEND statement, the contents of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name is updated by the MCS. (See THE COMMUNICATION DIVISION - COMPLETE ENTRY SKELETON.)

5. The effect of having special control characters within the contents of the data item referenced by identifier-1 is undefined.

6. A single execution of a SEND statement for Format 1 releases only a single portion of a message or of a message segment to the MCS.

A single execution of a SEND statement of Format 2 never releases to the MCS more than a single message or a single message segment as indicated by the contents of the data item referenced by identifier-2 or by the specified indicator ESI, EMI or EGI.

However, the MCS will not transmit any portion of a message to a communications device until the entire message is placed in the output queue.

7. During the execution of the run unit, the disposition of a portion of a message not terminated by an EMI or EGI is undefined. However, the message does not logically exist for the MCS and hence cannot be sent to a destination.

   After the execution of a STOP RUN statement, any portion of a message transferred from the run unit via a SEND statement, but not terminated by an EMI or EGI, is purged from the system. Thus no portion of the message is sent.

8. Once the execution of a SEND statement has released a portion of a message to the MCS, only subsequent execution of SEND statements in the same run unit can cause the remaining portion of the message to be released.

Format 2:

9. The contents of the data item referenced by identifier-2 indicate that the contents of the data item referenced by identifier-1 are to have associated with it an end of segment indicator, and end of message indicator or an end of transmission indicator according to the following schedule:

If the contents of the data item referenced by identifier-2 is	then the contents of data item referenced by identifier-1 have associated with it	which means
'0'	no indicator	no indicator
'1'	ESI	an end of segment indicator
'2'	EMI	an end of message indicator
'3'	EGI	an end of group indicator
Any character other than '1', '2', or '3' will be interpreted as '0'		
If the content of the data item referenced by identifier-2 is other than '1', '2', or '3', and identifier-1 is not specified, then an error is indicated by the value in the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred.		

10. The ESI indicates to the MCS that the message segment is complete. The EMI indicates to the MCS that the message is complete. The EGI indicates to the MCS that the group of messages is complete. The implementor will specify the interpretation that is given to the EGI by the MCS. The MCS will recognize these indications and establish whatever is necessary to maintain group, message, and segment control.

11. The hierarchy of ending indicators is EGI, EMI and ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI.

12. The ADVANCING phrase allows control of the vertical positioning of each message or message segment on a communication device where vertical positioning is applicable. If vertical positioning is not applicable on the device, the MCS will ignore the vertical positioning specified or implied.

13. If identifier-2 is specified and the content of the data item referenced by identififer-2 is zero, the ADVANCING phrase is ignored by the MCS.

14. On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing will be provided by the implementor to act as if the user had specified AFTER ADVANCING 1 LINE.

15. If the ADVANCING phrase is implictly or explicitly specified and vertical positioning is applicable, the following rules apply:

    a. If identifier-3 or integer is specified, characters transmitted to the communication device will be repositioned vertically downward the number of lines equal to the value associated with the data item referenced by identifier-3 or integer.

    b. If mnemonic-name is specified, characters transmitted to the communication device will be positioned according to the rules specified by the implementor for that communication device.

    c. If the BEFORE phrase is used, the message or message segment is represented on the communication device before vertical repositioning according to general rules 15a and 15b above.

    d. If the AFTER phrase is used, the message or message segment is represented on the communication device after vertical repositioning according to general rules 15a and 15b above.

    e. If PAGE is specified, characters transmitted to the communication device will be represented on the device before or after (depending upon the phrase used) the device is repositioned to the next page. If PAGE is specified but page has no meaning in conjunction with a specific device, then advancing will be provided by the implementor to act as if the user had specified BEFORE or AFTER (depending upon the phrase used) ADVANCING 1 LINE.

Chapter 14

Programming Techniques, Useful Hints and Program Sizing

PROGRAMMING TECHNIQUES

Although COBOL is written in an essentially free form, the user will nevertheless reap many advantages from a few self-imposed disciplines. It is suggested that these should include the following:

1. Use of the first 256 bytes of working-storage for variables which are frequently referenced will produce more compact and efficient code.

2. Use subscripts as sparingly as possible because each subscript has a storage requirement approximately equal to the size of a normal instruction.

3. For ACCEPT and DISPLAY the compiler generates one instruction per elementary item of the data-name being displayed/accepted. Therefore redefine a group of fields as a single field for DISPLAY whenever possible and avoid unnecessary numbers of small fields in ACCEPT.

4. Use FILLER instead of a data-name for any elementary field not referenced explicitly because the word FILLER is compacted to one character in the Data Dictionary.

5. Keep the number of digits in numeric fields as small as possible.

6. Whenever possible move a group instead of making several elementary moves.

USEFUL HINTS

When writing interactive programs the following facilities of Apple III COBOL should be remembered:

1. By use of the CURSOR IS facility and the ACCEPT statement it is easy to program conditionally depending on the cursor position after a menu type of prompt. The operator need then only move the cursor to the option required to reply to the prompt, or just press RETURN in the default case.

2. By use of the ACCEPT FROM CONSOLE facility it is easy to pass parameters to your program via the Run command line. See THE ACCEPT STATEMENT in Chapter 3.

3. Remember always to end your Apple III COBOL program with a full stop (period). Invalid intermediate code can result if this final full stop is missing.

4. If the STOP "literal" statement is used in a program, execution of the program halts at this statement with the literal displayed on the CRT screen. Execution continues on pressing the Carriage Return (CR) key. It should be noted that if any string of characters terminated by a CR character has been keyed and is waiting to be read the program will appear not to halt. Examples of this are if the CR key was inadvertently pressed twice at the last command entered or if parameters to the Run command have not yet been read.

SIZING

GENERAL DESCRIPTION

There are three aspects to sizing a program: the source code, the Data Dictionary and the compiled code.

The maximum number of source statements per program is limited, firstly by the space available for the compiler's data dictionary and secondly by that available to load the generated program.

The Data Dictionary contains an entry for every user-defined name in the program. Detailed information is contained in the next section.

The maximum number of bytes available for the user's program and work space for any given configuration, can be found in the appropriate Operating Guide. A guide for calculating the size of the generated program is as follows:

```
The sum of the Record size for each file in bytes
+    the Record size for each Working-Storage record in bytes
+    the number of characters in all Procedure Division literals
+    60 bytes per File
+    300 bytes control area
+    6 bytes per COBOL instruction with the following qualifiers:
```

for an ACCEPT/DISPLAY statement add 3 bytes per elementary item within the Accepted/Displayed data-name.

for every subscript used in a statement add 7 bytes

for a comparision add 6 bytes

for an implicitly generated comparison,e.g. PERFORM UNTIL, READ AT END, add 6 bytes

DATA DICTIONARY

The Data Dictionary is constructed as the program is compiled. Its size depends on the host operating system. Each user defined name will have an entry in this dictionary. The number of bytes required for each entry is given in Table 14-1.

Table 14-1. Data Dictionary Entry Sizing

User-defined name	Number of Bytes [1]
File-name	18 + n
Record-name	8 + n
Key-name	8 + n
Status-name	8 + n
Paragraph-name	6 + n
Data-name Group	8 + n [2]
Alphanumeric < 32 characters	7 + n [2]
Alphanumeric > 32 characters	8 + n [2]
Numeric integer	7 + n [2]
Numeric non integer	8 + n
Numeric edited	7 + n + x

1 —— n = number of characters in user-defined name.

For a FILLER, n = 1.

x = number of characters in PICture, after coalescing repetitions.

e.g. 9999.9 = 3 bytes
9(4).9 = 3 bytes
Z(2)9(4).9(3) = 4 bytes

2 —— Subtract 1 byte if item is in the first 256 bytes of Working-Storage.

Add 4 bytes if item has an OCCURS clause associated with it.

Add 2 bytes if item is subordinate to an item described with OCCURS.

Appendix A

Reserved Word List

This appendix contains a full list of COBOL and Apple III COBOL reserved words. A shaded reserved word is an Apple III COBOL extension to ANSI COBOL.

The / symbol denotes that the text up to that point is a reserved word, as is the whole word.

e.g., In INDEX/ED, INDEX and INDEXED are reserved words. In SPACE/S, SPACE and SPACES are reserved words.

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
ARE
AREA/S
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BLANK
BLOCK
BOTTOM
BY

CALL
CANCEL
CD
CHARACTER/S
CLOCK-UNITS
CLOSE
COBOL
CODE/-SET
COLLATING
COMMA
COMMUNICATION
COMP/UTATIONAL/-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTAINS
COPY
CORR/ESPONDING
COUNT
CRT
CRT-UNDER
CURRENCY
CURSOR

DATA
DATE-WRITTEN
DATE/-COMPILED
DAY
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-2
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

ELSE
ENABLE
END
ENTER
ENVIRONMENT
EQUAL
ERROR
EVERY
EXCEPTION
EXCESS-3
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILLER
FIRST
FOOTING
FOR
FORMFEED
FROM

GIVING
GO
GREATER

HIGH-VALUE/S

I-O/-CONTROL
IDENTIFICATION
IF
IN
INDEX/ED
INITIAL
INPUT/-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST/IFIED

KEY

LABEL
LEADING
LEFT
LESS
LIMIT/S
LINAGE/-COUNTER
LINE/S
LINKAGE
LOCK
LOW-VALUE/S

MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NOT
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN

OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PERFORM
PIC/TURE
POINTER
POSITIVE
PROCEDURE/S
PROCEED
PROGRAM/-ID

QUEUE
QUOTE/S

RANDOM
RD
READ
RECEIVE
RECORD/S
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
RERUN
RETURN
REWRITE
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT/-LIMIT
SELF T
SENL
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET

SIGN
SIZE
SORT
SORT-MERGE
SOURCE/-COMPUTER
SPACE/S
SPECIAL-NAMES
STANDARD/-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SWITCH
SYMBOLIC
SYNC/HRONIZED
SYSIN
SYSOUT

TAB
TABLE
TALLYING
TAPE
TERMINAL
THAN
THEN
THROUGH
THRU
TIME/S
TO
TOP
TRAILING
TYPE

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE/S
VARYING

WHEN
WITH
WORDS

WORKING-STORAGE
WRITE

ZERO/ES or S

. (period)
(
-
*
**
)
;
+
/
,
<
=
>

Appendix B

Character Sets and Collating Sequence

ASCII	HEX	COBOL
NUL	ØØ	x
SOH	Ø1	x
STX	Ø2	x
ETX	Ø3	x
EOT	Ø4	x
ENQ	Ø5	x
ACK	Ø6	x
BEL	Ø7	x
BS	Ø8	x
HT	Ø9	x
LF	ØA	x
VT	ØB	x
FF	ØC	x
CR	ØD	x
SO	ØE	x
SI	ØF	x
DLE	1Ø	x
DC1	11	x
DC2	12	x
DC3	13	x
DC4	14	x
NAK	15	x
SYN	16	x
ETB	17	x
CAN	18	x
EM	19	x
SUB	1A	x
ESC	1B	x
FS	1C	x
GS	1D	x
RS	1E	x
US	1F	x
space	2Ø	
!	21	x
"	22	
#	23	x
$	24	
%	25	x
&	26	x
'	27	x
(	28	
)	29	
*	2A	
+	2B	
,	2C	
-	2D	
.	2E	

ASCII	HEX	COBOL
/	2F	
Ø	3Ø	
1	31	
2	32	
3	33	
4	34	
5	35	
6	36	
7	37	
8	38	
9	39	
:	3A	x
;	3B	
<	3C	
=	3D	
>	3E	
?	3F	x
@	4Ø	x
A	41	
B	42	
C	43	
D	44	
E	45	
F	46	
G	47	
H	48	
I	49	
J	4A	
K	4B	
L	4C	
M	4D	
N	4E	
O	4F	
P	5Ø	
Q	51	
R	52	
S	53	
T	54	
U	55	
V	56	
W	57	
X	58	
Y	59	
Z	5A	
	5B	x
	5C	x
	5D	x

ASCII	HEX	COBOL
	5E	x
	5F	x
	6Ø	x
a	61	
b	62	
c	63	
d	64	
e	65	
f	66	
g	67	
h	68	
i	69	
j	6A	
k	6B	
l	6C	
m	6D	
n	6E	
o	6F	
p	7Ø	
q	71	
r	72	
s	73	
t	74	
u	75	
v	76	
w	77	
x	78	
y	79	
z	7A	
	7B	x
	7C	x
	7D	x
	7E	x
DEL	7F	x

Appendix C
Glossary

INTRODUCTION

The terms in this Chapter are defined in accordance with their meaning as used in this document describing Apple III COBOL and may not have the same meaning for other languages.

These definitions are also intended to be either reference material or introductory material to be reviewed prior to reading the detailed language specifications that are contained in this manual. For this reason, these definitions are, in most instances, brief and do not include detailed syntactic rules.

DEFINITIONS

Abbreviated Combined Relation Condition. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode. The manner in which records are to be operated upon within a file

Actual Decimal Point. The physical representation, using either of the decimal point characters[. (period) or , (comma)] of the decimal point position in a data item.

Alphabet-Name. A user-defined word in the SPECIAL-NAMES paragraph of the Environment Division that assigns a name to a specific character set and/or collating sequence.

Alphabetic Character. A character that belongs to the following set of letters: A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z and the space. Also a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y and z which are converted to their upper case equivalents.

Alphanumeric Character. Any character in the computer's character set.

Animation Option. The Apple III COBOL Animation Option is a COBOL oriented debugging tool for use with the Apple III COBOL product.

Arithmetic Expression. An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator. A single character, or a fixed two-character combination, that belongs to the following set:

Character	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Ascending Key. A key upon whose values data is ordered, starting with the lowest value of key up to the highest value of key in accordance with the rules for comparison of the data items.

Assumed Decimal Point. A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition. A condition caused in one of two circumstances:

1. During the execution of a READ statement for a sequentially accessed file.

2. During the execution of a RETURN statement when no next logical record exists for the associated sort or merge file.

Block. A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Cd-Name. A user-defined word that names an MCS interface area described in a communication description entry within the Communication Section of the Data Division.

Called Program. A program which is the object of a CALL statement combined at run time with the calling program to produce a run unit.

Calling Program. A program which executes a CALL to another program.

Character. The basic indivisible unit of the language.

Character Set (Apple III COBOL). The complete Apple III COBOL character set consists of all characters listed below:

Character	Meaning
0,1,...,9	Numeric digit
A,B,...,Z	Uppercase alphabetic
a,b,...,z	Lowercase alphabetic

	Space (Blank)
+	Plus Sign
-	Minus Sign
*	Asterisk
/	Stroke (Virgule or Slash)
=	Equal Sign
$	Currency Sign
,	Comma
;	Semicolon
.	Period (Decimal Point, Fullstop)
"	Quotation Mark
(	Left Parenthesis
)	Right Parenthesis
>	Greater Than Symbol
<	Less Than Symbol

Character Position. A character position is the amount of physical storage required to store a single standard data format character described as usage in DISPLAY. Further characteristics of the physical storage are defined by the implementor.

Character-String. A sequence of contiguous characters which form a Apple III COBOL word, a literal, a PICTURE character-string or a comment-entry.

Class Condition. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Clause. A clause is an ordered set of consecutive Apple III COBOL character- strings whose purpose is to specify an attribute of an entry.

COBOL Word. (See Word)

Collating Sequence. The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging and or comparing.

Column. A character position within a print line. The columns are numbered from one, by one, starting at the left-most character position of the print line and extending to the right-most character position of the print line.

Combined Condition. A condition that is the result of connecting two or more conditions with the 'AND' or the 'OR' logical operator.

Comment Entry. An entry in the Identification Division that may be any combination of characters from the computer character set.

Comment Line. A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection before printing the comment.

Communication Description Entry. An entry in the Communication Section of the Data Division that is composed of the level indicator CD, followed by a cd-name, and then followed by a set of clauses as required. It describes the interface between the Message Control System (MCS) and the COBOL program.

Communication Device. A mechanism (hardware or hardware/software) capable of sending data to a queue and/or receiving data from a queue. This mechanism may be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

Communication Section. The section of the Data Division that describes the interface areas between the MCS and the program, composed of one or more CD description entries.

Compile Time. The time at which an Apple III COBOL source program is translated by the compiler to an Apple III COBOL intermediate code program.

Compiler-Directing Statement. A statement, beginning with a compiler-directing verb, that causes the compiler to take a specific action during compilation.

Complex Condition. A condition in which one or more logical operators act upon one or more conditions. (See Negated Simple Condition, Combined Condition, Negated Combined Condition).

Computer-Name. A system-name that identifies the computer upon which the program is to be compiled or run.

Condition. A status of a program at execution time for which a truth value can be determined. Where the term "condition" (condition-1, condition-2,...) appears in these language specifications in or in reference to "condition" (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesised, or a negated simple condition.

Condition Name. The user-defined word assigned to a status of an implementor-defined switch or device.

Condition-Name Condition. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Variable. A data item one or more values of which has a condition-name assigned to it.

Conditional Expression. A simple condition specified in an IF, or PERFORM. (See Simple Condition and Complex Condition.)

Conditional Statement. A conditional statement specifies that the truth value of a condition is to be determined, and that the subsequent action of the run-time program is dependent on this truth value.

Conditional Variable. A data item one or more values of which has a condition-name assigned to it.

Configuration Section. A section of the Environment Division that describes overall specifications of source and run computers.

Connective. A reserved word that is used to:

1. Associate a data-name, paragraph-name, condition-name, or text-name with its qualifier.
2. Link two or more operands written in a series.
3. Form conditions (logical connectives). (See Logical Operator.)

Contiguous Items. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to one another.

Counter. A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

CRT. An interactive input/output device comprising a cathode ray tube by which an Operator can enter and receive visual data.

Currency Sign. The character "$" (dollar sign) in the Apple III COBOL character set.

Currency Symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in an Apple III COBOL source program, the currency symbol is identical to the currency sign.

Current Record. The record which is available in the record area associated with the file.

Current Record Pointer. A conceptual entity that is used in the selection of the next record.

Cursor. The indicator on a CRT screen that marks the line and character position which the input/output control is currently referencing.

Data Clause. A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Data Description Entry. An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses as required.

Data Dictionary. A dictionary file of user defined names constructed by the Compiler containing the number of bytes for each entry.

Data Item. A character or set of contiguous characters (excluding in either case literals) defined as a unit of data by the Apple III COBOL program.

Data-name. A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats, "data-name" represents a word which can neither be subscripted, nor indexed unless specifically permitted by the rules for that format.

Debugging Line. A debugging line is any line with "D" in the indicator area of the line.

Debugging Section. A debugging section is a section that contains a USE FOR DEBUGGING statement.

Declaratives. A set of one or more special purpose sections written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sequence, followed by a set of associated paragraphs (0 or more).

Declarative-Sentence. A compiler-directing sentence consisting of a single USE statement terminated by the separator period (.).

Default Disk. The disk from which the compiler on run-time system is loaded and from which, in the absence of a specific drive identifier, any copy file or called code will be loaded if required.

Delimiter. A character (or sequence of contiguous characters) that identifies the end of a string of characters, and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key. A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Destination. The symbolic identification of the receiver of a transmission from a queue.

Digit Position. A digit position is the amount of physical storage required to store a single digit. This amount varies depending on the usage of the data item describing the digit position. Further characteristics of the physical storage are defined by the implementor.

Division. A set of sections or paragraphs (0 or more) that are formed and combined in accordance with a specific set of rules are called a division body. There are four divisions in an Apple III COBOL program: Identification, Environment, Data and Procedure.

Division Header. A combination of words followed by a period and a space that indicate the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION   USING data-name-1   data-name-2 ... .
```

Dynamic Access. An access mode in which specific logical records can be obtained from or placed into a disk file in a non-sequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access) during the scope of the same OPEN statement.

Editing Character. A single character or a fixed two character combination belonging to the same set:

Character	Meaning
B	Space
0	Zero
+	Plus
-	Minus
CR	Credit
DB	Debit
Z	Zero Suppress
*	Check Protect
$	Currency Sign
,	Comma
.	Period (Decimal Point)
/	Stroke (Virgule, Slash)

Elementary Item. A data item that is described as not being further logically subdivided.

End of Procedure Division. The physical position in an Apple III COBOL source program after which no further procedures appear.

Entry. Any descriptive set of consecutive clauses terminated by a period (.) and written in the Identification Division, Environment Division or Data Division of an Apple III COBOL source program.

Environment Clause. A clause that appears as part of an Environment Division entry.

Extend Mode. With the EXTEND phrase specified, the state of a file after execution of an OPEN statement, and before the execution of a CLOSE statement for the file.

Figurative Constant. A compiler-generated value referenced through the use of certain reserved words.

File. A collection of records.

File Clause. A clause that appears as part of any of the following Data Division entries:

File Description (FD)
Sort-Merge File Description (SD)
Communication Description (CD)

FILE-CONTROL. The name of an Environment Division paragraph in which the data files for a given source program are declared.

File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name. A user-defined word that names a file described in a file description entry or a sort-merge file description entry within the File Section of the Data Division.

File Organization. The permanent logical file structure established at the time that a file is created.

File Section. The section of the Data Division that contains file description entries together with their associated record descriptions.

Format. A specific arrangement of a set of data.

FORMS-2 Program. A screen formatting program that automatically generates Apple III COBOL CRT input/output coding from actual screen layout.

Group Item. A named contiguous set of elementary or group items.

High Order End. The leftmost character of a string of characters.

I-O-CONTROL. The name of an Environment Division paragraph in which object program requirements for specific input/output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input/output device are specified.

I-O Mode. The state of a file after execution of an OPEN statement, with the I-O phrase specified for that file, and before the execution of a CLOSE statement for that file.

Identifier. A data-name, followed as required by the syntactically correct combination of subscripts and indices necessary to make unique reference to a data item.

Imperative Statement. A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Implementor-Name. A system-name that refers to a particular feature available on the implementor's computing system.

Index. A computer storage position or register, the contents of which represent the identification of a particular element in a table.

Index Data Item. A data item in which the value associated with an index-name can be stored in a form specified by the implementor.

Index-Name. A user-defined word that names an index associated with a specific table.

Indexed Data-Name. An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed File. A file with indexed organization.

Indexed Organization. The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Indicator Area. The leftmost parameter position of an Apple III COBOL source record that indicates the use of the record.

Input File. A file that is opened in the input mode.

Input Mode. The state of a file after execution of an OPEN statement, with the INPUT phrase specified for that file and before the execution of a CLOSE statement for that file.

Input-Output File. A file that is opened in the I-O mode.

Input-Output Section. The section of the Environment Division that names the files and the external media used by a program and which provides information required for transmission and handling of data during execution of the run-time program.

Input Procedure. A set of statements that is executed each time a record is released to the sort file.

Integer. A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Intermediate Code. The code produced by the Apple III COBOL compiler from the source code entered, and which the Run Time System 'fast loads' for execution.

Invalid Key Condition. A condition, at object time, caused when a specified value of the key associated with an indexed or relative file is determined to be invalid.

Issue Disk. The flexible diskette on which the Apple III COBOL software is supplied to users.

Key. A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key of Reference. The key currently being used to access records within an indexed file.

Key Word. A reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name. A system-name that specifies a particular programming language.

Level Indicator. Two alphabetic characters that identify a specific type of file or a position in hierarchy.

Level-Number. A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one or two digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record.

Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-number 77 identifies special properties of a data description entry.

Library-Name. A user-defined word that names an Apple III COBOL library intermediate file that is to be used by the compiler for a given source program compilation.

Library-Text. A sequence of character-strings and/or separators in a COBOL library.

Line Sequential File Organization. A sequential file containing variable length records separated by the C/R (carriage return) and L/F (line feed) characters.

Linkage Section. The section in the Data Division of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal. A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator. The reserved word 'NOT'. It can be used for logical negation.

Logical Record. The most inclusive data item. The level-number for a record is 01.

Low Order End. The rightmost character of a string of characters.

MCS. (See Message Control System).

Merge File. A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

Message. Data associated with an end of message indicator or an end of group indicator. (See Message Indicators)

Message Control System (MCS). A communication control system that supports the processing of messages.

Message Count. The count of the number of complete messages that exist in the designated queue of messages.

Message Indicators. EGI (end of group indicator), EMI (end of message indicator), and ESI (end of segment indicator) are conceptual indications that serve to notify the MCS that a specific condition exists (end of group, end of message, end of segment).

Within the hierarchy of EGI, EMI, and ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI. An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by an ESI, EMI, or EGI. A message may be terminated by an EMI or EGI.

Message Segment. Data that forms a logical subdivision of a message normally associated with an end of segment indicator. (See Message Indicators).

Mnemonic-Name. A user-defined word that is associated in the Environment Division with a specified implementor-name.

Native Character Set. The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence. The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Combined Condition. The 'NOT' logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition. The 'NOT' logical operator immediately followed by a simple condition.

Next Executable Sentence. The next sentence to which control will be transferred after execution of the current statement is complete.

Next Executable Statement. The next statement to which control will be transferred after execution of the current statement is complete.

Next Record. The record which logically follows the current record of a file.

Noncontiguous Items. Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.

Nonnumeric Item. A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal. A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character. A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item. A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

Numeric Literal. A literal composed of one or more numeric characters that also may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the run-time program is executed, is described.

Open Mode. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

Operand. Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this publication, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional Word. A reserved word that is included in a specified format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File. A file that is opened in either the output mode or extend mode.

Output Mode. The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.

Output Procedure. A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merged order.

Paragraph. In the Procedure Division, a paragraph-name followed by a period and a space and optionally by one or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header. A reserved word, followed by a period and a space, that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers are:

In the Identification Division:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

In the Environment Division:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name. A user-defined word that identifies and begins a paragraph in the Procedure Division.

Phrase. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of an Apple III COBOL procedural statement or of a COBOL clause.

Physical Record. (See Block)

Prime Record Key. A key whose contents uniquely identify a record within an indexed file.

Procedure. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-Name. A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name or a section-name.

Program-Name. A user-defined word that identifies a COBOL source program.

Pseudo-Text. A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters.

Pseudo-Text Delimiter. Two contiguous equal sign (=) characters used to delimit pseudo-text.

Punctuation Character. A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
.	period
"	quotation mark
)	left parenthesis
(	right parenthesis
	space
=	equal sign

Qualified Data-Name. An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier.
1. A data-name which is used in a reference together with another data-name at a lower level in the same hierarchy.
2. A section-name which is used in a reference together with a paragraph-name specified in that section.
3. A library-name which is used in a reference together with a text-name associated with that library.

Queue. A logical collection of messages awaiting transmission or processing.

Queue Name. A symbolic name that indicates to the MCS the logical path by which a message or a portion of a completed message may be accessible in a queue.

Random Access. An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

Record. (see Logical Record)

Record Area. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description. (See Record Description Entry)

Record Description Entry. The total set of data description entries associated with a particular record.

Record Key. A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record-Name. A user-defined word that names a record described in a record description entry in the Data Division.

Reference-Format. A format that provides a standard method for describing COBOL source programs.

Relation. (See Relational Operator)

Relation Character. A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition. The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specified relationship to the value of another arithmetic expression or data item. (See Relational Operator).

Relational Operator. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

Relative File. A file with relative organization.

Relative Key. A key whose contents identify a logical record in a relative file.

Relative Organization. The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Reserved Word. A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Routine-Name. A user-defined word that identifies a procedure written in a language other than COBOL

Run-Time Debug. An option available to Apple III COBOL programmers entered as a user option enabling break-point facilities in run-time programs.

Run-Time. The time at which the intermediate code produced by the compiler is interpreted by the Run Time System for execution.

Run-Time-System (RTS). The software that interprets the intermediate code produced by the Apple III COBOL compiler and enables it to be executed by providing interfaces to the operating system and CRT.

Run Unit. A set of one or more intermediate code programs which function, at run time, as a unit to provide problem solutions.

Section. A set of none, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header. A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data and Procedure Divisions.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

```
CONFIGURATION SECTION
INPUT-OUTPUT SECTION
```

In the Data Division:

```
FILE SECTION
WORKING-STORAGE SECTION
LINKAGE SECTION
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-Name. A user-defined word which names a section in the Procedure Division.

Segment-Number. A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0', '1', ..., '9'. A segment-number may be expressed either as a one or two digit number, and is checked for syntax only.

Sentence. A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator. A punctuation character used to delimit character-strings.

Sequential Access. An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

Sequential File. A file with sequential organization.

Sequential Organization. The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Sign Condition. The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Simple Condition. Any single condition chosen from the set:

relation condition
class condition
switch-status condition
sign condition
(simple-condition)

Sort File. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

Sort-Merge File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

Source. The symbolic definition of the originator of a transmission to a queue.

SOURCE-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always

refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'source program'.

Special Character. A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

Special-Character Word. A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an Environment Division paragraph in which implementor-names are related to user-specified mnemonic-names.

Special Registers. Compiler-generated storage areas whose primary use is to store information produced in conjunction with the user of specified COBOL features.

Standard Data Format. The concept used in describing the characteristics of data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

Statement. A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Sub-Queue. A logical hierarchical division of a queue.

Subject of Entry. An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

Subprogram. (See Called Program).

Subscript. An integer whose value identifies a particular element in a table.

Subscripted Data-Name. An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Switch-Status Condition. The proposition, for which a truth value can be determined, that an implementor-defined switch, capable of being set to an 'on' or 'off' status, has been set to a specified status.

Symbol Function. The use of specified characters in the PICTURE clause to represent data types.

System-Name. A COBOL word which is used to communicate with the operating environment.

Syntax. The order in which elements must be put together to form a program.

Table. A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element. A data item that belongs to the set of repeated items comprising a table.

Terminal. The originator of a transmission to a queue, or the receiver of a transmission from a queue.

Text-Name. A user-defined word which identifies library text.

Text-Word. Any character-string or separator, except space, in a COBOL library or in pseudo-text.

Truth Value. The representation of the result of the evaluation of a condition in terms of one of two values

true
false

Unary Operator. A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1 respectively.

User-Defined Word. A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable. A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb. A word that expresses an action to be taken by a COBOL compiler or run time program.

Word. A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

Working-Storage Section. The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

77 Level-Description-Entry. A data description entry that describes a noncontiguous data item with the level-number 77.

Appendix D

Compile-Time Errors

The error descriptions that correspond to error numbers as printed on listings produced by the Apple III COBOL compiler are listed below. In the case of alternative meanings, relevancy is obvious from context.

ERROR	DESCRIPTION
01	Compiler Error; consult your Technical Support Service
02	Illegal format: data-name
03	Illegal format: literal
04	Illegal format: character
05	Declaration violates uniqueness of qualification
06	Too many data and procedure names have been declared
07	Obligatory reserved word missing
08	Nested COPY or unknown library text
10	The statement starts in the wrong area of the source line, i.e., reference format violation
21	'.' missing
22	'DIVISION' missing
23	'SECTION' missing
24	'IDENTIFICATION' missing
25	'PROGRAM-ID' missing
26	'AUTHOR' missing
27	'INSTALLATION' missing
28	'DATE-WRITTEN' missing
29	'SECURITY' missing
30	'ENVIRONMENT' missing
31	'CONFIGURATION' missing
32	'SOURCE-COMPUTER' missing
33	MEMORY SIZE/COLLATING SEQUENCE/SPECIAL-NAMES clause in error
34	'OBJECT-COMPUTER' missing
36	'SPECIAL-NAMES' missing
37	SWITCH Clause in error or system-name/mnemonic-name error
38	DECIMAL-POINT Clause in error
39	CONSOLE Clause in error
40	Illegal currency symbol
41	'.' missing
42	'DIVISION' missing
43	'SECTION' missing
44	'INPUT-OUTPUT' missing
45	'FILE-CONTROL' missing
46	'ASSIGN' missing
47	'SEQUENTIAL' or 'INDEXED' or 'RELATIVE' missing
48	'ACCESS' missing on indexed/relative file
49	'SEQUENTIAL/DYNAMIC' missing or too many alternate keys (>64)

50 Illegal combination ORGANIZATION/ACCESS/KEY
51 SELECT Clause phrase unrecognised
52 RERUN Clause syntax error
53 SAME RECORD AREA clash
54 file-name missing or illegal
55 'DATA DIVISION' missing
56 'PROCEDURE DIVISION' missing or unknown statement
57 Collating sequence not defined
58 'EXCLUSIVE', 'AUTOMATIC' or 'MANUAL' missing [1]
59 Non-exclusive lock mode specified for restricted file [1]
61 '.' missing
62 'DIVISION' missing
63 'SECTION' missing
64 file-name not specified in SELECT statement or invalid CD name
65 Record size integer missing or too large for line sequential
66 Illegal level number (01-49), or 01 level required, or level hierarchy incorrect
67 FD, CD or SD qualification contains syntax error
68 'WORKING-STORAGE' missing
69 'PROCEDURE DIVISION' missing or unknown statement
70 Data Description Qualifier or '.' missing
71 SIGN/USAGE illegal with COMP data-item or unsigned PICTURE data or incompatible with other qualifier
72 BLANK is illegal with non-numeric data-item
73 PICTURE clause too long (Numeric 18 Numeric Edited 512 Alphanumeric 8192)
74 VALUE clause on non-elementary data-item, or truncation, or wrong data type
75 'VALUE' in error or illegal for PICTURE type
76 FILLER/SYNCHRONIZED/JUSTIFIED/BLANK non-elementary item
77 Level with more than 8192 bytes or with 0 bytes
78 REDEFINES of unequal fields or different levels.
79 Data storage exceeds 64K bytes
81 Data Description Qualifier inappropriate or repeated
82 REDEFINES data-name not declared
83 USAGE must be COMP, DISPLAY or INDEX
84 SIGN must be LEADING or TRAILING
85 SYNCHRONIZED must be LEFT or RIGHT
86 JUSTIFIED must be RIGHT
87 BLANK must be ZERO
88 OCCURS must be numeric, non-zero and unsigned or multi-dimension "depending"
89 VALUE must be a literal, numeric literal or figurative constant
90 PICTURE string has illegal precedence or illegal character
91 INDEXED data-name missing or already declared
92 numeric edited PICTURE string is too large
101 Unrecognised verb
102 IF ... ELSE mismatch

103	Wrong data-type or data-name not declared
104	Paragraph name declared twice
105	Paragraph name same as data-name
106	Name required
107	Wrong combination of data types
108	Conditional statement not allowed in this context; must be an imperative statement
109	Malformed subscript
110	ACCEPT/DISPLAY wrong or Communications syntax incorrect
111	Illegal I-O Syntax
112	Invalid arithmetic statement
113	Invalid arithmetic expression
114	Working-storage space overflow during run-time
115	Invalid conditional expression
116	IF statements nested too deep, too many AFTER phrases in a PERFORM statement or internal transfer range exceeded
117	Incorrect structure of Procedure Division e.g. Sections out of order
118	Obligatory Reserved Word missing
119	Too many subscripts in one statement
120	Too many operands in one statement
141	Inter-segment procedure name duplication
142	IF ... ELSE mismatch at end of Source Input
143	Wrong data-type or data-name not declared
144	Paragraph name undeclared
145	Index-name declared twice
146	Bad cursor control: AT clause incorrectly specified
147	KEY declaration missing or alternate key clash or key in wrong place
148	STATUS declaration missing
149	Bad STATUS record
150	Undefined inter-segment reference
151	PROCEDURE DIVISION in error
152	USING parameter not declared in linkage section
153	USING parameter is not level 01 or 77
154	'FD' missing
157	Incorrect structure of Procedure Division: e.g. Sections out of order
154	USING parameter used twice in parameter list
160	Too many operands in one statement

In addition to these numbered error messages, the following message can be displayed with subsequent termination of the compilation:

I-O ERROR: {filename / OBJECT FILE}

where: filename is the erroneous file.
OBJECT FILE is one of .INT, .D?? or I?? (for segmented programs)

Any intermediate code file produced is not usable.

The following conditions will cause this error:

Disk overflow
File directory overflow
File full
Impossible I-O device usage

Other operating system dependent conditions can also cause this error.

NOTE:

You will notice that the numbers of the numbered error messages listed above are not continuous i.e., there are gaps in the numbering. The compiler should never have cause to generate an error message with a number not listed above. If you ever encounter such a number, consult your Apple III Product Technical Support office.

Appendix E
Run-Time Errors

Run-time error messages are preceded by the name and segment number of the currently executing intermediate code file.

There are two types of run-time errors: Recoverable and Fatal.

(a) Recoverable errors

If the programmer has selected STATUS for a file then error handling is his responsibility. This will generally only apply to errors that are not considered fatal by the operating system.

(b) Fatal errors

All errors except those above are fatal. They may come from the operating system or from the run-time system. Fatal errors cause a message to be output to the console which includes a 3 digit error code and reference to the COBOL statement subsequent to that in which the error occurred. These fall into two classes:

(i) Exceptions
These cover arithmetic overflow, subscript out of range, too many levels of perform nesting.

(ii) I-O errors
These exclude those for which STATUS is not selected as above.

Error	Description
151	Random read on sequential file
152	REWRITE on file not open I-O
153	Subscript out of range
154	Perform nesting exceeds 22 levels
155	Illegal command line
156	Invalid file operation
157	Object file too large
158	REWRITE on line-sequential file
159	Malformed line-sequential file
160	Overlay loading error
161	Illegal intermediate code
162	Arithmetic overflow or underflow
164	Specified CALL code not supplied or Attempt to call a COBOL module recursively (i.e. when is already active)
165	Incompatible releases of compiler and run-time system
170	Illegal operation in Indexed Sequential
171	Attempt to read I-S record in output/extend mode

172	Attempt to delete I-S record in non I-O mode
173	Attempt to write I-S record in input mode
176	Illegal inter-segment reference
180	COBOL file malformed
181	Fatal file malformation
194	File size too large (>0.5MB) or Failure to Open on Extent (CP/M 1.4 only)
195	DELETE/REWRITE not preceded by a READ
196	Relative (or Indexed) - Record number too large (> 65535)
197	File save failure
198	Program load failure (using CHAIN)
199	Undefined error condition

Appendix F

Syntax Summary

All the syntax for Apple III COBOL is summarized below.

Shading denotes that the feature is an Apple III COBOL extension to ANSI COBOL.

D denotes that the feature serves only a documentary purpose in Apple III COBOL.

GENERAL FORMAT FOR IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.
PROGRAM-ID.       program name
[AUTHOR.              [comment entry] ...]
[INSTALLATION.        [comment entry] ...]
[DATE-WRITTEN.        [comment entry] ...]
[DATE-COMPILED.       [comment entry] ...]
[SECURITY.            [comment entry] ...]
```

GENERAL FORMAT FOR ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.  source-computer-entry  [WITH DEBUGGING MODE].

OBJECT-COMPUTER.  object-computer-entry

    [,MEMORY SIZE integer  {WORDS     }]
                           {CHARACTERS}
                           {MODULES   }
    [,PROGRAM COLLATING SEQUENCE IS alphabet-name].

SPECIAL-NAMES.

    [,{SYSIN  } IS mnemonic-name-1
      {SYSOUT }
    [,{TAB} IS mnemonic-name-2
      {Ø  }
    [SWITCH {.} [IS mnemonic-name] ON STATUS IS condition-name-1
            {.}
            {.}
            {7}
            [OFF STATUS IS condition-name-2]

    [, alphabet-name IS
                                                          ]
            STANDARD-1
            NATIVE
            implementor-name                              ...
            literal-1 [{  THROUGH                        }]
                       {  THRU       literal-2           }
                       {ALSO literal-3 [, ALSO literal-4] ...}
              [literal-5[{  THROUGH                      }]]...
                         {  THRU       literal-6         }
                         {ALSO literal-7 [, ALSO literal-8]}
    [,CURRENCY SIGN IS literal-9]
    [,DECIMAL-POINT IS COMMA]
    [,CURSOR IS data-name-1]
    [,CONSOLE IS CRT]
```

```
[INPUT-OUTPUT SECTION.

 FILE-CONTROL.

      {file-control-entry}...  ].

[I-O-CONTROL.

     [ ; RERUN [ON {file-name-1        }]
               [   {implementor-name   }]

                                { {[END OF] {REEL}       }   OF file-name-2 } ]
                                { {         {UNIT}       }                  } ]      D
                          EVERY { {   integer-1 RECORDS  }                  } ]
                                { integer-2 CLOCK-UNITS                     } ]
                                { condition-name                            } ]

[ ; SAME    [RECORD    ]
            [SORT      ]  AREA FOR file-name-3  {,file-name-4}...  ] ...
            [SORT-MERGE]

[ ; MULTIPLE FILE TAPE CONTAINS file-name-5    [POSITION integer-3]
                                                                              D
       [, file-name-6  [POSITION integer-4]  ]      ...] ...]    .
```

GENERAL FORMAT FOR FILE-CONTROL ENTRY

Sequential SELECT:

```
SELECT file-name     [ OPTIONAL ] file-name

ASSIGN TO   external-file-name-literal      [,{external-file-name-literal}]
            file-identifier                   {file-identifier           }

[; RESERVE integer-1  [{AREA }]]                                    D
                      {AREAS}

;ORGANIZATION IS [{SEQUENTIAL     }]
                 {LINE SEQUENTIAL}

[;ACCESS MODE IS SEQUENTIAL]

[;FILE STATUS IS data-name]  .
```

Relative Select:

```
SELECT file-name

ASSIGN TO    {external-file-name-literal}  [,{external-file-name-literal}]
             {file-identifier           }     {file-identifier           }

[; RESERVE integer-1  [{AREA }]]                                    D
                      {AREAS}

ORGANIZATION IS RELATIVE

[;ACCESS MODE IS {SEQUENTIAL}   [,RELATIVE KEY IS data-name]}]
                 {RANDOM    }    ,RELATIVE KEY IS data-name }
                 {DYNAMIC   }

[;FILE STATUS IS data-name]  .
```

Indexed Select:

```
SELECT file-name

ASSIGN TO    {external-file-name-literal}  [,{external-file-name-literal}]
             {file-identifier           }     {file-identifier           }

[; RESERVE integer-1  [{AREA }]]                                    D
                      {AREAS}
```

```
    ;ORGANIZATION IS INDEXED
[
;ACCESS MODE IS   { SEQUENTIAL }
                  { RANDOM     }
                  { DYNAMIC    }
]
    ;RECORD KEY IS data-name-1

    [; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES ] ]   ...

    [;FILE STATUS IS data-name-3]  .
```

Sort or Merge Select:

```
    SELECT file-name

    ASSIGN TO { external-file-name-literal } ...  .
              { file-identifier           }
```

GENERAL FORMAT FOR THE DATA DIVISION

```
DATA DIVISION.
[ FILE SECTION.
  [ FD file-name

      [; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }]          D
                                                 {CHARACTERS}

      [; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]            D

       ; LABEL {RECORD IS  } {STANDARD}                                  D
               {RECORDS ARE} {OMITTED }

    [; VALUE OF data-name-1 IS {data-name-2}
                               {literal-1  }
       [, data-name-3 IS {data-name-4}]    ...]                         D
                         {literal-2  }

    [; DATA {RECORD IS  } data-name-3 [, data-name-4]...]                 D
            {RECORDS ARE}

    [; LINAGE IS {data-name-5} LINES [, WITH FOOTING AT {data-name-6}]
                 {integer-5  }                          {integer-6  }

       [, LINES AT TOP {data-name-7}][, LINES AT BOTTOM {data-name-8}]]
                       {integer-7  }                    {integer-8  }

    [; CODE-SET IS alphabet-name]  .                                      D

[record-description-entry] ... ]...

[SD file-name

      [; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS ]            D

      [; DATA {RECORD IS  } data-name-1 [, data-name-2] ... ] .
              {RECORDS ARE}

 {record-description-entry}  ... ] ... ]

      [ WORKING-STORAGE SECTION
        [77-level-description-entry]  ...
        [record-description-entry  ]
      ]

      [ LINKAGE SECTION
        [77-level-description-entry]  ...
        [record-description-entry  ]
      ]

      [ COMMUNICATION SECTION

        [communication-description-entry]
        [record-description-entry     ...] ...
      ]
```

GENERAL FORMAT FOR DATA DESCRIPTION ENTRY

Format 1:

```
level-number  {data-name-1}
              {FILLER     }
     [;REDEFINES data-name-2]
     [;{PICTURE}IS picture-string]
     {PIC    }
     [                  (COMPUTATIONAL    )  ]
     [;USAGE IS         (COMP             )  ]
     [                  (COMPUTATIONAL-3  )  ]
     [                  (COMP-3           )  ]
     [                  (DISPLAY          )  ]
     [                  (INDEX            )  ]
     [ [; SIGN IS]  {LEADING }  [SEPARATE CHARACTER] ]
     {TRAILING}
     [  ; OCCURS {integer-1  TO integer-2  TIMES  DEPENDING ON data-name-3} ]
                 {integer-2  TIMES                                         }
        [{ASCENDING } KEY IS data-name-4     [,data-name-5]] ... ...
        {DESCENDING}
           [INDEXED BY index-name-1    [, index-name-2]  ...]]
     [;{SYNCHRONIZED}   {LEFT }]                                    D
     {SYNC          }   {RIGHT}
     [;{JUSTIFIED}        RIGHT ]
     {JUST      }
     [;BLANK WHEN ZERO]
     [;VALUE IS literal] .
```

Format 2:

```
66 data-name-1; RENAMES data-name-2  [{THROUGH} data-name-3]
                                     {THRU   }
```

Format 3:

```
88 condition-name; {VALUE IS } literal-1 [{THROUGH} literal-2]
                   {VALUES ARE}          {THRU   }
     [,literal-3  [{THROUGH} literal-4] ]  ... .
                  {THRU   }
```

GENERAL FORMAT FOR COMMUNICATION DESCRIPTION ENTRY

FORMAT 1:

```
CD cd-name;
                                  [[; SYMBOLIC QUEUE IS data-name-1]
                                        [; SYMBOLIC SUB-QUEUE-1 IS data-name-2]
                                        [; SYMBOLIC SUB-QUEUE-2 IS data-name-3]
                                        [; SYMBOLIC SUB-QUEUE-3 IS data-name-4]
                                        [; MESSAGE DATE IS data-name-5]
FOR  [INITIAL ] INPUT                   [; MESSAGE TIME IS data-name-6]
                                        [; SYMBOLIC SOURCE IS data-name-7]
                                        [; TEXT LENGTH IS data-name-8]
                                        [; END KEY IS data-name-9]
                                        [; STATUS KEY IS data-name-10]
                                        [; MESSAGE COUNT IS data-name-11]]
                                   [data-name-1, data-name-2, ..., data-name-11]]
```

FORMAT 2:

```
CD cd-name;  FOR OUTPUT
     [; DESTINATION COUNT IS data-name-1
     [; TEXT LENGTH IS data-name-2]
     [; STATUS KEY IS data-name-3]
    [[; DESTINATION TABLE OCCURS integer-2 TIMES
        [[; INDEXED BY index-name-1    [, index-name-2]  ...]]]
     [; ERROR KEY IS data-name-4]
     [; SYMBOLIC DESTINATION IS data-name-4]
```

GENERAL FORMAT FOR PROCEDURE DIVISION

Declarative format:

```
PROCEDURE DIVISION [USING data-name-1   [, data-name-2] ...].

DECLARATIVES.
{section-name SECTION [segment-number]. declarative-sentence
  [paragraph-name.  [sentence] ... ] ... } ...

END DECLARATIVES.
{section-name SECTION   [segment-number]  .
  [paragraph-name. [sentence] ... ] ... } ...
```

Non-declarative format:

```
PROCEDURE DIVISION [ USING  data-name-1   [,data-name-2]   ...] .
  { paragraph-name. [sentence] ...} ...
```

GENERAL FORMAT FOR VERBS

```
ACCEPT dataname-1 [ AT { data-name-2 } ]   FROM CRT
                       { literal-1   }

ACCEPT identifier   [FROM CONSOLE]

ACCEPT identifier FROM { DATE }
                       { DAY  }
                       { TIME }

ACCEPT cd-name MESSAGE COUNT                                    D

ADD { identifier-1 } [ identifier-2 ]... TO identifier [ROUNDED]
    { literal-1    } [ literal-2    ]

              [; ON SIZE ERROR imperative-statement]

ADD { identifier-1 } { identifier-2 } ;[ identifier-3 ...]
    { literal-1    } { literal-2    }   [ literal-3    ]

              GIVING identifier [ROUNDED]

              [; ON SIZE ERROR imperative-statement]

ADD { CORRESPONDING } identifier-1 TO identifier-2 [ROUNDED]
    { CORR          }

ALTER { procedure-name-1 [ TO PROCEED TO ] procedure-name-2 }   ...

CALL { identifier-1 } USING data-name-1 [, data-name-2] ...
     { literal-1    }

CANCEL { identifier-1 } [ {,identifier-2} ]   ...
       { literal-1    } [ {,literal-2   } ]

                                D                D                    D
CLOSE file-name { REEL } [WITH LOCK] [,file-name [WITH LOCK] ] ..
                { UNIT }

                                                                     D
CLOSE file-name-1 [ { REEL } [ WITH NO REWIND ] ]
                  [ { UNIT }   FOR REMOVAL     ]
                  [   WITH   { NO REWIND }     ]
                             { LOCK      }

                  [ { REEL } [ WITH NO REWIND ] ] ]
, file-name-2     [ { UNIT }   FOR REMOVAL     ]   ...
                  [   WITH   { NO REWIND }     ]
                             { LOCK      }
```

```
                                                D                                D
CLOSE     file-name-1     [WITH LOCK]  [, file-name-2    [WITH LOCK]  ]  ...

COMPUTE   identifier-1    [ ROUNDED ]  [, identifier-2 [ROUNDED]  ]  ...

      = arithmetic-expression [; ON SIZE ERROR imperative-statement]

DELETE  file-name  RECORD  [; INVALID KEY  imperative-statement]   D
                                                                   D
DISABLE { INPUT  }{[TERMINAL]}   cd-name WITH KEY  {identifier-1}
        { OUTPUT }{          }                     {literal-1   }

DISPLAY {identifier-1} , {identifier-2}  ...  UPON CONSOLE
        {literal-1   }   {literal-2   }

DISPLAY {data-name-1} AT {data-name-2}  UPON {CRT      }
        {literal-3  }    {literal-4  }       {CRT-UNDER}

DIVIDE {identifier-1}  INTO  identifier-2  [ROUNDED]
       {literal-1   }

            [, identifier-3     [ROUNDED]  ]  ...
            [:ON SIZE ERROR     imperative-statement]

DIVIDE {identifier-1} {INTO} {identifier-2} GIVING identifier-3  [ROUNDED]
       {literal-1   } {BY  } {literal-2   }

          REMAINDER identifier-4 [;ON SIZE ERROR  imperative-statement]

                                                                   D
ENABLE  { INPUT  [TERMINAL] }  cd-name WITH KEY  {identifier-1}
        { OUTPUT            }                    {literal-1   }

ENTER  language-name  [routine-name].

EXIT   [PROGRAM].

GO TO[procedure-name].

GO TO procedure-name-1 {, procedure-name-2}...

     DEPENDING ON identifier

IF condition; {statement-1 }  [; ELSE statement-2      ]
              {NEXT SENTENCE}  [; ELSE NEXT SENTENCE   ]
```

```
INSPECT identifier-1 TALLYING tally-clause (as follows)

{ identifier-2 FOR { , { ALL       } { identifier-3 } } }
{                  { , { LEADING   } { literal-2   } } }
{                  { , { CHARACTERS }               } }
{ [ { BEFORE } INITIAL { identifier-4 } ] }                - (tally-clause)
{ [ { AFTER  }         { literal-3    } ] }

INSPECT identifier-1 REPLACING replacing-clause (as follows)

{ CHARACTERS BY { identifier-6 }                                                    }
{               { literal-4    }                                                    }
{ [ , ] { ALL       } , { identifier-5 } BY { identifier-6 }                        }
{       { LEADING   }   { literal-3    }    { literal-4    }                        }
{       { FIRST     }                                                              }
{ [ { BEFORE } INITIAL { identifier-7 } ]                                           }
{ [ { AFTER  }         { literal-5    } ]                                           }   - (replacing clause)

INSPECT identifier TALLYING tally-clause REPLACING replacing-clause

MERGE file-name-1 ON { ASCENDING  } KEY data-name-1 [, data-name-2] ...
                     { DESCENDING }

                [ ON { ASCENDING  } KEY daya-name-3 [, data-name-4] ... ] ...
                     { DESCENDING }

     [COLLATING SEQUENCE IS alphabet-name]

     USING file-name-2, file-name-3 [, file-name-4] ...
{ OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
{                                     { THRU    }                 }
{ GIVING file-name-5                                              }

MOVE { identifier-1 } TO identifier-2 [,identifier-3] ...
     { literal-1    }

MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

```
MULTIPLY {identifier-1} BY identifier-2  [ROUNDED]
         {literal-1   }

     [, identifier-3 [ROUNDED]   ...  [; ON SIZE ERROR imperative-statement]

MULTIPLY {identifier-1} BY{identifier-2} GIVING identifier-3  [ROUNDED]
         {literal-1   }  {literal-2   }

       [, identifier-4 [ROUNDED]   ...

       [; ON SIZE ERROR  imperative-statement]

      INPUT file-name-1 [REVERSED          ][,file-name-2 [REVERSED          ]]...   D
                        [WITH NO REWIND    ]            [WITH NO REWIND    ]
OPEN  OUTPUT file-name-3  [WITH NO REWIND]  ,file-name-4 [WITH NO REWIND] ] ...  ...
      I-O file-name-5     [, file-name-6] ...
      EXTEND  file-name-7      [, file-name-8]  ...

PERFORM  procedure-name-1[{THROUGH} procedure-name-2]
                         {THRU    }

PERFORM  perform-limits [VARYING {identifier-2} FROM {identifier-3}
                                 {index-name-1}      {index-name-2}
                                                     {literal-1   }
                          BY      {identifier-4} ] UNTIL  condition-1
                                  {literal-2   }

                        [ AFTER   {identifier-5} FROM   identifier-6
                                  {index-name-3}        index-name-4
                                                        literal-3
                          BY      {identifier-7}  UNTIL  condition-2
                                  {literal-4   }

                        [ AFTER   {identifier-8} FROM   identifier-9
                                  {index-name-5}        index-name-6
                                                        literal-5
                          BY      {identifier   }  UNTIL  condition-3 ]]
                                  {literal-6    }

READ file-name   [NEXT]   RECORD   [INTO identifier]

        [;AT END imperative-statement]

READ file-name  RECORD   [INTO identifier]  [;KEY IS data-name]

        [;INVALID KEY imperative-statement]                          D
```

```
RECEIVE cd-name {MESSAGE} INTO identifier-1 [; NO DATA imperative-statement]
                {SEGMENT}

RELEASE record-name [FROM identifier]

RETURN file-name RECORD [INTO identifier] ; AT END imperative-statement

REWRITE record-name [FROM identifier]
        [;INVALID KEY imperative-statement]

SEARCH identifier-1 [VARYING {identifier-2}]
                             {index-name-1}
     [; AT END imperative-statement-1]
      ; WHEN condition-1 {imperative-statement-2}
                         {NEXT SENTENCE        }
     [; WHEN condition-2 {imperative-statement-3}]
                         {NEXT SENTENCE        }

SEARCH ALL identifier-1 [; AT END imperative-statement-1]
      ; WHEN { data-name-1      {IS EQUAL TO} {identifier-3          }
             {                 {IS =       } {literal-1             }
             {                               {arithmetic-expression-1}
             { condition-name-1                                       }
     [ AND  { data-name-2      {IS EQUAL TO} {identifier-4          }
            {                  {IS =       } {literal-2             }
            {                               {arithmetic-expression-2}
            { condition-name-2                                       }] ...
   {imperative-statement-2}
   {NEXT SENTENCE         }

SEND cd-name FROM identifier-1

SEND cd-name [FROM identifier-1] {WITH identifier-2}
                                 {WITH ESI         }
                                 {WITH EMI         }
                                 {WITH EGI         }
     [BEFORE  ADVANCING {{identifier-3} [LINE ]}
     [AFTER             {{integer     } [LINES]}
                        {{mnemonic-name}       }
                        {{PAGE         }       }]
```

```
SET  {identifier-1}  [identifier-2]
     {index-name-1}  [index-name-2]  ...   TO   {identifier-3}
                                                {index-name-3}
                                                {integer-1   }

SET  {index-name-4}  [, index-name-5]
     {identifier-5}  [  identifier-6]  ...  {UP BY  }  {identifier-4}
                                            {DOWN BY}  {integer-2   }
                                                       {index-name-6}

SORT  file-name-1  ON {ASCENDING } KEY data-name-1  [, data-name-2] ...
                      {DESCENDING}
                [ON {ASCENDING } KEY data-name-3  [, data-name-4]] ...  ...
                    {DESCENDING}
    [COLLATING SEQUENCE IS alphabet-name]
    {INPUT PROCEDURE IS  section-name-1[{THROUGH} section-name-2]}
    {                                  {THRU   }                 }
    {USING file-name-2  ,[file-name-3] ...                       }
    {OUTPUT PROCEDURE IS section-name-3 [{THROUGH} section-name-4]}
    {                                   {THRU   }                }
    {GIVING file-name-4                                          }

START  file-name[ KEY  {IS EQUAL TO          }          data-name]
                       {IS =                 }
                       {IS GREATER than      }
                       {IS >                 }
                       {IS NOT LESS THAN     }
                       {IS NOT <             }
          [;INVALID  KEY  imperative-statement]

STOP  {RUN    }
      {literal}

STRING  {identifier-1}[,  {identifier-2}] ... DELIMITED BY {identifier-3}
        {literal-1   }    {literal-2   }                   {literal-3   }
                                                           {SIZE        }

    [,  {identifier-4}[,  {identifier-5}] ... DELIMITED BY {identifier-6}]
        {literal-4   }    {literal-5   }                   {literal-6   }
                                                           {SIZE        }
    INTO  identifier-7  [WITH POINTER  identifier-8]
    [, ON OVERFLOW imperative-statement

SUBTRACT  {identifer-1}[, {identifier-2}]...  FROM identifier-m  [ROUNDED]
          {literal-1  }   {literal-2   }
          [, identifier-n [ROUNDED] ]  ...
          [; ON SIZE ERROR  imperative-statement]
```

```
SUBTRACT {identifier-1} ,{identifier-2} ... FROM {identifier-m}
         {literal-1   }  {literal-2   }         {literal-m   }

         GIVING identifier -n [ROUNDED] [, identifier-o [ROUNDED] ] ...

         [; ON SIZE ERROR imperative-statement]
```

```
UNSTRING identifier-1
    [ DELIMITED BY [ALL]  {identifier-2} [, OR [ALL] {identifier-3}]... ]
                          {literal-1   }             {literal-2   }

      INTO identifier-4 [, DELIMITER IN identifier-5] [, COUNT IN identifier-

[, identifier-7 [, DELIMITER IN identifier-8][, COUNT IN identifier-9] ] ...

      [WITH POINTER identifier-10] [TALLYING IN identifier-11]

      [; ON OVERFLOW imperative-statement]
```

```
USE AFTER STANDARD {EXCEPTION} PROCEDURE ON {file-name-1 } [ , file-name-2 ] ...
                   {ERROR    }              {INPUT     }
                                            {OUTPUT    }
                                            {I-O       }
                                            {EXTEND    }
```

```
USE FOR DEBUGGING ON {cd-name-1                          }
                     {[ALL REFERENCES OF] identifier-1   }
                     {file-name-1                        }
                     {procedure-name-1                   }
                     {ALL PROCEDURES                     }

    [ cd-name-2                                ]
    [ [ALL REFERENCES OF] identifier-2         ]
  , [ file-name-2                              ]      ... .
    [ procedure-name-2                         ]
    [ ALL PROCEDURES                           ]
```

```
WRITE record-name [FROM identifier-1 ]
    [ {BEFORE} ADVANCING {integer                 } ]
    [ {AFTER }           {identifier-2 {LINE }    } ]
                         {             {LINES}    } ]
                         {PAGE                    } ]
                         {TAB                     } ]
      [; AT {END-OF-PAGE} imperative statement ]
            {EOP        }
```

```
WRITE record-name FROM identifier

      [;INVALID KEY imperative-statement]
```

GENERAL FORM FOR COPY STATEMENT

```
COPY      "text-name"
```

GENERAL FORMAT FOR CONDITIONS

Relation condition:

```
{identifier-1          }  {IS [NOT] GREATER THAN}  {identifier-2          }
{literal-1             }  {IS [NOT] LESS THAN   }  {literal-2             }
{arithmetic-expression-1}  {IS [NOT] EQUAL to   }  {arithmetic-expression-2}
{index-name-1          }  {IS [NOT] >           }  {index-name-2          }
                          {IS [NOT] <           }
                          {IS [NOT] =           }
```

Class Condition:

```
identifier IS [NOT]  {NUMERIC     }
                     {ALPHABETIC  }
```

Sign Condition:

```
arithmetic-expression IS [NOT]  {POSITIVE}
                                {NEGATIVE}
                                {ZERO    }
```

Condition-name Condition:

```
condition-name
```

Switch-status Condition:

```
condition-name
```

Negated Simple Condition:

```
NOT simple-condition
```

Combined Condition:

```
condition  {{AND} condition}  ...
           {{OR }          }
```

Abreviated Combined Relation Condition:

```
relation-condition{{AND} NOT [relational-operator]        object}...
                  {{OR }                                      }
```

MISCELLANEOUS FORMATS

QUALIFICATION:

```
{data-name-1     }  [{OF}        data-name-2 ]  ...
{condition-name  }   {IN}

paragraph-name   [{OF}  section-name ]
                 {IN}

text-name        [{OF}  library-name ]
                 {IN}
```

SUBSCRIPTING:

```
{data-name     }
{condition-name}   (subscript-1  [, subscript-2  [, subscript-3] ] )
```

INDEXING:

```
{data-name     }(  {index-name-1  [± literal-2]}
{condition-name}   {literal-1                  }

  [, {index-name-2  [± literal-4]}  [, {index-name-3  [± literal-6]}  ]]
     {literal-3                  }      {literal-5                  }
```

IDENTIFIER: FORMAT 1

```
data-name-1  {OF}  data-name-2  ...  [(subscript-1  [, subscript-2
             {IN}

     [, subscript-3]]  )]
```

IDENTIFIER: FORMAT 2

```
data-name-1  [{OF}  data-name-2 ]  ...  [(  {index-name-1  [ ± literal-2]}
             {IN}                       {literal-1                     }

    , [{index-name-2  ± literal-4}  , [{index-name-3  [ ± literal-6]}  ]] )]
      {literal-3                 }     {literal-5                    }
```

Appendix G

Summary of Extensions to ANSI COBOL

Apple III COBOL is oriented to microcomputer users with the system close at hand and usually with a CRT. Apple III COBOL therefore provides extensions for interactive working, program control of files, text file handling and rapid development and testing. These facilities are summarised below.

SCREEN FORMATTING AND DATA ENTRY

THE ACCEPT STATEMENT

An additional format for the ACCEPT statement is provided as follows:

Format

```
ACCEPT  data-name-1 [AT {data-name-2} ]  FROM CRT
                       {literal-1   }
```

data-name-2 allows the start of screen to be changed dynamically. It refers to a PIC 9999 field where the most significant 99 is a line count 1-25 and the least significant 99 is a character position 1-80.

data-name-1 refers to a record, group or elementary item but may not be subscripted.

literal-1 is an alphanumeric literal

NOTE: See Chapter 3 for description. See also Appendix H for Environment Division changes.

THE DISPLAY STATEMENT

An additional format for the DISPLAY statement is provided as follows:

Format

```
DISPLAY {data-name-1} [AT {data-name-2}]  UPON { CRT      }
        {literal-3  }      {literal-1  }       { CRT-UNDER }
```

literal-3 is an alphanumeric literal

data-name-1 refers to a record, group or elementary item but may not be subscripted

data-name-2 defines the left-most position on the screen. It refers to a PIC 9999 field where the most significant 99 is a line count 1-25 and the least significant 99 is a character position 1-80.

NOTE: See Chapter 3 for description.

DISK FILES

Two extensions are offered by Apple III COBOL file processing; these are as follows:

1. Line sequential files
2. Run time input of filenames

LINE SEQUENTIAL FILES

When LINE SEQUENTIAL ORGANIZATION is specified in the FILE CONTROL paragraph ORGANIZATION IS entry, the file is treated as consisting of variable length records separated by the line delimiter characters. Trailing spaces in output records are replaced by a Carriage Return and a Line Feed character.

RUN TIME INPUT OF FILENAMES

The ASSIGNed name in the SELECT statement for a file is processed on OPENing as follows:

When the INPUT or OUTPUT phrase is specified, execution of OPEN causes checking of the file names in accordance with the operating system connections for opening on input or output file. The full operating system features for file reallocation and device control are therefore available to the Apple III COBOL program.

LOWER CASE CHARACTERS

The full alphanumeric lower case a to z is available in Apple III COBOL. Reserved and user word characters are read as their upper case equivalents (A to Z).

HEXADECIMAL VALUES

Hexadecimal binary values can be attributed to non-numeric literals in Apple III COBOL by expressing them as X "xx", where x is a hexadecimal character in the set 0-9, A-F; xx can be repeated up to 120 times, but the number of hexadecimal digits must be even.

INTERACTIVE DEBUGGING

There is an Animation Option to provide sophisticated debugging facilities in the user's program. Programs may be run from the start with or without the Animation Option until a specified break-point is reached, when control is passed back to the user. At this point, data areas or program logic may be inspected or changed.

The Animation Option is entered as an option by the user and the user program is then monitored and tested line by line, paragraph by paragraph and so on as required. The commands to the package can reference procedure statements and data areas directly against each line of the compilation listing as displayed. Powerful macros of commands can be used to give very sophisticated debugging facilities. The precise details for using the Animation Option are described in the Apple III COBOL Introduction and Operating System Manual.

Appendix H

System Dependent Language Features

This Appendix summarises those parts of a COBOL program that need to be changed to run them as Apple III COBOL programs and those parts that do not need changing specifically but are ignored by the Apple III COBOL compiler when generating the object program.

MANDATORY CHANGES

ENVIRONMENT DIVISION

The only statements in the environment division that must be specialised for Apple III COBOL are shown below:

Configuration Section

SPECIAL-NAMES. special names entry

special names entry must include the following:

CURSOR IS data-name-1

The CURSOR IS dat]-name-1 clause specifies the data-name which will contain the CRT cursor address as used by ACCEPT statements. Data-name-1 must be declared in the Working-Storage section as a 4 character item. The interpretation of the 4 characters is given in the ACCEPT statement description.

Input-Output Section

File names must be as described in Appendix F of the Apple III COBOL Introduction and Operating System Manual.

STATEMENTS COMPILED AS DOCUMENTATION ONLY

COBOL programs not specifically written for compilation as Apple III COBOL on microcomputers can still be compiled. Statements using features that are not available are treated as documentary only, and are not compiled. A summary of these features follows:

ENVIRONMENT DIVISION

I-O-Control Paragraph

The clauses that refer to a real time clock and magnetic tape in this paragraph are ignored by the compiler during compilation but do not cuase compile times errors. These clauses are as follows:

END OF $\left\{\begin{matrix}\text{REEL}\\\text{UNIT}\end{matrix}\right\}$ of file-name-2 (no magnetic tape)

```
integer-2 CLOCK UNITS                (no clock)
```

DATA DIVISION

File Description Paragraph

The following complete statements in the file description are ignored by the compiler during compilation but do not cause compile time errors:

```
BLOCK  CONTAINS   integer-1  TO  integer-2

                                { RECORDS    }
                                { CHARACTERS }

CODE-SET  IS  alphabetic-name

LABEL     { RECORD  IS  }   { STANDARD }
          { RECORDS ARE }   { OMITTED  }

VALUE OF  implementor-name-1  IS  literal-1
        [,implementor-name-2  IS  literal-2] ...
```

PROCEDURE DIVISION

CLOSE Statement

The following phrases in the CLOSE statement are ignored by the compiler during compilation but do not cause compiler-time errors:

```
{ REEL }                             (No magnetic tape)
{ UNIT }
```

Appendix I

Language Specification

Apple III COBOL is ANSI COBOL as specified in "American National Standard Programming Language COBOL" (ANSI X3.23 1974). The Apple III COBOL Implementation has been selected from both levels of ANSI COBOL. The following modules are fully implemented at Level 1:

- Nucleus
- Table Handling
- Sequential Input and Output
- Relative Input and Output
- Indexed Input and Output
- Segmentation
- Library
- Inter-Program Communication
- Debug
- Sort Merge

In addition the following modules are fully implemented at level 2:

- Nucleus
- Table Handling
- Sequential Input and Output
- Relative Input and Output
- Indexed Input and Output
- Inter-Program Communication
- Sort Merge

The full level 2 syntax of the Communications module is accepted at this release but the Run-time System does not yet include the capability to execute it.

This appendix specifies the implementation of Apple III COBOL. The implementation of each of the eight standard COBOL modules listed above is given under the following headings as applicable:

Level 1 Implementation
Level 2 Implementation
Apple III COBOL Extensions

Appendix F in this manual is an Apple III COBOL syntax summary.

NUCLEUS

Level One Implementation

Fully implemented to Level Two.

Level Two Implementation

Fully implemented to Level Two.

Apple III COBOL Extensions

1. Lower case letters a to z are read as upper case letters A to Z.

2. Hexadecimal binary values can be attributed to non-numeric values by expressing literals as X"nn".

3. Reserved word SPACE can be used to clear the whole CRT screen.

4. The ANSI switch unset enables omission of certain ANSI required "red tape" paragraphs and statements.

5. COMPUTATIONAL-3 or COMP-3 can be specified in the USAGE clause to specify packed internal decimal storage, (BCD).

6. ACCEPT data-name-1 [AT {data-name-2 / literal-1}] FROM CRT

   gives enhanced CRT input features

7. DISPLAY {data-name-1 / literal-1} [AT {data-name-2 / literal-2}] UPON {CRT / CRT-UNDER}

   gives enhanced CRT output facilities.

8. 'CURSOR IS data-name' can be specified in SPECIAL-NAMES and 'data-name' in WORKING-STORAGE section to specify CRT cursor address for ACCEPT statements

9. The function names SYSIN, SYSOUT and TAB can be assigned to user specified mnemonic-names in the SPECIAL NAMES paragraph. SYSIN and SYSOUT are equivalent to ACCEPT and DISPAY from and to CONSOLE respectively. TAB is used with the WRITE statement to printer to throw a page. A directive is available in the compiler command line to alter these function names if they are already used in your COBOL program for other purposes.

   In addition the following IBM type extensions are incorporated. Extensions numbered 1 and 2 below are always active unless the ANSI switch directive is set in the compiler command line. Extension

number 3 takes effect only when the directive IBM is used in the compiler command line.

1. Redefinition of data names need not be the same length - the compiler reserves the largest area.

2. Level numbers need not be specified in sequence. Thus:
   01 - - - -
   03 - - - -
   02 - - - -
   will be valid - with 03 being treated as if it were 02.

3. Introduction of FILLER group items.

SEQUENTIAL, RELATIVE AND INDEXED I-O

Level One Implementation

Fully implemented to Level One.

Level Two Implementation

Fully implemented to Level Two.

Apple III COBOL Extensions

1. Run Time allocation of file-names. See Appendix F in Operating Guide.

2. LINE SEQUENTIAL is an additional file type.

3. All File Description (FD) clauses are optional when the ANSI switch is unset.

4. Tabbing is available, specified by TAB in the WRITE statement. (See note 9 under NUCLEUS(Apple III COBOL Extensions) above.

TABLE HANDLING

Level One Implementation

Fully implemented to Level One.

Apple III COBOL Extensions

Fully implemented to Level Two.

SEGMENTATION

Level One Implementation

Fully implemented to Level One.

LIBRARY

Level One Implementation

Fully implemented to Level One.

DEBUG

Level One Implementation

Fully implemented to Level One plus an additional Interactive Run-Time Debug package.

Apple III COBOL Extensions

A powerful Run-Time Debug package is available. See the Apple III COBOL Introduction and Operating System Manual.

INTER-PROGRAM COMMUNICATION

Level One Implementation

Fully implemented to Level One.

Level Two Implementation

Fully implemented to Level Two.

SORT-MERGE

Level One Implementation

Fully implemented to Level One.

Level Two Implementation

Fully implemented to Level Two.

COMMUNICATIONS

Level One Implementation

Full Level One syntax accepted and checked.

Level Two Implementation

Full Level Two syntax accepted and checked.

EXISTING RESTRICTIONS

The Apple III COBOL Run Time System does not yet include the capability to execute the Communications module.

Appendix J

IBM Extensions

The following IBM extensions are implemented in the Full Apple III COBOL product:

1. "ID" is a synonym for "IDENTIFICATION".

2. In redefinition of data names the areas need not be the same size: the compiler allocates space for the largest.

3. Level number rules are relaxed so that they need not be declared in ascending sequence e.g.:

```
01  ...
    03  ...
    02  ...  is allowed
```

4. FILLER items can be grouped e.g.:

```
    03  FILLER
        04  ...
```

These extensions are set to apply by the IBM directive in the compiler command line (see the Apple III COBOL Introduction and Operating System Manual).

Appendix K

Table of Possible MOVEs in a COBOL Program

Category of Sending Data Item		Category of Receiving Data Item		
		Alphabetic	Alphanumeric, Alphanumeric Edited	Numeric Integer, Numeric Non-Integer, Numeric Edited
ALPHABETIC		Yes	Yes	No
ALPHANUMERIC		Yes	Yes	Yes
ALPHANUMERIC EDITED		Yes	Yes	No
NUMERIC	INTEGER	No	Yes	Yes
	NON-INTEGER	No	No	Yes
NUMERIC EDITED		No	Yes	Yes

Appendix L

Table of MOVE Data Categories

Category of MOVE	Contents of Source Field*	PICTURE of Destination Field	Result in Destination Field*
Alphabetic to Alphabetic or to Alphanumeric-edited	ABCDE	A(6) or X(6) A(5) or X(5) A(2) or X(2)	ABCDE ABCD AB
Alphabetic to Alphanumeric-edited	ABCDE	XXBXX XXBXXX	AB^CD AB^CDE
Alphanumeric to Alphabetic or to Alphanumeric-edited	ABCDE	A(6) or X(6) A(5) or X(5) A(4) or X(4)	ABCDE ABCDE ABCD
Alphanumeric to Alphanumeric-edited	ABCDE	XXBXXX XXBXX	AB^CDE AB^CD
Alphanumeric to Numeric-edited	1980	99.99 $$$$9	80.00 $1980
Alphanumeric to Numeric	1980	9999	1980
Numeric-edited to Alphanumeric	$12.34	X(7) X(6) X(5)	$12.34 $12.34 $12.3
Numeric-edited to Alphanumeric-edited	$12.34	XXXBXXXX XXXBXXX XXXBX	$12^.34 $12^.34 $12^.
Numeric integer to Alphanumeric	1234	X(5) X(4) X(3)	1234 1234 123
Numeric integer to Alphanumeric-edited	1234	XBXXXX XBXXX XBXX	1^234 1^234 1^23
Alphanumeric-edited to Alphanumeric-edited	PIC = XXBXXX AB^CDE	XX/XXXX XX/XX X/XXXX	AB/^CDE AB/^C A/B^CD
Alphanumeric-edited to alphabetic or to alphanumeric	PIC = XXBXXX AB^CDE	A(6) or X(6) A(5) or X(5) A(7) or X(7)	AB^CDE AB^CD AB^CDE

Category of MOVE	Contents Source Field*	PICTURE of Destination Field	Result Destination Field*
Numeric-integer to numeric	1980	9999	
Numeric-integer to Numeric-edited	1980	99.99 $$$99	80.00 $1980
Numeric non-integer (floating point) to numeric	19.76	9999	0019
Numeric non-integer to numeric-edited	12.34	$99.99 $$.9	$12.34 $2.3
MOVE SPACE TO ITEM-B	-	A(4) or X(4) XX/XXX	^^^^ ^^/^^^
MOVE ZERO TO ITEM-B	-	X(4) XXBXX	0000 00^00
MOVE QUOTE TO ITEM-B	-	A(6) or X(6) XXBXXXX XB/XX	""""""""" """^"""""" "^"""
MOVE 12 to ITEM-B	-	XXX X/X	12 1/2
MOVE ZERO TO ITEM-B	-	XXX X/X	000 0/0
MOVE ZERO TO ITEM-B (Numeric MOVE)	-	$$9	$0
MOVE 12 TO ITEM-B (Numeric MOVE)	-	$$9 $9	$12 $2

* ^ = one space, i.e., one depression of the SPACE bar; physical spaces are included in this table for clearer presentation only.

Appendix M

Table of Valid PICTURE Clause Sequences

EDIT TYPE	PICTURE	DATA: 0	.0/2	2	1234.5	-123.4
		RESULTING FORM:				
No editing	9999	0000	0000	0002	1234	0123
Floating Insertion	ZZ99 ZZZZ .ZZ ZZ.ZZ	00	00 .02 .02	02 2 .00 2.0	1234 1234 .50 34.50	123 123 .40 23.4
	99 **.	**00 **.**	**00 **.02	**02 *2.00	1234 34.50	*123 23.40
	$$99 $$$$$ $$$$.99 $$$$.$$	$00 $.00	$00 $.02 $.02	$02 $2 $2.00 $2.00	$234 $1234 $234.50 $234.50	$123 $123 $123.40 $123.40
	+9999 9999+ -9999 ++++9 ----- ++++.++	+0000 0000+ 0000 +0	+0000 0000+ 0000 +0 +.02	+0002 0002+ 0002 +2 2 +2.00	+1234 1234+ 1234 +1234 1234 +1234.50	-0123 0123- 0123 -123 -123 +123.40
Fixed Insertion	99B99 9,999 9B9/900 $99CR $999DB	00 00 0,000 0 0/000 $00 $000	00 00 0,000 0 0/000 $00 $000	00 02 0,002 0 0/200 $02 $002	12 34 1,234 2 3/400 $34 $234	01 23 123 1 2/300 $23CR $123DB
Fixed and Floating Insertion	Z,ZZZ *,***.** -,---.-- $$$BCR ZZ.9DB +$999 $***9 $ZZZ9	*****.** .0 +$000 $***0 $ 0	*****.02 .0 +$000 $***0 $ 0	2 ****2.02 2.00 $2 2.0 +$002 $***2 $ 2	1,234 1,234.50 1,234.50 $34 34.5 +$234 $1234 $1234	123 **123.40 -123.40 $23 CR 23.4DB -$123 $*123 $ 123

Note that the forced insertion of fixed editing characters takes precedence over data characters.

Appendix N

SET Statement Valid Operations

SET TO FORMAT		TO			
		Integer Literal			
			Integer data item		
				Index Name	
					Index Data Item
Identifier-1	Integer data item				X
	Index data item			X	X
Index-name-1		X	X	X	X

		Index data item			
			Elementary numeric integer		
				Greater than 0	
					Optional sign
SET TO Format	Identifier-1, -2	X	X		
	Integer-1			X	X
SET { UP / DOWN } BY Format	Identifier-3		X		
	Integer-2				X

Appendix O

Permissible I-O Statements and File OPEN Modes

ACCESS METHOD	STATEMENT	I-O MODULE									
		SEQUENTIAL OPEN MODE				RELATIVE OPEN MODE			INDEXED OPEN MODE		
		INPUT	OUTPUT	I-O*	EXTEND	INPUT	OUTPUT	I-O	INPUT	OUTPUT	I-O
Sequential	READ	X		X		X		X	X		X
	WRITE		X		X		X			X	
	REWRITE			X				X			X
	START					X		X	X		X
	DELETE							X			X
Random	READ					X		X	X		X
	WRITE						X	X		X	X
	REWRITE							X			X
	START								X		X
	DELETE							X			X
Dynamic	READ					X		X	X		X
	WRITE						X	X		X	X
	REWRITE							X		X	X
	START					X		X	X		X
	DELETE							X			X

* This OPEN mode not supported for ORGANIZATION LINE SEQUENTIAL

Figures and Tables
Volume 1

Figures and Tables

Volume 2

Index

Pages in Volume 2 are shown in [].

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

T

U

V

W

X

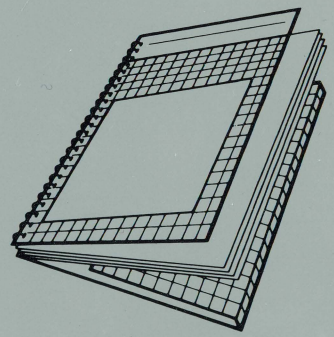
Y

Z

Apple III

Apple III COBOL: Language Reference Manual Volume 2

Tuck end flap
inside back cover
when using manual.



apple computer
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576
030-0437-A

Photographs courtesy of Vorpal Gallery, San Francisco, New York, Laguna Beach, CA.