Honeywell

SERIES 6000/600

SOFTWARE

dataBASIC SYSTEM LANGUAGE MANUAL

Honeywell

SERIES 6000/600

dataBASIC SYSTEM LANGUAGE MANUAL

SUBJECT:

The dataBASIC System Language Considerations Including an Introduction to the Language, General System Characteristics, Statements, Control Commands, Subsystems, Methods of Communicating with the dataBASIC System, and Summaries of Language Statements and Replacement Expressions.

DATE:

May, 1971

ORDER NUMBER: DA08

Rev. 0

PREFACE

This manual is a reference guide for using Honeywell's Series 6000/600 dataBASIC language. Section I explains the basic function of the language and lists the notations used in programs. Section II defines the dataBASIC program in terms of elements and data structure in files, records, and fields; it is followed by Section III which describes and illustrates each program in one or more typical program constructions. Section IV, entitled Advanced Data Selection, provides instructions for creating generalized procedures so as to be reusable in other applications. Section V explains the control commands for disposition of programs; Section VI describes the dataBASIC system in terms of its subsystems; and Section VII outlines procedures for communicating with the system via a remote Teletype terminal. For quick reference, dataBASIC language statements, along with expressions and replacements, are summarily tabulated in Appendix A, while terminal commands and reserved words are alphabetically listed in Appendices B and C. In Appendix D terminal error messages are tabulated; Appendix E contains guidelines to be followed when trying for higher programming efficiency and more effective selection of file loading techniques. Finally, Appendix F contains a block diagram of the Series 6000/600 dataBASIC file structure.

Within the text of this manual, all references to Series 6000 systems are applicable to 600 systems unless otherwise stated.

Other Honeywell publications related to the dataBASIC system include the following titles and document order numbers:

dataBASIC Load/Unload System, Order No. DA09
GECOS Time-Sharing System General Information Manual, Document No. CPB-1643
Comprehensive Operating Supervisor, Document No. CPB-1518
Integrated Data Store (I-D-S), Document No. CPB-1565
File and Record Control, Document No. CPB-1003

The dataBASIC system is a coded system designed to extend the power of Series 6000 in the area of data control. It is supported by comprehensive documentation and training; periodic program maintenance and, where feasible, improvements are furnished for the current version of the system, provided it is not modified by the user.

© 1971, Honeywell Information Systems Inc.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I
INTRODUCTION

GENERAL SYSTEM DESCRIPTION

Honeywell's dataBASIC system provides for data base management and inquiry by combining data base manipulation capabilities with a BASIC type language. It permits a file to be constructed, maintained, retrieved, and deleted on a content-addressable basis. Records of any size, containing from one to hundreds of fields, may be created completely without record descriptions. The records are, in fact, self-described and processed on the basis of field names and values which are supplied by the user at the time of record storage.

dataBASIC SYSTEM APPLICATIONS

The dataBASIC system has many applications, the few listed below are suggestive of many other potential uses:

Real estate listings, single or multiple, where an inquiry might ask for all houses of Spanish architecture, having four bedrooms and a pool, and located in the Arcadia School district.

Personnel files, where an inquiry might involve all single, male programmers having FORTRAN application experience, and whose last rate change preceded January 1, 1966.

A public service/police file, where a request might involve all 1968 white Chevrolets registered with Maricopa County and owned by individuals with a previous criminal record.

A medical index of symptoms and diseases to aid in the diagnoses of illnesses.

A library index, which permits access to documents on the bases of subject, author, title, citations, publisher, and date of publication.

THE dataBASIC LANGUAGE

In its basic form, the dataBASIC language is a procedural language for record storage, retrieval, and display; additional programming capabilities, conditional and unconditional transfers, and subroutine functions complete all the requirements for the advanced dataBASIC system user. User learning time for the dataBASIC language is minimized; a small set of control words call upon all the basic functions of the system. These words make up the dataBASIC language.

DOCUMENT NOTATIONS

In the documentation of the dataBASIC language, standard English notations are used with the following restricted meanings.

Reserved words

Reserved words for the dataBASIC language are in the upper case and are underlined. A reserved word must be spelled exactly as it appears in a statement definition, and its use is restricted to the function defined for it. It cannot, for example, be used as field:name.

Example: $\underline{\text{COPY}}$

Braces

{ }   denote alternatives one of which must be selected

Example

$$\underline{\text{STORE}} \quad \begin{Bmatrix} \underline{\text{RECORD}} \\ \underline{\text{COPY}} \end{Bmatrix}$$

means that the user may state STORE RECORD or STORE COPY.

Brackets

[ ]   denote an option

Example

$$\left[\underline{\text{NOT}}\right]$$

means that the user may elect to use the $\underline{\text{NOT}}$ option at his own discretion.

Ellipses

. . . denotes optional repetition of the preceding expression, which is delimited by the immediately preceding set of braces or brackets.

Example

$\underline{\text{PRINT}} \quad \left[\text{value}\right] \quad \ldots$

means the user may substitute one or more values.

SECTION II
GENERAL SYSTEM CHARACTERISTICS

THE dataBASIC PROGRAM

A dataBASIC program consists of an ordered set of statements that instruct a computer to solve a problem. In its simplest form it contains the data (values) to be worked on, formulae which tell the computer what to do with these numbers, and input/ output statements which tell the computer where to get the data and what to do with the answers.

THE dataBASIC STATEMENT

A dataBASIC system statement comprises four parts in the following sequence.

line:number

The line:number has one to five digits and serves two purposes: (1) It is used as a sequence control within the program, specifying the order in which statements are to be executed; and (2) it uniquely identifies a statement. The line:number cannot contain imbedded spaces but must be followed by one or more spaces.

control:word

The control:word tells the dataBASIC system what function is to be performed. (Control words are listed in Appendix A of this manual.)

all other words

All other words are written in accordance with the specifications for each control:word.

carriage:return character

The carriage:return character denotes the end of a line; the dataBASIC system responds with a line:feed and prints an asterisk (*) when it is ready to accept the next line of input. Characters following the 72nd character of a line will not be used.

dataBASIC LANGUAGE ELEMENTS

Elements of the dataBASIC language include both alphanumeric and numeric literals and the words rfield and wsfield.

Alphanumeric leterals contain letters, numbers, punctuation marks, etc., and may contain all characters (except "@" and data transmission control characters). They are enclosed within quotation marks. An example of an alphanumeric literal is "JOE JONES."

Numeric literals consist only of the numbers 0-9 written without quotation marks, and may contain an embedded decimal point. In absence of an embedded decimal point, the numeric literal is considered an integer; in the absence of a sign, the numeric literal is assumed to be plus. Examples of numeric literals are 1492, 1967, 8.97, 0.0001.

The word rfield denotes user-assigned record field:names. The field:name may consist only of the letters A thru Z, the numbers 0 thru 9, and the special character. An rfield must start with two alphabetic characters.

The word wsfield denotes user-assigned working storage names. The user may specify one or several working storage fields to be used for temporary storage purposes within the dataBASIC program. They may be used to store a field name, field value, or the results of an arithmetic or functional operation. Working storage field names are one alphabetic character (A-Z) in length and are established by virtue of their use. A working storage field whose content is to be used as a field:name, as opposed to a field:value, is indicated by an ampersand (&) suffix on the working storage field, e.g., A&. In either case, reference is to the same wsfield; only the use of its content is being declared to be different. Whenever the contents of a working storage field is changed by a dataBASIC program statement, the working storage assumes the size of the new content.

DATA STRUCTURE

Data consists of information which is stored in the file as fields of records and which will remain there until deleted under user-program-control.

Data Files

All records stored under a specified dataBASIC file:name form a logically separate structure; they can only be retrieved for maintenance and reporting by programs referred to by that file:name. A file:name is from one to eight alphanumeric characters in length and may contain the letters A thru Z, the digits 0-9, and the special characters - and . . There may be no embedded spaces in the file:name.

Terminals or inquiry stations can access a specific file:name, but only if properly authorized. One terminal may access several files, but a given program can access only one file.

A file may contain any number of different record types, that is, records which contain different fields.

Data Records

The dataBASIC record serves to collect and hold a set of field:names and associated field:values. These fields describe an entity, or a person, event, place, thing, concept, or whatever else the user desires. The data concerning this entity is the record, and it is stored so that it is available for subsequent retrieval and processing.

dataBASIC records have no predefined format or content. They consist merely of the collection of fields with which they are associated at the moment. It is this flexibility that allows for multiple record types. Data records themselves have no names and are known solely in terms ot their content of data fields. Fields may be stored in, or deleted from, a record at any time, as long as at least one field continues to exist within the record. When the last field of a record is deleted, the record ceases to exist.

The concept of "current record" is important because many dataBASIC language functions are based on the existence of a current record and operate on this record and its fields. The following two statements are the only statements in the system which can make a record the current record:

1. The STORE RECORD, a statement which creates a new record and makes it the current record.
2. The FOR record, a process which selects and makes current (one at a time) all the records satisfying the selection criteria.

There are only three situations during processing in which there is no current record. The first is at the beginning of processing when no STORE RECORD or FOR record statements have been processed to create a current record; the second is after a DELETE RECORD statement has deleted the current record; the third is after the EXIT from a FOR record selection process, when there was no current record prior to the FOR record process. An attempt to execute a statement requiring a current record when no such record exists will result in LINE XXXXX NO CURRENT RECORD being printed at run time and retreat being performed.

Data Fields

record field:name

The user of the dataBASIC system may select and establish his own field:names. A field:name may not exceed twenty-four (24) characters in length. A field:name may be expressed using rfield or wsfield&. rfield denotes the expression itself to be used as the field:name. An example would be the field:name MAKE in

```
100 STORE MAKE "DATSUN"
```

wsfield& denotes that the content of the working storage specified is to be used as the field:name. The following two statements would achieve the same result as did the above statement:

```
090 LET B = "MAKE"
100 STORE B& "DATSUN"
```

record field:value

The field:value to be stored in a record may be one to twenty-four (24) characters in length. A field:value may be expressed using rfield, wsfield&, wsfield or literal.

rfield specifies a field:name whose corresponding field:value is to be used as the value. For example, the rfield MAKE in

```
100 LET A = MAKE
```

places the value of the field:name MAKE in the wsfield A. If the field:value for the field name MAKE had been "DATSUN" wsfield A would have contained "DATSUN" after the statement had been executed.

wsfield& indicates that the field:value for a working storage field whose content specifies a field:name is to be used as a value. The following two statements would achieve the same result as in the above example for rfield:

```
090 LET B = "MAKE"
100 LET A = B&
```

wsfield denotes that the content of the specified wsfield will be used as the value; for example, the content of B would be used as the value in the statement:

```
100 STORE MAKE B
```

Literal denotes that the expression itself will be used as the value. For example, the literal "DATSUN" is used as the value in the statement:

```
100 STORE MAKE "DATSUN"
```

multiple field:values with same field:name

The dataBASIC system permits multiple field:values to be stored for a specified field:name. There is no limit to the number of field:values associated with a particular field:name; nor is there any requirement that the field:values be logically consistent.

Duplicates

Any number of duplicate field:names and/or field:values may be contained in a given record; moreover, a file may contain any number of duplicate records. The existence or non-existence of duplicate records and/or fields is controlled by the user.

Special Convention

Through the use of unique text:numbers, the user may store, modify, and delete descriptive information concerning different records. For example,

```
100 STORE TEXT:001 "TEXT LINE ONE"
110 STORE TEXT:002 "TEXT LINE TWO"
```

Text:numbers and text:values differ from field:names and field:values in that they cannot be used as selection criteria and that a line of text may be as long as 60 characters.

SECTION III
dataBASIC LANGUAGE STATEMENTS

DECLARATION STATEMENTS

THE DATA STATEMENT

The DATA statement enables the user to store within the dataBASIC program data he would like to use during the execution of his program. The statement itself is never executed; it only supplies a stream of data for READ statements which ask for data.

The DATA statement can include as many literals as can be contained completely within one line, but there can be multiple DATA statements. Care should be taken against having an odd number of quotation marks within one DATA statement. This causes scan termination of the line.

Notation:

line:number DATA { literal } ... cr

Example:

```
900 DATA "BUICK" "RIVIERA" "AIR-COND" cr
```

Data Storage and Maintenance

THE FILE STATEMENT

Each file has a name which is unique to a user:number. Any number of files may exist under one user:number. Associated file passwords are asked for at execution time. A file is created or deleted at the system command level (See Section V, Control Commands), but declaring the file as being used is accomplished within the program. A program can access only one file.

Notation:

line:number FILE file:name cr

Example:

```
100 FILE CARLOT    (cr)
```

This statement causes the dataBASIC system to request passwords from the user at the time the program is run. The passwords are established at the time the file is created. If the user supplies invalid passwords, he is not allowed access to the file.

DATA SELECTION STATEMENTS

THE FOR STATEMENT

The FOR record statement provides a technique for processing, one at a time, all records which meet a selection criteria, or all the field:values within a record having the same field:name. Details of record and field selection follow. (A maximum of ten nested FOR statements is allowed.)

Accessing All File Records

THE FOR ALL STATEMENT

All records in the file may be accessed for processing by use of the FOR ALL statement. Records are accessed, one at a time; and for each record accessed, the system executes the statements nested between the FOR ALL and the NEXT statements. Records are accessed in reversed storage sequence; that is, the newest record is accessed first.

Notation:

```
line:number  FOR ALL    (cr)
                .
                .
                .
line:number  NEXT    (cr)
```

Example:

```
          100 FOR ALL    (cr)
                .
                .
                .
          500 NEXT    (cr)
```

Accessing Selected Records

The dataBASIC system provides the FOR record statement to selectively access records so that their data fields may be used for subsequent processing. This statement is based on the content addressability of the dataBASIC data records and data fields.

The FOR record statement initiates the record accessing process and, with the matching NEXT statement, defines the limits of processing for a selected record. Records accessed for processing are those records within the file whose field content is consistent with the relational:expression, any number of which may be specified to delimit the records to be accessed within the file.

THE NEXT STATEMENT

As each record is accessed for processing, it becomes both the current record and the object of all record and record field operations included within the action statements. When the NEXT statement is encountered, the record ceases to be the current record, and another record is selected. After the last record is processed, control is transferred to the statement following the NEXT statement.

Simple Condition: This version of the FOR record statement allows the selection of records which contain the specified field:name/field:value pair. A maximum of 25 simple conditions are allowed within any relational expression.

Notation:

$$\text{line:number} \quad \underline{\text{FOR}} \quad \left[\underline{\text{NOT}}\right] \quad \left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{wsfield}\\ \text{literal}\end{array}\right\} \quad \text{relational:operator} \quad \left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{wsfield}\\ \text{literal}\end{array}\right\} \quad \textcircled{cr}$$

$$\text{line:number} \quad \underline{\text{NEXT}} \quad \textcircled{cr}$$

relational:operator may be any of the following:

= (which means equal to)
> (which means greater than)
< (which means less than)
> = or = > (which means greater than or equal to)
= < or < = (which means less than or equal to)
> < or < > (which means not equal to)

Example:

```
100 FOR MAKE = "FORD"   cr
        .
        .
        .
200 NEXT   cr
```

All records which contain a field:name/field:value of MAKE/FORD will be selected.

```
100 FOR NOT MAKE = "FORD"   cr
        .
        .
        .
200 NEXT   cr
```

In this example, all records which do not contain a field:name/field:value of MAKE/FORD will be selected.

Range Condition: This version of the FOR record statement allows the selection of records which have a field either within a range or not within a range.

Notation:

$$\text{line:number}\ \ \underline{\text{FOR}}\ \left[\underline{\text{NOT}}\right]\left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{literal}\\ \text{wsfield}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{literal}\\ \text{wsfield}\end{array}\right\}\ \underline{\text{TO}}\ \left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{literal}\\ \text{wsfield}\end{array}\right\}\ \ \text{cr}$$

.
.
.

$$\text{line:number}\ \ \underline{\text{NEXT}}\ \ \text{cr}$$

Example:

```
100 FOR YEAR FROM "1966" TO "1968"    (cr)
        .
        .
        .
200 NEXT    (cr)
```

Using this example, all records which contained a field:value for the field:name YEAR that is in the range of 1966 to 1968 will be selected for further processing within the FOR record statements.

Value Selection: With this version of the FOR record statement it is possible to select all records which contain a specific field:value regardless of associated field:name.

Notation:

```
                                    ┌ rfield    ┐
line:number FOR [NOT]  ALL          │ wsfield&  │     (cr)
                  .                 │ literal   │
                  .                 └ wsfield   ┘
                  .
line:number   NEXT    (cr)
```

Example:

```
100 FOR ALL  "RED"    (cr)
       .
       .
       .
200 NEXT   (cr)
```

The above example will cause all records containing a field:value to be presented to the program. If we use the previous used car lot example, RED might appear in the field:name COLOR and UPHOLSTERY:COLOR.

Name Selection: With this version of the FOR statement it is possible to select all records containing a specific field:name regardless of the associated field:values.

Notation:

```
line:number  FOR  [NOT]  {wsfield&}  ALL    (cr)
                         {rfield  }
              .
              .
              .
line:number   NEXT  (cr)
```

Examples:

```
100 FOR COLOR ALL  (cr)
      .
      .
      .
200 NEXT  (cr)
```

All records containing a field:name COLOR, regardless of the field:value, will be selected.

```
100 FOR NOT COLOR ALL  (cr)
      .
      .
      .
200 NEXT  (cr)
```

All records not containing a field:name COLOR will be selected. This option could be used to check for records in a file not having a mandatory (as defined by the user) field.

Compound Record Selection: The dataBASIC system allows a user to specify multiple conditions as a criteria for record selection. These conditions can be any of the simple: conditions. In order for a record to be selected, all conditions must be met. Conditions are entered on a one-per-line basis. The first line contains the FOR action statement, and all subsequent lines contain the AND statement.

Notation:

```
line:number FOR simple:condition     cr
line:number AND simple:condition     cr
                 .
                 .
                 .
line:number AND simple:condition     cr
                 .
                 .
                 .
line:number NEXT   cr
```

Example:

```
100 FOR YEAR FROM "1966" TO "1968"    cr
110 AND MAKE = "FORD"   cr
120 AND NOT BODY = "STATION WAGON"    cr
130 AND HORSEPOWER FROM 392 TO 406    cr
         .
         .
         .
200 NEXT   cr
```

This statement will cause the dataBASIC system to search the file for records describing Fords made in model years 1966 thru 1968, but which are not station wagons and whose horsepower ranges from 392 to 406.

Complex Record Selection: At times it may be desirable to select records based upon unrelated selection criteria. This flexibility is allowed for by use of the "OR" action statement. The OR connects two or more compound:conditions which consist of one or more simple conditions.

Notation:

```
line:number FOR compound:condition   (cr)
line:number OR  compound:condition   (cr)
             .
             .
             .
line:number OR  compound:condition   (cr)
             .
             .
             .
line:number NEXT  (cr)
```

Example:

```
100 FOR MAKE = "CHEVROLET"  (cr)
110 AND MODEL = "IMPALA"  (cr)
120 OR MAKE = "PONTIAC"  (cr)
130 AND MODEL = "BONNEVILLE"  (cr)
     .
     .
     .
200 NEXT  (cr)
```

In this example all records which described Chevrolet Impalas or Pontiac Bonnevilles will be selected.

```
100 FOR MAKE = "DATSUN"  (cr)
110 OR MAKE = "TOYOTA"  (cr)
120 AND OPTIONS = "AIR-COND"  (cr)
130 AND NOT TRANSMISSION = "3-SPEED STICK"  (cr)
     .
     .
     .
200 NEXT  (cr)
```

Using this example, all records describing MAKE = "DATSUN", regardless of any other field in the record, and all records with MAKE = "TOYOTA" that have air-conditioning but do not have a standard 3-speed transmission, will be selected. The criteria OPTIONS = and NOT TRANSMISSION = do not apply to MAKE = "DATSUN". If those criteria should apply, they will have to be repeated for the "DATSUN" selection.

```
100 FOR COLOR = "WHITE"  cr
110 OR OPTIONS = "AIR-COND"   cr
      .
      .
      .
200 NEXT  cr
```

The above example illustrates a set of criteria which may not be mutually exclusive; that is, a record which is selected by the criteria COLOR = "WHITE" could also be selected for OPTIONS = "AIR-COND" if a white car with air-conditioning exists on the file. However, the same record will not be selected more than once within each selection process.

Field Selection

The dataBASIC system allows the association of an unlimited number of field:values with a given field:name. Process statements use only one occurrence of a field:value unless each is selected separately. Selection is accomplished using the FOR name statement. It should be noted that a current record must have been previously selected.

Notation:

```
                       ⎧ rfield   ⎫
line:number FOR        ⎩ wsfield& ⎭    cr
                .
                .
                .
line:number NEXT   cr
```

This version of the FOR statements does not select a "current" record. Instead, it is always subordinate to (and logically within) a FOR record statement, as described earlier in this section of the language manual. When used it will serially select all the field:values of a field:name which are part of the current record.

Example:

```
          [ 100 FOR MAKE = "BUICK"    (cr)
          [ 110 AND OPTIONS = "AIR-COND"   (cr)
part 1    [       .
          [       .
          [       .
          [ 150 FOR OPTIONS   (cr)
          [       .
part 2    [       .
          [       .
          [ 200 NEXT   (cr)
          [       .
part 3    [       .
          [       .
          [ 300 NEXT   (cr)
```

Part 1 specified the criteria for record selection (a "BUICK" with "AIR-COND") and the statements to be processed for the current record prior to Part 2. Part 2 specifies that each field:value for OPTIONS is to be processed. Because all FOR statements must have a NEXT statement, the NEXT statement appears in Part 2 (which refers to the FOR in Part 2) and the NEXT in Part 3 (which refers to the FOR in Part 1). Statements contained in Part 1 and Part 3 will be executed once for each record selected; whereas statements in Part 2 will be executed once for each value of OPTIONS in the current record. Parts 1, 2, and 3 all refer to the same current record.

Dictionary Functions

The dataBASIC language provides the ability to access the dictionary (i.e., field:names and field:values) independent of records. This selection differs from record selection in that there is "no current record" resulting from the dictionary selection functions. Instead, field:names and field:values are made current and are available for use in record selection criteria, or vice versa.

Selection of field:name

THE FOR FNAME STATEMENT

The FOR FNAME statement allows selection of the field:names within a file.

Notation:

```
                                 ALL
                                                          rfield
line:number FOR FNAME            relational:operator      wsfield&        (cr)
                                                          wsfield
                                                          literal

                                 FROM  rfield     TO  rfield
                                       wsfield&       wsfield&
                                       wsfield        wsfield
                                       literal        literal
```

The ALL option is used to select all field:names within a file. The relational operator option allows selection of field:names based upon their relation to a record rfield, working storage field, or literal. The valid relational:operators are:

= (which means equal to)

> (which means greater than)

< (which means less than)

= > (which means greater than or equal to)

< = (which means less than or equal to)

> < or < >(which means not equal to)

The FROM option allows selection of field:names within the specified range. The current field:name is available to the user and is contained in the system working storage field named FNAME.

Selection of field:value

THE FOR FVALUE STATEMENT

The FOR FVALUE statement allows selection of the field:values which are associated with the current field:name. This use requires a field:name having been made current by the FOR FNAME process. The formats available are the same as those available to the FOR FNAME statement.

Notation:

$$\text{line:number }\underline{\text{FOR}}\ \underline{\text{FVALUE}}\quad\left\{\begin{array}{l}\text{ALL}\\[2ex] \text{relational:operator}\qquad\left\{\begin{array}{l}\text{rfield}\\ \text{wsfield\&}\\ \text{wsfield}\\ \text{literal}\end{array}\right\}\\[2ex] \text{FROM}\left\{\begin{array}{l}\text{rfield}\\ \text{wsfield\&}\\ \text{wsfield}\\ \text{literal}\end{array}\right\}\ \underline{\text{TO}}\left\{\begin{array}{l}\text{rfield}\\ \text{wsfield\&}\\ \text{wsfield}\\ \text{literal}\end{array}\right\}\end{array}\right\}\quad\text{(cr)}$$

The ALL option is used to make available all values associated with the current field:name.

The relational:operator option is used to make available all field:values associated with the current field:name based upon their relation to a record field, working storage field, or literal. The valid relational:operators are the same for FVALUES as they are for FNAME.

The FROM options allow range selection. The current field:value is available to the user in the system working storage field:named FVALUE.

Example:

This example shows a way to print all the names along with the values associated with those names, for the entire file.

```
100 FILE USED CAR  (cr)
110 FOR FNAME ALL  (cr)
120 PRINT "FIELD NAMEƀ" FNAME  (cr)
130 FOR FVALUE ALL  (cr)
140 PRINT "VALUEƀ" FVALUE  (cr)
150 NEXT  (cr)
160 PRINT  (cr)
170 NEXT  (cr)
180 END  (cr)
```

The above example would produce a report that might look, in part, like the following:

```
FIELD NAME MAKE
      VALUE "FORD"
      VALUE "PLYMOUTH"
      VALUE "DODGE"
      VALUE "TOYOTA"
          .
          .
          .
      VALUE "BUICK"
FIELD NAME OPTIONS
      VALUE "AIR-COND"
          .
          .
          .
```

Uses of FNAME and Value: The value-references FNAME and FVALUE can be used only in the "FOR FNAME" and "FOR FVALUE" statements, and in the two statements which follow:

1. LET wsfield = $\left\{\begin{matrix}\text{FNAME}\\ \text{FVALUE}\end{matrix}\right\}$

2. PRINT $\left\{\begin{matrix}\text{FNAME}\\ \text{FVALUE}\end{matrix}\right\}$

DATA MANIPULATION STATEMENTS

Data Deletion

THE DELETE RECORD STATEMENT

The current record of the program will be deleted from the file only when a "DELETE RECORD" statement is executed. After the deletion is complete there is no current record; and statements which require the current record as an operand will cause LINE XXXXX NO CURRENT RECORD to be printed and a retreat to be performed.

Notation:

line:number DELETE RECORD (cr)

Example:

```
100 DELETE RECORD (cr)
```

Field Deletion

THE DELETE STATEMENT

A dataBASIC record field:value is deleted by using the DELETE name value statement. This statement operates only on the current record and will delete either (1) the field specified by field:name and field:value; (2) all the field:values for a specified field:name; or (3) all fields which contain a specific field:value.

Notation:

$$\text{line:number}\qquad \underline{\text{DELETE}} \left\{ \begin{array}{l} \left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\end{array}\right\} \qquad \left[\begin{array}{l}\text{wsfield}\\ \text{literal}\end{array}\right] \qquad \ldots \\ \\ \underline{\text{ALL}} \quad \left\{\begin{array}{l}\text{wsfield}\\ \text{literal}\end{array}\right\} \end{array} \right\} \quad \textcircled{\scriptsize cr}$$

Note: field:value references are restricted to wsfield and literal for the DELETE statement.

Example:

200 DELETE ALL "FORD" $\textcircled{\scriptsize cr}$

This statement deletes all field:names and field:values which contain the specific field: value "FORD".

200 DELETE MAKE $\textcircled{\scriptsize cr}$

This statement deletes the field:name "MAKE" and all field:values for that field:name.

200 DELETE MAKE "PONTIAC" "OLDS" $\textcircled{\scriptsize cr}$

This statement deletes the field MAKE PONTIAC and the field MAKE OLDS.

Data Modification

THE FIX STATEMENT

Modification of field:values for a given field:name of the current record is accomplished by using the FIX statement. The FIX statement provides the capability of modifying all or a specific field:value which is associated with a field:name. This is the <u>only</u> statement which will modify the contents of an existing file. An attempt to modify field values of a non-existent field:name will result in no action.

Notation:

$$\text{line:number} \quad \text{FIX} \quad \begin{Bmatrix}\text{rfield}\\ \text{wsfield}\underline{\&}\end{Bmatrix} \quad \begin{bmatrix}\text{wsfield}\\ \text{literal}\end{bmatrix} \quad = \quad \begin{Bmatrix}\text{wsfield}\\ \text{literal}\end{Bmatrix} \quad \textcircled{cr}$$

Note: field:value references are restricted to wsfield and literal for the FIX statement.

Example:

100 FIX COLOR = "RED" $\textcircled{cr}$

The above example will delete all but one field:value with field:name COLOR from the current record and will set the one remaining field:value equal to RED.

200 FIX MAKE "VOLKSWAGEN" = "VW" $\textcircled{cr}$

The above example will change the field MAKE VOLKSWAGEN in the current record to MAKE VW, and will not affect other field:values.

THE LET STATEMENT

The general format of this statement is LET wsfield = expression. The LET statement enables the user to temporarily hold or manipulate values in working storage fields during the execution of his program. There are 26 working storage fields available, denoted by the letters A through Z. Note that only working storage field names may appear to the left of the equal sign in this statement. If it is necessary to change the value of a record field, then the FIX statement must be used.

In general, the content of working storage fields is completely under the user's control. It is he who must assign initial values, make changes as needed, and make use of the working storage field values. The exception to this are fields which are used in the "LET wsfield = SUM/MIN/MAX" statements. These are initialized to null at the beginning of the program and when first referenced within a FOR loop. However, if no records are selected by the FOR loop, the wsfield will contain, after execution of the loop, whatever it had going into the loop.

Assignment Expressions

Simple assignment: The simplest form of the LET statement is that which sets the value of a working storage field equal to the value of another field. This is the only form of the LET statement that can be used with working storage fields containing names.

Notation:

$$\text{line:number } \underline{\text{LET}} \quad \text{wsfield} \quad = \left\{ \begin{array}{l} \text{literal} \\ \text{rfield} \\ \text{wsfield} \\ \text{wsfield}\underline{\&} \end{array} \right\} \quad \textcircled{\text{cr}}$$

If value is a literal, the literal itself will be moved to the working storage field. If value is a record field or another working storage field, then the contents of the field will be moved.

Example:

```
    100 LET A = 100      (cr)
    110 LET B = "FORD"   (cr)
```

When the previous statements were executed, the contents of A would have been made equal to the numeric value 100, and B would have been made equal to the alphanumeric value "FORD".

```
    100 LET A = COLOR    (cr)
```

This statement would cause the contents (or field:value) of the field:name COLOR of the current record to be moved to the working storage field A. If the current record contained more than one field:value for the field:name COLOR, A would contain the first value referenced. All other values of a multi-valued field would have been ignored.

```
    100 LET B. = A    (cr)
```

The contents of working storage field A would be moved to working storage field B.

Null Assignment: In order to allow a field to be reset to a NULL value, a special form of the LET statement has been provided. Working storage fields are initially set to null during compilation.

Notation:

$$\text{line:number } \underline{\text{LET}} \text{ wsfield } = \underline{\text{NULL}} \quad \textcircled{\text{cr}}$$

Example:

```
    100 LET A = NULL    (cr)
```

Minimum/Maximum Assignment: A special form of the LET statement has been designed to facilitate the determination of the minimum or maximum field:value.

Notation:

$$\text{line:number LET wsfield} = \left\{\begin{matrix}\underline{\text{MIN}}\\ \underline{\text{MAX}}\end{matrix}\right\} \quad \left\{\begin{matrix}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{wsfield}\end{matrix}\right\} \quad \textcircled{cr}$$

This assignment statement will reset the specified working storage field value to null value when first encountered within a repetition of a FOR sequence. The field:value of the field:name specified is compared with the current value in the working storage field. If value is an rfield or wsfield&, and there is no field:name as specified, then no action is taken. If the named field has multiple field:values within the record, then all field:values are compared.

The contents of the working storage field holding the minimum/maximum value is accessible at any time within the selection/action process. The final value is accessible at the completion of the process.

Example:

100 LET A = MIN YEAR $\textcircled{cr}$

Dictionary Assignment: The following form of the LET statement is provided to assign the current field:name and/or field:value for the file to working storage. If the statement is encountered and there is no current field:name and/or field:value, then a null value is assigned.

Notation:

$$\text{line:number } \underline{\text{LET}} \text{ wsfield} = \left\{\begin{matrix}\underline{\text{FNAME}}\\ \underline{\text{FVALUE}}\end{matrix}\right\} \quad \textcircled{cr}$$

Example:

100 LET A = FNAME $\textcircled{cr}$

In the above example, the working storage field A includes the current field:name contained in FNAME after execution of statement 100.

Arithmetic Expression

Formula Assignment: The LET statement also permits arithmetic operations where data values may be added, subtracted, multiplied divided or exponentiated with the resulting value stored in the specified working storage field. Only one arithmetic process is allowed in each LET statement.

Notation:

$$\text{line:number } \underline{\text{LET}} \text{ wsfield } = \left\{\begin{matrix}\text{wsfield}\underline{\&} \\ \text{rfield} \\ \text{wsfield} \\ \text{literal}\end{matrix}\right\} \left\{\begin{matrix} + \\ - \\ * \\ / \\ \wedge \end{matrix}\right\} \left\{\begin{matrix}\text{wsfield}\underline{\&} \\ \text{rfield} \\ \text{wsfield} \\ \text{literal}\end{matrix}\right\} \quad \textcircled{cr}$$

Valid arithmetic operators are:

+	add
-	subtract
*	multiply
/	divide
∧	raise to the power of

Example:

To calculate interest based on the formula interest = principal x rate x time, these steps would be necessary:

100 LET A = PRINCIPAL * RATE $\textcircled{cr}$

110 LET B = A * TIME $\textcircled{cr}$

Working storage location B would then contain the interest.

Summary Assignment: A special form of the LET statement has been designed to facilitate the accumulating or summing of field:values.

Notation:

$$\text{line:number } \underline{\text{LET}} \text{ wsfield } = \underline{\text{SUM}} \left\{\begin{matrix}\text{rfield} \\ \text{wsfield}\underline{\&} \\ \text{wsfield} \\ \text{literal}\end{matrix}\right\} \quad \textcircled{cr}$$

If the value is an rfield or wsfield&, and there is no field with the specified field: name within a selected record, then no action is taken. If there are multiple fields with the specified field:name within a selected record, each field is added. If wsfield is specified, the contents of that field is added to the sum field each time the LET statement is encountered. If a literal is specified, the literal is added to the sum field each time the LET statement is encountered.

Example:

100 LET A = SUM SALES (cr)

The record field:name SALES will be added to the working storage field named A.

LET A = SUM 1 (cr)

The above example will cause the working storage field A to be incremented by one (1) each time the statement is executed.

Functions Assignment: The LET statement allows for use of special arithmetic functions. These functions include absolute value, integer, random number generation and negate.

Notation:

$$\text{line:number} \quad \underline{\text{LET}}\ \text{wsfield} \ = \left\{\begin{matrix} \underline{\text{ABS}} \\ \underline{\text{INT}} \\ \underline{\text{RND}} \\ \underline{\text{NEG}} \end{matrix}\right\} \left\{\begin{matrix} \text{wsfield} \\ \text{literal} \\ \text{wsfield}\underline{\&} \\ \text{rfield} \end{matrix}\right\} \quad (\text{cr})$$

Example:

100 LET A = ABS NET:PROFIT (cr)

In the above example, the working storage value A is assigned the absolute value of the field:value whose field:name is NET:PROFIT.

100 LET A = INT COST (cr)

In the above example, the working storage field A is assigned the integer value of the field:value whose field:name is COST.

```
100 LET A = RND 1.5    (cr)
```

In this example the working storage field A is assigned a system-generated random number. The value specified (in this case, the literal 1.5) is used as a base for the number generation.

```
100 LET A = NEG B    (cr)
```

In this example the working storage field A is assigned the algebraic negative of the content of working storage field B.

Data Storage

THE STORE RECORD STATEMENT

A record is stored by the recognition and execution of a STORE RECORD statement, and the storing in that record of one or more data fields by execution of the STORE name value statement. The execution of the STORE RECORD statement makes the record created the current record of the program, which may then be accessed to store additional data fields, or to print, or delete, or perform any other relevant action. If no STORE name value statements appear after a STORE RECORD, the newly created record is automatically deleted.

Notation:

line:number STORE RECORD (cr)

Example:

```
100 STORE RECORD    (cr)
```

THE STORE COPY STATEMENT

A copy of the current record is stored by the recognition and execution of a STORE COPY statement. The execution of the STORE COPY statement makes the record stored the current record of the program.

Notation:

line:number STORE COPY (cr)

Example:

```
100 STORE COPY    (cr)
```

Field Storage

THE STORE STATEMENT

A new field may be added to the current record by execution of a STORE name value statement. The fields of a dataBASIC record are defined by a field:name and the field: values associated with that field:name. The first field defined is either the record field: name (rfield) or a working storage field which contains the record field:name (wsfield&). All other fields are either the values assigned (literal) or contain the value (wsfield).

Notation:

$$\text{line:number} \quad \underline{\text{STORE}} \left\{ \begin{matrix} \text{rfield} \\ \text{wsfield}\underline{\&} \end{matrix} \right\} \left\{ \begin{matrix} \text{wsfield} \\ \text{literal} \end{matrix} \right\} \ldots \quad \text{(cr)}$$

Note: field:value references are restricted to wsfield and literal for the STORE statement.

Example:

```
090 STORE RECORD (cr)
100 STORE YEAR 1967 (cr)
110 STORE MAKE "VOLKSWAGEN" (cr)
120 STORE C& "BLUE" (cr)
130 STORE COLOR "WHITE" "BLUE" (cr)
140 STORE OPTIONS "RADIO" "HEATER" (cr)
```

INPUT/OUTPUT STATEMENTS

Terminal Input

THE INPUT STATEMENT

The INPUT Statement enables the user to input variable data without changing the dataBASIC program. If while a program is executing an INPUT statement is encountered, the dataBASIC system will type a question mark (?) and wait for the user to type in the needed data. Any number of fields that can be specified in one line of the dataBASIC program can be input with one INPUT statement. Data is always INPUT to working storage field and is entered in the form of literal. Data can be either field:names or field:values.

The INPUT statement is usually used with the PRINT statement, which tells the user what information is expected by the program.

Notation:

line:number <u>INPUT</u> {wsfield} . . . (cr)

Example:

```
     010   PRINT "ENTER MAKE, MODEL, OPTION"   (cr)
     050   INPUT  A B C   (cr)
     100   FOR MAKE = A   (cr)
     110   AND MODEL = B   (cr)
     120   AND OPTIONS = C   (cr)
     130   PRINT "MODEL REQUESTED AVAILABLE WITH";   (cr)
     140   PRINT "THESE OPTIONS"   (cr)
     150   FOR OPTIONS   (cr)
     160   PRINT OPTIONS EDIT "BBXXXXXXXXXXXXXX"   (cr)
     170   NEXT   (cr)
     180   NEXT   (cr)
```

Statement 010 prints instructions to the user and is immediately followed by the INPUT command. As soon as the system types the "?", the user can enter his data, which would appear as:

```
     ? "BUICK" "RIVIERA" "AIR-COND"  (cr)
```

Output of this example would be the same as for the preceding example. If fewer fields are entered than were requested, the system will respond with the message "LINE 050 DATA FORMAT ERROR", and then request that all data be input again.

<u>Printing</u>

THE PRINT STATEMENT

The PRINT statement may be used for the following five purposes:

1. To display the contents of a specified record field or working storage field
2. To display all field:names and field:values of a record
3. To display literals
4. To start at the beginning of a new print line
5. To perform a combination of the above

The operand of the PRINT statement is called a print:expression; a PRINT statement may have one, none, or multiple print:expressions.

PRINT statements containing only literals and/or working storage references are record-independent and may be executed anywhere within the program.

PRINT statements containing print:expressions followed by the word RECORD or by record field:names operate on field:value within the current record. An attempt to execute such a record-dependent PRINT statement when the current record of the program is undefined will result in the transmission of the message, "LINE XXXXX NO CURRENT RECORD" the program is then terminated and a retreat is performed.

Print format control editing may be left entirely to the dataBASIC system or may be specified by the user-program. The PRINT RECORD always causes printing in the unedited mode. All other print:expressions may be printed in either the edited or unedited mode.

When printing in the unedited mode, character strings are followed by two space characters, except for literals which have no spaces following. Alphanumeric field: values are printed enclosed in quotation marks.

THE PRINT RECORD STATEMENT

The PRINT RECORD statement causes the system to display all field:names and field:values of the current record. If the current record is undefined, the message, "LINE XXXXX NO CURRENT RECORD" is transmitted to the user, the program is aborted, and a retreat is performed. If the current record has been established, it is printed according to the following four rules:

1. The first field:name is printed in column 5 of a new print line.
2. Field:names are followed by two blank characters.
3. Multiple field values with the same field:name are separated by a comma followed by a blank character.
4. Four blank characters separate the last field:value of a field:name from the next field:name.

Notation:

line:number PRINT RECORD

Example:

```
    100 PRINT RECORD   (cr)
```

Output from such a statement might be

```
     MAKE "TOYOTA"  MODEL "CORONA"
OPTIONS "AIR-COND",  "AM-FM",  "BUCKET SEATS"
YEAR "1967"  COLOR "BLACK",  "WHITE"
TRANSMISSION "STANDARD"  SALES:PRICE
1050.00
```

Print Record Field

This form of PRINT statement causes the system to display all the field:values in the current record which are associated with the specified record field:name. If no field:value is present, the system prints the character string, "NO VALUE" in lieu of the field:value. If multiple field:values are present, each is printed, being separated by a comma and single space. Spacing between the field:value(s) displayed and the output of prior and successive print expressions is controlled by the rules listed under "Horizontal Spacing" and "Vertical Spacing" in this section of the Language Manual. (See pages 3-28 and 3-29.)

Notation:

line:number PRINT $\left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\end{array}\right\}$ (cr)

Examples:

```
     100 PRINT OPTIONS   (cr)
     110 PRINT A&   (cr)
```

Output of statement 100 above might be:

```
     "AIR-COND",  "AM-FM",  "BUCKET-SEATS"
```

If A& contained the field name "OPTIONS", output would look like that of the preceding statement.

Print Literal

This form of the PRINT statement allows the user to display a literal exactly as shown in the program. If the literal is alphanumeric (that is, displayed in the user program within quotation marks), the dataBASIC system prints the exact character string without quotation marks. Numeric literals are printed exactly as they appear in the PRINT statement. No space characters are produced by the dataBASIC system following literals.

Notation:

line:number <u>PRINT</u> literal (cr)

Example:

```
    100 PRINT "THIS IS AN ALPHANUMERIC LITERAL"  (cr)
    110 PRINT  1.0    (cr)
    120 PRINT  1      (cr)
```

Output of above would be

```
    THIS IS AN ALPHANUMERIC LITERAL
    1.0
    1
```

<u>Print Working Storage Field</u>

This form of the PRINT statement causes the system to display the contents of a working storage field. Working storage fields are those established by a LET, INPUT, or READ statement. If there is no data currently stored in the specified working storage field, the system prints the character string, "NO VALUE", in lieu of the values.

Notation:

line:number <u>PRINT</u> $\left\{\begin{array}{l}\text{wsfield}\\ \text{wsfield}\underline{\&}\end{array}\right\}$ (cr)

Example:

```
    100 PRINT A    (cr)
    110 PRINT A&   (cr)
```

If A contained the field name OPTIONS, the output would be as follows:

```
    "OPTIONS"
    "AIR-COND",  "AM-FM"
```

<u>Special Print Convention</u>

The PRINT statement may also be used to display all or selected text fields associated with any record. If there are no text fields associated with a record, the character string "NO VALUE" will be printed.

Notation:

line:number <u>PRINT</u> $\left\{\begin{array}{l}\text{text}\\ \text{text:number}\end{array}\right\}$ (cr)

Example:

```
    100 PRINT TEXT      (cr)
    110 PRINT TEXT:001     (cr)
```

Output of Line 100 above would be all the text associated with the current record; and the output of Line 110 would be the first line of text for the current record.

Print with Edit

The print expressions referring to record field:names or working storage field names may be printed under user-supplied EDIT format control at the users option. Whenever an EDIT format is supplied, it completely governs the printed character string. No space characters are added before or after the edited field. If an rfield to be edited is not found in the current record, or if a working storage field has null value, space characters will be printed for the length of the edit format. If it is known that multiple values exist for a specific field and it is desired to print each value edited, they may be selected by means of the FOR name statement. An edit format must be enclosed in quotation marks.

Notation:

$$\text{line:number } \underline{\text{PRINT}} \left\{\begin{array}{l}\text{rfield}\\ \text{wsfield}\underline{\&}\\ \text{wsfield}\\ \text{literal}\end{array}\right\} \quad \underline{\text{EDIT}} \text{ format:expression} \quad (\text{cr})$$

Example:

```
    100 PRINT A EDIT   "+99.9"     (cr)
```

If A contained the number 7, the output would be:

```
    +07.0
```

Numeric Editing

The following characters are valid within a format:expression for a numeric field.

B	(means insert one space character.)
9	(means replace with one numeric character.)
Z	(means replace with one space character if a leading zero, otherwise replace with one numeric character.)
.	(means insert a decimal point.)
,	(means insert a comma unless it lies immediately to the right of a space character, in which case insert a space character.)

- (means print a space character if the field being edited is positive, and a - if negative. This must be the first character of the edit format.)

+ (means print a + if the field being edited is positive, and a - if negative. This must be the first character of the edit format.)

The data field is decimal-aligned with the edit format. Truncation or addition of zero-valued characters is performed as specified by the edit format. If no decimal place is specified in either the edit format or in the data, the dataBASIC system assumes that the format or data is an integer for purposes of decimal alignment.

Examples:

DATA	EDIT FORMAT	RESULT
20	B99.9B	b̸20.0b̸
2	B99.9B	b̸02.0b̸
.02	B99.9B	b̸00.0b̸
120.4	B99.9B	b̸20.4b̸
2.134	BZZ.ZB	b̸b̸2.1b̸
0	BZZ.ZB	b̸b̸b̸b̸b̸b̸
null-value	BZZ.ZB	b̸b̸b̸b̸b̸b̸
.02	BZZ.ZB	b̸b̸b̸b̸b̸b̸
.1	BZZ,ZZ9.9B	b̸b̸b̸b̸b̸b̸0.1b̸
105	BZZ,ZZ9.9B	b̸b̸b̸b̸105.0b̸
1157.79	BZZ,ZZ9.9B	b̸b̸1,157.7b̸
100000	BZZ,ZZ9.9B	b̸b̸b̸b̸b̸b̸0.0b̸

Alphanumeric Editing

The following characters are valid within a format:expression for an alphanumeric field:

B (means insert one space character)

X (means replace with one alphanumeric character)

The data is left-justified and inserted into the character positions specified by X's. If there is more data than X format characters, the field is truncated on the right. If there are more X format characters than data, spare characters replace any excess X format characters.

Example:

DATA	EDIT FORMAT	RESULT
MARCH	BXXXB	b̸MARb̸
JØ	BXXXB	b̸JØb̸b̸
MAR191969	BXXXBSSBSSSSB	b̸MARb̸19b̸1969b̸

Horizontal Spacing

The following rules apply to horizontal spacing.

PRINT RECORD RULES:

1. The first field of the record will be printed in column 5 of the first line.
2. The field name will be printed, followed by two space characters, followed by the field:value(s).
3. Multiple field:values for one field:name will be printed, being separated by a comma and a space character.
4. Four space characters will separate the last (or only) field:value from the next field:name.

PRINT LITERAL RULES:

1. Numeric literals will be printed exactly as they are entered in the PRINT statement.
2. Alphanumeric literals will be printed exactly as shown within the quotation marks (the quotation marks will not be printed).
3. No space characters will be used to separate literals from the field following, if any.

PRINT UNEDITED rfield, wsfield, or wsfield:

1. The contents of such fields, if alphanumeric, will be printed enclosed within quotation marks.
2. The space characters will be used to separate an unedited field from the following field.
3. If an rfield to be printed is not contained within the current record, the message "NO VALUE" will be printed.

PRINT EDITED RULES:

1. No space characters will be used to separate an edited field from the following field.
2. If an rfield to be printed is not contained within the current record, or if a wsfield is null-valued, then space characters will occupy all positions of the edit format.

Vertical Spacing

A semicolon (;) is used to continue a print statement (i. e. , there will be no carriage return or line feed). If it terminates with no punctuation or with punctuation other than a semicolon, a line feed and carriage return will be generated.

A print statement alone (that is one with no print:expression) will cause a carriage return and line feed to be transmitted. This results in feeding paper to the next line.

If the dataBASIC system finds that the remaining space on the print line is not large enough to contain the data to be printed, it will insert a carriage return and line feed before printing that data. This spacing is in addition to, rather than in lieu of, user-specified vertical spacing.

THE READ STATEMENT

Whenever a READ statement is encountered, the literal values are moved from the DATA statement to the working storage field(s) specified in the READ statement. As many values will be moved as there are wsfield in the READ statement. If there are multiple READ statements, or if a READ statement is logically executed more than once, new literal values will be supplied for each statement as long as there are sufficient literals defined in the DATA statement(s). If there are multiple DATA statements, values will be taken from the first statement until it is depleted; then from the second, and so on, until all have been used. If a READ is executed after all the data has been used, the system will display the message "LINE XXXXX OUT of DATA", and then stop (no retreat occurs).

Notation:

line:number READ { wsfield } ... ⓒⓡ

Example:

010 READ A B C ⓒⓡ

Now let us suppose that the used car lot sales manager has written a program, to be used by all his salesmen, that will select all available cars of a specific make, model, and specific option, and that will print all options available with the car. The sales manager has stored this program, and it is available for use by everyone. This program might appear as:

```
005  FILE USED:CAR  (cr)
010  READ  A B C  (cr)
020  IF NOT A = "NONE" THEN 100  (cr)
030  PRINT "ENTER DATA IN LINE 200 THEN RUN"  (cr)
040  STOP  (cr)
100  FOR MAKE = A  (cr)
110  AND MODEL = B  (cr)
120  AND OPTIONS =  (cr)
130  PRINT "MODEL REQUESTED AVAILABLE WITH THESE";  (cr)
140  PRINT "OPTIONS"  (cr)
150  FOR OPTIONS  (cr)
160  PRINT OPTIONS EDIT "BBXXXXXXXXXXXXXXX"  (cr)
170  NEXT  (cr)
180  NEXT  (cr)
190  STOP  (cr)
200  DATA "NONE"  "NONE"  "NONE"  (cr)
210  END  (cr)
```

Statement 010 reads the data specified in statement 200. If the data has not been changed, the message "ENTER DATA IN LINE 200 THEN RUN" is printed and the program stops. Suppose the statement 200 DATA "BUICK" "RIVIERA" "AIR-COND" was entered. After execution of the read, A will contain BUICK, B will contain RIVIERA, and C will contain AIR-COND. The FOR statement and the related AND statements then select the records with the proper make, model and option; and statements 130 and 140 print a message which serves to indicate the start of a new car. Statements 150 through 170 select and process each option, and statement 180 delimits the record selection process.

Output of this example might appear as follows:

```
MODEL REQUESTED AVAILABLE WITH THESE OPTIONS
      AIR-COND
      AM-STEREO-FM
      BUCKET-SEATS
```

Now let us suppose that the DATA statement read:

```
200 DATA  "BUICK"  "RIVIERA"  (cr)
```

As soon as the READ in statement 010 tried to read data into working storage field C, the program would stop with the message "LINE 200 OUT OF DATA" displayed to the user.

If the data statement was changed to

```
        200 DATA "BUICK"  "RIVIERA"  "AIR-COND"  "AM-FM"    (cr)
```

and the program was executed, the last literal, "AM-FM", would be ignored because no READ statement calls upon it.

CONTROL STATEMENTS

Program Termination

THE END STATEMENT

The END statement is the last statement of the program and defines the end of program. It has the same effect as a STOP statement.

Notation:

```
line:number  END    (cr)
```

Example:

```
    99999   END    (cr)
```

Conditional Termination of Selection

THE EXIT STATEMENT

The EXIT statement enables the user to discontinue processing anywhere within a record or field selection statement. The statement immediately following the NEXT statement, which delimits the record or field selection process, receives control when the EXIT statement is encountered.

This statement cannot logically be replaced by a GOTO statement because EXIT also insures proper handling of FOR statement termination.

Notation:

line:number EXIT (cr)

Example:

```
100 FOR MAKE = VOLVO (cr)
110 AND OPTIONS = "AIR-COND" (cr)
120 PRINT RECORD (cr)
130 IF OPTIONS = "AM-FM" (cr)
140 AND YEAR "1967" THEN 160 (cr)
150 GO TO 170 (cr)
160 EXIT (cr)
170 NEXT (cr)
180 END (cr)
```

In the above example, printing will continue for all Volvo cars having air conditioners; however, after the printing of a Volvo model later than 1967, printing and processing will stop.

Subroutines

Within any computer program, as within any plan of action, there are procedures to be executed at many different points within the main procedure. For example, a banking system may, during the daily posting, check account numbers for validity, perform standard procedures whenever an overdraft occurs, compute service charges, and so on. The procedures for validity checking, overdrafts, and service charges may be executed at various points within the daily posting procedure and, on any given day, may be executed once, many times, or not at all. Such standardized procedures are efficiently handled in a computer program through the use of subroutines. Subroutines are computer procedures that may be called from anywhere within the total procedure; and, at the end of a subroutine execution, processing is resumed at the statement immediately following the one that called for the subroutine execution.

It should be noted that a procedure becomes a subroutine only when it is called through the execution of a GOSUB statement. A subroutine execution is terminated when a RETURN statement is detected in the sequence of statements being executed.

THE GOSUB STATEMENT

The GOSUB statement of the dataBASIC language provides for calling and entering a subroutine. It acts as an unconditional branch or GOTO statement, in that processing resumes by executing the statement specified by the statement identifier. However, the dataBASIC system records the line number of the GOSUB statement for future use. This line number will be used to return to the sequence of statements following the GOSUB statement when a RETURN statement is detected during subroutine execution. The listing of wsfields after the line number operand will cause the wsfield content to be saved, and will make the specified wfields available for use within the subroutine.

Notation:

line:number GOSUB   line:number {wsfield ...}   (cr)

Example:

In the following example, the used car salesman has written a program such that the selection and output are separate parts of the program, thus making it easy to change one part.

```
100 FILE USED:CAR  (cr)
110 GOSUB  200  (cr)
130 STOP  (cr)
200 FOR MAKE = "VOLKSWAGEN"  (cr)
205 IF COLOR = "RED"  THEN 220  (cr)
210 GOSUB 300  (cr)
220 NEXT  (cr)
230 RETURN  (cr)
300 PRINT SERIAL: NO OPTIONS  (cr)
310 RETURN  (cr)
400 END  (cr)
```

THE RETURN STATEMENT

The RETURN statement marks the end of a procedure when it is executed as a subroutine. Whenever the dataBASIC system encounters a RETURN statement, it resumes sequential processing at the line immediately following the GOSUB statement which called the subroutines. It then restores the original value of wsfields declared in the GOSUB statement. If the procedure is being executed as a result of sequential processing rather than a subroutine call, there is no GOSUB statement to return to, and the RETURN statement is ignored.

Notation:

line:number RETURN (cr)

Conditional Restoring of a File

THE RETREAT STATEMENT

The RETREAT statement enables the user to restore a file to its status at the beginning of a particular run, thus allowing him to enter modifications to his file, to check modifications against expected results, to assure accuracy, and to RETREAT should he elect to simulate a particular condition or locate an error in logic.

Notation:

line:number RETREAT (cr)

Example:

```
100 FILE USED:CAR (cr)
110 FOR MAKE = "VOLKSWAGEN" (cr)
120 OR MAKE = "TOYOTA" (cr)
130 OR MAKE = "DATSUN" (cr)
140 OR MAKE = "VOLVO" (cr)
150 OR MAKE = "OPEL" (cr)
160 STORE SPECIAL  "FOREIGN" (cr)
170 PRINT MAKE SPECIAL (cr)
180 NEXT (cr)
190 PRINT "IF RESULTS OK ENTER OK" (cr)
200 INPUT  A (cr)
210 IF A = "OK" THEN 900 (cr)
220 RETREAT (cr)
900 END (cr)
```

When this program was entered, an undetected error was made (line 160 should be, STORE SPECIAL "FOREIGN"). However, upon execution, the error becomes apparent because the field SPECIAL is printed, and no value "FOREIGN" appears. Therefore, when the message "IF RESULTS OK ENTER OK" appears, the user enters a value other than "OK", and the file is restored to its status prior to the start of that run. The user can then correct his program and re-enter the new field properly.

Branching

The dataBASIC system provides for database management and inquiry. Its basic repertoire of statements enables the user to perform all necessary data storage and maintenance functions as well as all data selection functions necessary for inquiry. The system processes statements in the order indicated by the statement numbers. Certain more complex problems, however, cannot be adequately handled by the sequential statement execution; rather they require decision making capabilities outside the data selection or storage process. They also require the capability of altering the order in which the statements are being executed, either conditionally depending on the outcome of a decision process, or unconditionally.

THE GOTO STATEMENT

This statement allows the user to unconditionally go to another part of his program and resume execution. The line:number referenced must be found within the user's program.

Notation:

```
line:number GOTO line:number     (cr)
```

Example:

```
     100 INPUT  A    (cr)
     110 PRINT  A    (cr)
     120 GOTO   100    (cr)
```

In this example, line 120 is a GOTO statement which causes the execution of 100 and 110 to be repeated.

THE IF STATEMENT

Conditional branch statements are required whenever it is necessary to change the sequence of execution of instructions based upon a decision. These statements take the following format:

Notation:

```
line:number IF  relational:expression  THEN  line:number     (cr)
```

It includes compound conditions (multiple simple conditions connected by AND's) and complex conditions (multiple simple and/or compound conditions connected by OR's). These conditions were previously defined under "Data Selection Statements", page 3-2, in this section of the Language Manual.

Control is passed to the statement number specified whenever any relational: expression is found to be true.

```
100 INPUT  A   (cr)
110 PRINT  A   (cr)
120 PRINT "DO YOU WISH TO CONTINUE"   (cr)
130 INPUT  B   (cr)
140 IF B = "YES"  THEN 100   (cr)
150 STOP   (cr)
```

In this example, the IF statement is used to test whether or not a user wishes to continue. If he inputs "YES" then control is transferred to statement 100; otherwise the next statement is executed.

Program Halting

THE STOP STATEMENT

Whenever the dataBASIC system encounters an END statement, it indicates to the user that the execution of that user program is complete, and it enables him to either compose and execute a new program or re-execute the program just completed. This same function is performed whenever the dataBASIC system encounters the STOP statement.

Notation:

```
line:number  STOP   (cr)
```

Example:

```
    400  STOP   (cr)
```

PROGRAM DOCUMENTATION STATEMENT

THE REM STATEMENT

The REM statement allows the user to enter as a part of his program, remarks about that program or some section of it. For instance, if a program has several sections, the user may wish to begin each section with one or more lines of REM statements describing what each section does. If a remark exceeds one line, the REM control word must be repeated on each line. GOSUB, GOTO, and IF statements may not refer to this line.

Notation:

line:number REM character:string (cr)

Example:

```
    100 REM THIS IS AN EXAMPLE OF A "REM"  STATEMENT.   (cr)
```

SECTION IV

ADVANCED DATA SELECTION

DATA SELECTION USING WORKING STORAGE FIELDS

In the previously explained LET statement (See Section III), it was mentioned that working storage fields could contain field:names rather than field:values. The dataBASIC system gives this ability to the user so that he can write procedures which are generalized so as to be reusable. The dataBASIC system differentiates between working storage fields to be considered as containing a field:name and those that contain a field:value by looking for an ampersand (&) as the second character of the working storage field:name. Therefore, the field named A would be treated as if it contained the field:name of the record field whose contents are to be addressed. In each case, reference is to the same field but for a different use.

Example:

```
100 FILE USED CAR   (cr)
110 PRINT "ENTER RECORD OR A FIELD NAME"  (cr)
120 INPUT A  (cr)
125 IF A ="END:OF:FILE" THEN 200  (cr)
130 IF A NOT = "RECORD" THEN 160   (cr)
140 STORE RECORD  (cr)
150 GOTO 110  (cr)
160 PRINT "ENTER FIELD VALUE"  (cr)
170 INPUT B  (cr)
180 STORE A&  B  (cr)
190 GOTO 110  (cr)
200 END  (cr)
```

The following example illustrates the use of wsfield& in selecting records:

```
100 FILE USED CAR (cr)
110 PRINT "YOU MAY ENTER UP TO 3 FIELD NAMES AND VALUES" (cr)
120 PRINT "ENTER FIRST FIELD NAME AND VALUE" (cr)
130 INPUT A  B (cr)
140 PRINT "ENTER 2ND NAME AND VALUE OR 0, 0" (cr)
150 INPUT C  D (cr)
160 PRINT "ENTER 3RD NAME AND VALUE OR 0, 0" (cr)
170 INPUT E  F (cr)
180 IF NOT C =0  THEN 210 (cr)
190 LET C = NULL (cr)
200 LET D = NULL (cr)
210 IF NOT E = 0 THEN 240 (cr)
220 LET E = NULL (cr)
230 LET F = NULL (cr)
240 FOR A& = B (cr)
250 AND C& = D (cr)
260 AND E& = F (cr)
270 PRINT RECORD (cr)
280 NEXT (cr)
290 PRINT "TO REPEAT ENTER REPEAT" (cr)
300 INPUT A (cr)
310 IF A = "REPEAT" THEN 110 (cr)
320 END (cr)
```

Note that when working storage fields C and E had a value of zero, the NULL value was moved to those fields. This is done because the data selection process ignores any null-valued fields it finds while in the process of determining the records to be selected.

PIVOTING

Pivoting refers to the use of data from a set of selected records as criteria for choosing another selection path. To exemplify this type logic, assume that part of our file contains information concerning major automobile accidents that have occurred in the state, the identification of these being the auto serial numbers. ("Major" is defined as causing damage greater than $200 on one car.)

This information, which may or may not relate to any of the cars in our previous used car lot example, is supplied by the State Highway Patrol. Assume also that the maintenance of this data is outside our responsibility, and that all data referring to accident records is denoted by having its field name preceded by "ACC:".

Having this data in our file enables us to select cars by the prospective customer's criteria and to provide him with some information of accident history without our needing to directly connect the used car lot records with the accident records. A program using this technique could look like the following:

```
100 FILE USED CAR ⓒⓡ
110 FOR MAKE = "DODGE" ⓒⓡ
120 AND MODEL = "DART" ⓒⓡ
130 AND YEAR = "1967" ⓒⓡ
140 LET A = SERIAL:NO  ⓒⓡ
150 PRINT RECORD ⓒⓡ
160 FOR ACC:SERIAL:NO = A  ⓒⓡ
170 PRINT RECORD ⓒⓡ
180 PRINT ⓒⓡ
190 NEXT  ⓒⓡ
200 PRINT ⓒⓡ
210 NEXT ⓒⓡ
220 END  ⓒⓡ
```

SYNTHETIC SELECTION OF FIELDS

It is sometimes desirable to select records based on a particular grouping, the criteria of which include several factors. For instance, our used car salesman may have a request for an air-conditioned compact car with an AM-FM radio. Without any synthetic selection fields, he might have to use:

```
100 FOR MAKE = "CHEVROLET" ⓒⓡ
110 AND MODEL = "CORVAIR" ⓒⓡ
120 AND OPTIONS = "AIR:COND" ⓒⓡ
130 AND OPTIONS = "AM:FM" ⓒⓡ
140 OR MAKE = "FORD" ⓒⓡ
150 AND MODEL = "MUSTANG" ⓒⓡ
160 AND OPTIONS = "AIR-COND" ⓒⓡ
```

```
170 AND OPTIONS = "AM-FM"  (cr)
180 OR MAKE = "RAMBLER"  (cr)
190 AND MODEL = "AMERICAN"  (cr)
              .
              .
              .
```

He would have to repeat the options definition for every type compact car, thus causing his program to become lengthy and difficult to enter. To avoid this, he may add to his record definitions the field:name SPECIAL. This field:name could contain as many field:values per car as needed to describe synthetic selection criteria. As an example, a Volkswagen could be classified as a SPECIAL "COMPACT" and a SPECIAL "IMPORT". Synthetic selection fields can be assigned to a record either as it enters the file or at a point in time after the file has been created when there is a need for such fields.

SECTION V

CONTROL COMMANDS

CONTROL COMMANDS VERSUS STATEMENTS

dataBASIC control commands direct the system in the disposition of a dataBASIC program; for example, they command the system to either execute, list, or save a program for future use. Commands differ from statements in that they do not form a part of the program and are effective immediately upon being entered at the terminal. Also, control commands are not prefixed with line numbers as statements are, and they may be entered whenever the dataBASIC system is in control.

CONTROL COMMAND CATEGORIES

The control commands available to the dataBASIC system fall into three categories. In the first category are the commands RUN, CREATE, DESTROY, ANALYZE, and VERIFY; these are especially relevant to the dataBASIC system. The function performed by these commands are described under "dataBASIC Commands (Category #1)."

In the second category are commands used by the dataBASIC system but are of the standard Time-Sharing types. These commands are also described but are identified by an asterisk. See "dataBASIC Time-Sharing Commands (Category #2)."

In the third category are standard Time-Sharing commands which are also available to the dataBASIC system but are not particularly relevant to it. These commands are not mentioned in this language manual but are described in the GECOS Time-Sharing System General Information Manual, Document Number CPB-1643.

dataBASIC Commands (Category #1)

RUN

The RUN command instructs the system to execute program statements in numerical sequence. The execution of the program is commonly referred to as running the program, or as a run of a program.

CREATE

The CREATE command is used to allocate and initialize storage space for a dataBASIC file (not a program file). It requests several parameters from the user in order to determine file size requirements. All dataBASIC data files must be created through use of this command within the dataBASIC system. (See Section VI, "dataBASIC Subsystems" for a detailed explanation of this command.)

DESTROY

The DESTROY command is used to release dataBASIC data files only. Program files should be deleted using the PURGE or RELEASE command.

ANALYZE

The ANALYZE command is used to execute the dataBASIC ANALYZE subsystem described in Section VI of this manual.

VERIFY

The VERIFY command is used to execute the dataBASIC VERIFY subsystem described in Section VI of this manual.

dataBASIC/Time-Sharing Commands (Category #2)

*LIST

The LIST command is given when the program is to be printed. This command will result in a printout of the entire program, along with any additions or changes that may have been made prior to the use of LIST. If only a portion of the program is desired, the LIST command can be modified by line numbers indicating the portion desired, as follows:

*LIST xxxx, yyyy

(will result in a printout of the program between line numbers xxxx and yyyy).

*LIST xxxx

(will result in the printout of statements beginning with statement xxxx through the end of the program).

*LIST , yyyy

(will result in a printout of statements from the beginning of the program through statement yyyy).

*DONE

The user terminates his session with the dataBASIC system by the use of the DONE command, but he may still retain use of the terminal for selection of another time-sharing system or re-selection of the dataBASIC system.

*BYE

This command is given when the user wishes to terminate his session with the computer. He will then receive a summary of the amount of resources used for this session along with the total resources used by his account to date. His terminal will then be disconnected from the system.

*SAVE file:name

This command permits the user to save a program for future use. File:name can be any combination of alphanumeric, period, and minus sign characters; but it cannot exceed eight characters. This command is given just prior to discontinuing the immediate use of the program.

*NEW

This command is given when the user is to continue the use of the dataBASIC system by building a new program.

*OLD file:name

This command is given if the user is to select another saved program as his current program. Other forms of the OLD command follow.

*OLD file:name (xxxx, yyyy)

The statements numbered xxxx to yyyy, inclusive, of the program saved under the the name file:name are brought into the user's working storage for processing.

*OLD file:name_1; file:name_2;. . . ; file:name_n

The n named programs are adjoined in the order given, and are brought into working storage. (The line numbers of the resultant program are not resequenced.) The contents of the current file can be included in the new file by use of the name "*" in the file name list. If the list is too long for one line, it may be continued on the next line if a semicolon is the last non-blank character before the carriage return.

*OLD file:name_1 (xxxx_1, yyyy_1);. . . ; file:name_n (xxxx_n, yyyy_n)

The segments of the named files specified by line numbers xxxx through yyyy are adjoined in the order given, and they replace the user's current program. (The line numbers of the resultant program are not resequenced.) If the list is too long for one line, it can be continued on to the next line if a semicolon is the last non-blank character before the carrage return.

For example, the command

```
*OLD PROGRAM1 (10,85);PROGRAM4
```

will cause the statements numbered 10 through 85 of the file PROGRAM1, along with the statements of the file PROGRAM4 to be concatenated in that order, to become the (new) current program.

*RESEQUENCE

This command causes the line numbers of the current program to be resequenced. Resequencing begins with line number 10 and is incremented by steps of 10. Statement-number references within the program (such as GOTO, GOSUB, AND IF statements) are modified correspondingly . Another form of RESEQUENCE is

```
*RESEQUENCE n,m
```

The line numbers of the current program are resequenced beginning with line number n and with increments of m. Either n or m may be omitted; the value 10 will be assumed in either case.

*AUTOMATIC

This command causes the automatic creation of line numbers, beginning at the point at which the automatic mode is entered (or re-entered), with line numbers initially starting at 10 and incremented in steps of 10. These line numbers are generated by the system; appear in the terminal copy, and are written in the file, just as though the user had typed them himself. Another form of AUTOMATIC is

```
*AUTOMATIC n,m
```

Automatic creation of line numbers begin with line number n and are incremented by m.

*TAPE

This command implies that statements are to be entered from the paper-tape reader instead of from the keyboard. See Section VII, "Entering the Program From Paper Tape" for detailed instructions.

*PURGE

This command deletes the specified program files from the system.

SECTION VI
dataBASIC SUBSYSTEMS

The dataBASIC system includes the four subsystems - CREATE, ANALYZE, DESTROY, and VERIFY.

THE CREATE SUBSYSTEM

Functions

The main functions of the CREATE subsystem are to establish the user's dataBASIC file and to initialize it for use by I-D-S and the dataBASIC system. A dataBASIC file is in fact an I-D-S file and as such may be accessed by I-D-S utility routines or user-written I-D-S programs. All I-D-S files which are also dataBASIC files may be processed by dataBASIC programs; however, I-D-S files which are not also dataBASIC files cannot be processed by dataBASIC programs. For further details regarding I-D-S programs, refer to the Honeywell publication entitled Integrated Store (I-D-S), Document Number CPB-1565.

The user supplies estimated values in response to questions asked by the subsystem. These numbers are used to estimate the user's space requirements. A file of the computed size is established and initialized on the disk. The catalog name is user-assigned, but the file name is always set to ".DATA0". A retreat file ".JOUR." is also created to contain before images of all altered I-D-S pages. The size of the retreat file is determined by the equations:

$$R = D \quad , \text{ for } D \leq 5$$

$$R = 5 + \frac{D}{10} , \text{ for } D > 5$$

Where R is the size in links (i.e., 3840 words) of the retreat file, and D is the size in links of the data file. The retreat file ".JOUR." is subordinate to the same subcatalog name assigned by the user to which .DATA0 is subordinated.

In addition to the standard I-D-S page format, the file and control records required for dataBASIC startup are placed on file.

Input

All input used by CREATE is from the keyboard input device via Time-Sharing System derails.

Processing

If the user inputs information that exceeds the hardware or software limits, CREATE will recycle and repeat all the questions. If it must recycle six times, it will terminate the user; if a total of one hundred disk errors occur during the run, the subsystem will be terminated.

Output

The user file is output to the mass storage device having the most available space. One link of twenty I-D-S pages is written at a time. All the I-D-S pages have one record, that is the dataBASIC control record which is stored on them. In addition, Page One has the file record stored on it.

The only other output are the questions asked of the user; these are output to the keyboard device via Time-Sharing System derails.

Error Handling

If the maximum size allowed for an input parameter (in characters) is exceeded, the system starts over after the message "ESTIMATE EXCEEDS SYSTEM LIMITATIONS" has been sent to the user.

If the computed space requirement exceeds the available user capacity, the message "ALLOCATED FILE SPACE EXCEEDED" is sent to the user, and all inputs are requested again up to a maximum of six restarts.

If there is an error return from one of the derails, the message "SYSTEM MALFUNCTION CQ99" (where 99 is the error code returned by TSS) is sent to the user, and all the user inputs are requested again.

If the total I/O errors for the run exceed 100, the message "SYSTEM MALFUNCTION CQ41" is sent to the user, the files are purged, and the abnormal termination return to TSS is executed.

If during the error wrapup, any of the file structure (catalog and/or file) cannot be purged, the message "ANY FILES CREATED MUST BE PURGED USING ACCESS" is printed, and the abnormal termination return to TSS is executed. (ACCESS is explained in Honeywell publication, GECOS Time-Sharing System General Information Manual, Document Number CPB-1643.)

If the name supplied by the user is already being used, the message "NON-UNIQUE FILE NAME" is printed, and the program recycles.

If the password supplied by the user has invalid characters, "INVALID CHARACTERS IN PASSWORD" will be printed, and the user will be asked for the password again.

Error Messages

Error messages and meanings for the CREATE subsystem are tabulated in Figure 6-1.

Error Message	Meaning
System Malfunction CQ41	More than 100 disk errors.
Excessive Errors	More than 10 errors have been made in inputting the answers to the create questions.
System Malfunction CQ43	Currently unused.
Any files created must be purged using access	Attempt to purge catalog and/or file has resulted in errors. Cannot get rid of catalog or files created within run.
System Malfunction CQNN	NN is error status returned from DRL FILACT.
Non-Unique File Name	User-supplied name is already under the user master catalog as a sub-catalog name.
Invalid file name	User-supplied name has an invalid character or a name scan failure.
AFT is full	No room in AFT for file name .DATA0
Duplicate file name in AFT	.DATA0 is already open and in AFT.
Estimate Exceeds System Limitation	User-supplied parameter is too large and would result in an answer that is beyond data storage ability.

Figure 6-1. CREATE Subsystem Error Messages

Error Message	Meaning
Allotted file space exceeded	Space required is greater than a DSU270, or no link space is available, or requested space exceeds maximum allowed.
Password Incorrect	The TSS has returned an error status indicating missing or invalid password as a result of attempt to access the file.
Invalid characters in password	Characters other than A thru Z, 0 thru 9, ".", or "-" found in password.

Figure 6-1. CREATE Subsystem Error Messages (Cont.)

Programming Notes

Below is a copy of the terminal messages with user-responses underlined. Note that all numeric responses are terminated by any non-numeric to appear in the character string. All numeric values must be less than 100,000 (decimal).

```
File name? DUMMY
File Password?
$$$$$$$$$$$$
File Estimates
General, Specific or Explain? EXPLAIN
```

To create a file you must provide estimates which will assist in determining the amount of file space to be allocated. The following describes the estimates you must provide depending on whether you elect General or Specific. All estimates should be as accurate as possible to insure maximum utilization of file space and should allow for any anticipated growth.

General

Records in file - you must provide an estimate of the number of records to be stored in the file.

Fields per record - you must provide an estimate of the average number of fields to be stored in a record.

Specific

Records in file - same as for general.

Fields per record - same as for general.

Unique field names - you must provide an estimate of the number of unique field names to be stored in the file.

Unique field values per field name - you must provide an estimate of the average number of unique field values per field name to be stored in the file.

```
General or specific? GENERAL

Records in file? 100
Fields per record? 10

File initialization complete.
```

```
File name? DUMMY

File password?
$$$$$$$$$$$$
File estimates
General, specific or explain? SPECIFIC

Records in file? 100
Fields per record? 10
Unique Field Names? 15
Unique field values per field name? 12

File initialization complete.
```

THE ANALYZE SUBSYSTEM

Functions

The functions of the ANALYZE subsystem are to analyze a named user's dataBASIC file to determine the percentage of available space used, and to display the information on the user's terminal in the form of a bar chart followed by a summary.

The user selects the file, the type of information to be presented, and the range of pages to be checked for percent of loading.

The execution of the program is dependent on the supplied file name referencing a valid dataBASIC file. The first page is read in and the file record (line number TWO) is examined to determine the page number of the last dictionary and the last page number of the file.

Input

Input comes from the user and from his named dataBASIC file on the disk. User-input is supplied to the program from a keyboard input device in response to questions asked of the user by the program. The disk input read is the first sector of the I-D-S pages indicated by the user.

Processing

The user is required to supply the dataBASIC file name and the program will continue to recycle and ask for the name until a valid one is supplied, but all other questions can be defaulted.

Output

Output to the keyboard I/O device is in one of four formats. The two basic formats are (1) the bar chart with summary and (2) the summary only.

The bar chart has two forms: the standard form with a " " as the left margin which is used for a specific graph (i.e. space used, or lines used), and the "S", "L" used with the maximum graph (i.e., the bar printed shows the maximum use of lines or space).

At the user's option, there are in addition a long form and a short form of the messages.

Error Handling

Error in accessing the catalog: If the file under the named catalog cannot be accessed, the message "SYSTEM MALFUNCTION CR00" is typed out, and the program returns control to the Time-Sharing System.

Invalid name: If the user-supplied name has a special character other than "-" or ".", or if it has more than eight characters, the message "INVALID CATALOG NAME" will be printed, and the program will recycle.

Invalid line number: If in response to the starting and ending page numbers the user inputs a page number larger than four digits, the message "LINE NUMBER GREATER THAN MAXIMUM ALLOWED" will be printed, and the program will again ask the question.

Disk errors: If there are more than twenty-five disk read errors, the message "UNABLE TO READ NAMED FILE" will be printed, and the program will exit back to the primitive list.

Incorrect page: When each page is read, its number in the header is compared to the expected (next) page; if they are not equal, "PAGE NUMBER INCORRECT" is printed and the program will exit back to the primitive list.

Invalid password: If the password contains a special character other than a decimal or a dash, the message "INVALID CHARACTERS IN PASSWORD" is printed, and the user is again asked for the password.

If the TSS returns an error status of 14_8, the message "PASSWORD INCORRECT" is printed and the program recycles.

Error Messages

Error messages and meanings for the ANALYZE subsystem are tabulated in Figure 6-2.

Error Message	Meaning
Invalid catalog name	Something other than a character 0 thru 9, A thru Z. .(Decimal) or -(minus) has been included in the name.
Cannot access named catalog	The attempt to access the catalog has resulted in an error status return from TSS.
Unable to read named file	The attempt to read the file (.DATA0) has resulted in 25 consecutive bad reads.
Page Number Incorrect	When the I-D-S page was read in, it did not match the expected page number.
Password Incorrect	Error status from TSS indicates missing or invalid password.
Invalid characters in password	Special characters other than "." or "-" were found in password.

Figure 6-2. ANALYZE Subsystem Error Messages

Programming Notes

"Y" FOR SHORT FORM MESSAGES

Each of the questions asked of the user has a short form and a long form. A response of Y will result in the printing of the question, explanations, and header in the short form. Any other response will give the "normal" long form.

Except for FILE NAME and FILE PASSWORD, the short form of each question is shown beneath each long form question in the examples which follow:

FILE NAME?

Enter the name of the dataBASIC file to be examined. If the name does not exist, the message "CANNOT ACCESS NAMED CATALOT" will be printed. If the name is too long or no name is supplied, the question will be repeated.

FILE PASSWORD?

****

Enter the password for this file; it is the same as that supplied to CREATE.

"L" LINE LOADING, "M" MAXIMUM, "S" SPACE (Long Form). L,M,S? (Short Form)

There are three types of file analyses:

1. Line loading analysis; i.e., the percentage lines used on each page. (Enter "L" to get this type of analysis.)

2. Space loading analysis; i.e., the percentage of space used on each page. (Enter "S" to get this type of analysis.)

3. Maximum loading analysis; i.e., the percentages are calculated for each page by both space and line, and the larger of the two is used for the graph. If an undefined character or only carriage return, is entered, the "S" option is assumed.

   When the "M" option is selected and the short form has not been selected, the following explanation is output:

   The "]" WILL BE REPLACED BY "S" IF SPACE IS LARGER

   "L" IF LINE PERCENT IS LARGER, UNCHANGED IF EQUAL.

"Y" TO ENTER PAGE NUMBERS (Long Form) (See Figure 6-3.)

RANGE? (Short Form)

If you wish to start and/or stop the analysis at specific page numbers, "X" will allow you to control these limits.

ENTER START IF OTHER THAN PAGE ONE. (Long Form)

START? (Short Form)

Enter the page number of the first page to be graphed; if it is to start with the first page, just hit "return".

ENTER END IF OTHER THAN EOF (Long Form)

END? (Short Form)

Enter the number of the last page to be graphed if the entire file from the "START" is not checked. By entering the same page number in the start and end, a one line (page) graph would result. (See Figure 6-4.)

ENTER "Y" TO LIST SUMMARY ONLY (Long Form) (See Figure 6-5.)

SUMMARY ONLY? (Short Form)

To skip the graph (bar graph) and output, only the summary of the analysis enter "Y". The summary follows all graphs and reflects only the area "analyzed". Thus, if a "range" of 10 to 20 were given and a "Summary Only" requested, the summary would reflect only pages 10 to 20, inclusively.

```
GRAPH OF PERCENT LOADING OF DATA-BASE BY MAXIMUM  (Long Form)
MAXIMUM  (Short Form)
```

Figure 6-3. Sample Long Form and Short Form Headers

```
           1     2     3     4     5     6     7     8     9
           0     0     0     0     0     0     0     0     0
******
0008       ] ..
% FILLED             10%   20%   30%   40%   50%   60%   70%   80%   90%   100%
# OF PAGES          0001  0000  0000  0000  0000  0000  0000  0000  0000  0000
```

Figure 6-4. Sample "One Page" Graph

```
MAXIMUM
******
% FILLED     10%  20%  30%  40%  50%  60%  70%  80%  90%  100%
# OF PAGES  0021 0027 0009 0002 0003 0006 0003 0001 0005 0003
```

Figure 6-5. Sample "Summary Only"

```
    0059 S....1.
    0060 S....1.
    0061 S....1....
    0062 L..
    0063 L..
    0064 S....1....
    0065 S....
    0066 S....
    0067 S....1.
    ******
    0068 S....1....2....3....4....5....6...
    0069 S....1....2....3....4....5....6....7....8
    0070 S....1....2....3....4....5....6....7....8
    0071 S....1....2....3....4....5....6....7....8
    0072 S....1....2....3....4....5....6....
    0073 L..
    0074 L..
```

(Note the "S" and "L" marking of left column to show which percent was larger, space or lines. Also note that the six asterisks (******) mark the division of dictionary area from record area.)

Figure 6-6. Sample "Maximum"

THE DESTROY SUBSYSTEM

Function

The DESTROY subsystem is used to release a dataBASIC file. If the tree structure of catalog and files is not properly handled, there will be an unanswered and almost unrecoverable loss of link space. To avoid this, DESTROY was developed to handle the release of the entire structure.

The user supplies the name of his "file" which, in the file structure, is actually the catalog name; a check is made to determine if there is a retreat file and data file under the catalog. The files that are present are released, starting with the retreat file, followed followed by the data file, and finally the catalog.

If any file is present and cannot be released, then DESTROY stops to notify the user and to return control to the system.

If all the files and catalog are released, the word "SUCCESSFUL" is printed on the user's keyboard device.

Input

The only input to the program is the dataBASIC file name which is supplied by the user from the keyboard input device via TSS.

Processing

The user-supplied name is put into the cat/file description as the catalog name, and the name ".JOUR." is put in as the file name. An attempt is made to access the file. If the file is not present, the program next considers the data file ".DATA0". If it is present, it must be successfully released or the program will not continue.

The catalog is then purged last.

Output

The only output consists of the messages to the user at the keyboard device via TSS derails.

Error Handling

If the journal is present and cannot be purged, the message "SYSTEM MALFUNCTION CS30" is printed and the program exists to the system. If the catalog cannot be purged, the message "SYSTEM MALFUNCTION CS10" is printed and the program exits to the system.

If the user-supplied name has special characters other than "." or "-", the message "INVALID FILE NAME" is printed and the program recycles.

If the password contains special characters other than decimal or dash, the message "INVALID CHARACTERS IN PASSWORD" is printed and the user is asked for the password.

If TSS returns a status of 14_8 to the program, a message of "PASSWORD INVALID" is printed and the program recycles.

Error Messages

Error messages and meanings for the DESTROY subsystem are tabulated in Figure 6-7.

Error Message	Meaning
Invalid File Name	Name has character included that is non-alpha, non-numeric, and not a decimal (.) or dash (-).
System Malfunction CS30	Journal is present but cannot be purged from the file.
System Malfunction CS20	Data file is present and cannot be purged. Journal has been purged.
System Malfunction CS10	Cannot purge catalog. Journal and data files have been purged. .
Invalid Characters in Password	Special characters other than "." or "-" found in password.
Password Incorrect	Error status from TSS indicates missing or invalid password.

Figure 6-7. DESTROY Subsystem Error Messages and Meanings

THE VERIFY SUBSYSTEM

Functions

The VERIFY subsystem is a Time-Sharing subsystem which checks a dataBASIC file by traversing the I-D-S chains in the file. A count of all records of each record type is made and compared with the corresponding counter in the file record. All record counts in the PRIME NAME and PRIME VALUE records are also verified. Whenever the count differs from that in the file, it replaces the previous count. ENTITY, NAME, and VALUE records having no OCCURRENCE detail records are deleted. Whenever no SYNONYM record exists which corresponds to a NAME record (or no INDEX record corresponding to a Value record), a SYNONYM record is generated.

Input

Input is any dataBASIC file.

Processing

In response to SYSTEM? specify DATABASIC.

For OLD OR NEW, respond NEW.

For *, respond VERIFY.

For FILENAME, give the DATABASIC file name.

For PASSWORD, give the password for the dataBASIC file.

Output

Output consists of a list of (1) record types found in the file, (2) corresponding record counts as recorded in the file record (noted as "OLD"), and (3) the record counts generated as this program traverses the various I-D-S chains (noted as "NEW"). Whenever these counts differ, an "*" is printed following the new count, and the new count replaces the corresponding count in the file record.

Error Handling

Whenever a Prime Name record, a Prime Value record, or an Entity record has no occurrences, that record is deleted and the corresponding record count is adjusted. Any deleted names or values are printed.

Whenever a Prime Name record has no Synonym record, a Synonym record is generated.

Whenever a Prime Value record has no Index record, an Index record is generated.

A three-way check of the total number of occurence records is made which includes (1) total entity occurrences, (2) total name occurrences, and (3) a sum of the value occurrence counts in the value records. Whenever two of these disagree, the three totals are printed.

Error Messages

The following error messages may occur when attempting to access the dataBASIC file (in each case the file name or password should be given when requested):

"*** ERR - ILLEGAL DELIMITER X"

"*** ERR - NAME TOO LONG"

"*** ERR - PASSWORD TOO LONG"

If an I-D-S error of the type which can be trapped by the program occurs, a message of the following type is printed:

"*** ERR - UNSUCCESSFUL AAAAA BBBBB - CURRENT RECORD XXXXX YY"
where AAAAA is the dataBASIC record (or chain) name, BBBBB is either RETRIEVE, STORE, MODIFY or DELETE, and XXXXX YY is the reference code of the current record.

```
    VERIFICATION OF FILE DBTEST                  12/05/69
    VALUE COUNT ERROR, PRIME NAME - MAIL:DROP
         OLD COUNT      8      NEW COUNT       7
    PRIME VALUE DELETED - - X
    VALUE COUNT ERROR, PRIME NAME - ASSIGNED
         OLD COUNT      11     NEW COUNT       10
    2 PRIME NAME RECORDS WITH VALUE COUNT ERRORS
         TOTAL COUNT          OLD        NEW        DELETIONS     ADDITIONS
    ENTITY                     23         23
    PRIME NAME                 15         15
    SYNONYM                    15         15
    VALUE                     122        122            1
    INDEX                     122        122            1
    OCCURRENCE                179        179
*** ERRORS NOTED ABOVE WERE CORRECTED ***
END OF DATABASIC FILE VERIFICATION
```

Figure 6-8. Output Report Sample of Verification of File DBTEST

SECTION VII

COMMUNICATING WITH THE SYSTEM

TERMINAL OPERATION CONTROLS

This manual assumes that the terminal used is a Teletype Model 33 or 35. With this terminal, the communication between user and computer is displayed by means of typed copy on paper. The Teletype keyboard is a standard typewriter keyboard except that it has special-purpose keys which the user must be familiar with. These keys with associated functions are indicated in Figure 7-1.

KEY	FUNCTION
RETURN	Depressing the RETURN key returns the carriage and transmits the typed line to the system. The computer ignores the typed line until this key is depressed.
CTRL plus X	When these keys are depressed simultaneously, the terminal deletes the entire line being typed. The word DEL is printed and the carriage is returned. The line is ignored by the computer.
SHIFT plus $\overset{@}{\text{P}}$	The @symbol is located on the P key and is generated when depressed with either shift key. It is used to delete the character or space immediately preceding the @. If this key is depressed n times, the n preceding characters or spaces will be deleted. For example: ABCWT@@DE will be treated as ABCDE when RETURN is depressed. AB C@@@CDE will be treated as ACDE when RETURN is depressed.
BREAK	Depressing the BREAK key causes the system to discontinue printing or computation. One type of terminal requires that a BRK-RLS (break-release) button be depressed following the use of BREAK in order that operations continue.

Figure 7-1. Teletype Model 33 or 35 Special Purpose Keys

KEY	FUNCTION
BREAK	BREAK should not be used unless absolutely necessary because it causes the file to be restored to its contents prior to execution if the dataBASIC program caused the file to be changed.

Figure 7-1. Teletype Model 33 or 35 Special Purpose Keys (continued)

Other operational controls not on the keyboard are necessary to the operation of the terminal; they include power on-off, connection to a phone line, and selection of operating mode. The location and operation of these controls differ according to the type of terminal in use. The user must receive on-site instruction or must study the instruction manual for his terminal to gain familiarity with these operational controls. For a complete description of the Teletype unit, refer to the instruction manual accompanying the unit.

CONNECTING TERMINAL TO COMPUTER

In order to connect with the computer from a terminal, proceed as follows:

1. Turn unit on and obtain a dial tone.
2. Dial one of the numbers at the Time-Sharing Center.

When the connection is made, a high-pitched tone is received; then there will be no tone at all, and the terminal will print out an indication that the computer is available and that communication with the computer can be made through the terminal.

GETTING ON PROCEDURE

With the terminal connected to the computer, the system initiates a "log-on" procedure. During this procedure, the terminal will ask for information; to this a proper response must be made, each response followed by a carriage return (achieved by depressing the RETURN key). First, the terminal will ask for a user's identification. (This is a string of characters assigned to uniquely identify the user to the computer for the purposes of identifying his programs and accounting for the user's charges.)

Next, the terminal will ask for a password. The area on which the password is printed will be scored over by the terminal to make the password illegible. The purpose of this password is to assure the computer that it is "talking" to the legitimate user and not to someone else using his identification. The password is his protection against unauthorized use of his user identification.

Next, the terminal will ask the user to select the system he wishes to use (in this case, the dataBASIC system). If an invalid system name is given, the system will print the message "SYSTEM UNKNOWN" and will repeat the request for a name until a valid name is given. After a valid response, the terminal will ask if the user is going to work with an OLD or NEW program, to which the user must reply with either OLD or NEW.

A NEW program is one in which the user will enter all of the program statements at this session at the terminal. An OLD program, on the other hand, is a program that has been previously generated at other sessions at the terminal and has been saved for future use. If the user's response is OLD to the question OLD or NEW, the system will ask him for the OLD file name. This will be the same name he had previously used when saving his program with the control command SAVE.

After the terminal prints READY FOR INPUT, and an asterisk on the following line, the user may begin to enter his new program, add or modify statements in his old program, or use one of the control commands (e.g., LIST or RUN).

A typical log-on sequence follows:

```
THIS IS THE T/S SYSTEM ON 09/14/67 AT 9.183
USER ID -- DOE
PASSWORD
$$$$$$$$
SYSTEM? DATABASIC
OLD OR NEW-NEW
READY FOR INPUT
*
```

This example illustrates the most elementary use of the OLD/NEW selection of programs.

CREATING A dataBASIC PROGRAM

The essentials of forming statements or lines and of creating a dataBASIC program are as follows. Each line contains four parts in the below listed sequence.

1. Line number: Each statement is prefixed by a one to five digit line number that serves two purposes: (a) It is used as a sequence control within the program, specifying the order in which statements are to be executed, and (b) it uniquely identifies a line. It can contain no imbedded spaces, but must be followed by one or more spaces.

2. Control word: This word tells the dataBASIC system what function it is to perform. (See Appendix A for list of control words.)

3. All other words: These are written in accordance with the specifications for each control word.

4. Carriage return character: This character denotes the end of a line. The dataBASIC system responds to this with a line feed and prints an asterisk (*) when it is ready to accept the next line of input.

   Characters after the 72nd character of a line will not be used.

   An example of a statement follows:

```
10 READ  A B C D    (cr)
```

   (The line is identified as statement 10, READ is the control word, and A B C D are variables.)

   A second example is:

```
40 END   (cr)
```

   The line is identified as statement 40, and END is the control word constituting the statement. The actual entry at the terminal of a sequence of statements of a dataBASIC program requires knowledge of control commands, terminal operation, and elementary dataBASIC statements.

ENTERING A PROGRAM

After the terminal prints READY FOR INPUT, it indicates its availability for input by printing an asterisk on the next line at the left margin. Thereafter, each carriage return generates an asterisk at the left margin of each succeeding line, thus indicating readiness for input. Each statement should begin with a line number (after the asterisk) containing no more than eight digits and no spaces or non-digit characters. The RETURN key must be depressed at the completion of each line of input to achieve a carriage return, causing the transmission of the information to the computer.

The program input for a simple program following READY FOR INPUT and subsequent asterisk would appear as follows:

```
READY FOR INPUT
*10 FOR MAKE = "BUICK"   (cr)
*20 PRINT RECORD   (cr)
*30 NEXT   (cr)
*40 END   (cr)
*RUN   (cr)
```

The above program would print the records describing BUICK upon the receipt of the control command RUN.

ENTERING THE PROGRAM FROM PAPER TAPE

If the user is to enter his program from paper tape, he must respond with the control command TAPE after READY FOR INPUT. The procedure for using paper tape is as follows:

1. Place paper tape in terminal tape reader.
2. Select tape-input operating mode, if required.
3. Start tape reader.
4. Input from paper tape will be accepted until one of the following occurs:
    a. tape reader is turned off,
    b. tape runs out,
    c. tape jams in tape reader, or
    d. an X OFF character is encountered on the tape.

ERROR CORRECTIONS

If while entering his program the user has made errors which are self-evident, he can correct his program during typing or before giving the RUN command as follows:

- A new statement may be substituted for a statement containing errors by retyping the statement number and a corrected version of the statement. The first version of the statement will be ignored in the running or listing of the program.
- A statement may be eliminated from the program by typing its number and depressing the RETURN key. That statement will then be ignored during the running or listing of the program.
- The current line being typed can be deleted by depressing the CTRL and X keys simultaneously. That line will then be ignored.
- Typing errors if perceived during the typing process may be corrected by using the @ symbol. The character or space immediately preceding the @ will thus be deleted. If this key is depressed n times, the n preceding characters will be deleted.
- Additional statements may be inserted into the program by typing them with line numbers which indicate their places within the program sequence. For example, if one or more new statements are desired between statements 30 and 40, they could be assigned line numbers from 31 to 39. In the running or listing of the program, the new statements will be properly sequenced.

If language errors (statements violating the dataBASIC language format) are made by the user while entering his program and they are not perceived, error messages of a diagnostic nature will be printed upon use of the control command RUN to aid the user in making corrections. (See Appendix D.)

RUNNING THE PROGRAM

After typing the complete program, the user types the control command RUN and depresses the RETURN key. If there are no format errors, the computer will execute the statements, and the terminal will print out the results. If it is obvious to the user that wrong answers are being given, he can depress the BREAK key, causing output to cease. However, it is better that he debug his program by limiting the record selection phase through the use of counts to some small numbers of records. For example, if the user were interested in finding the total value of all Plymouths with automatic transmissions in his lot, and he wanted to check his program prior to running it completely, he might use this sequence of statements:

```
10 FOR MAKE = "PLYMOUTH"
20 AND TRANSMISSION = "AUTOMATIC"
30 AND B < 10
40 LET A = SUM SALES:PRICE
50 LET B = SUM 1
60 PRINT SALES:PRICE
70 NEXT
80 PRINT "TOTALS" A " "NO. CARSb̸"B
90 END
```

Using this program, he could check his results by computing the sum of the sales prices for the first 10 selected cars, and then checking this computation against the total printed on line 80. When he is satisfied that his results are correct, he could remove lines 30 and 60 from the program, and run it. If logical errors were made by the user in constructing his program, the results will be erroneous or may not appear at all.

Logical errors do not generate error messages, they must be found by analyzing the program. Upon completion of program execution and its resulting output (if any), the terminal prints READY which indicates the system's availability for further input. If the user wishes to modify his program, he may now do so by retyping only those statements he wishes changed to achieve the desired modification. When the control command RUN is again given, a new output will be produced. The modification process can be repeated as often as wanted by the user. The control command LIST may be used at any time the user wishes to inspect the current content of his program; it will show the result of any modification.

If the user wishes to save his program for future use, he must use the control command SAVE file:name; the system will respond with

DATA SAVED-file:name

where file:name is the name under which the program is saved. If the user wishes to discontinue working with his present problem but to continue the use of the dataBASIC system, he may use either the command NEW or OLD. If NEW is typed, the system will respond with READY and the user can then enter a new program. If OLD is typed, the system will ask for OLD NAME. When the old program name file:name is supplied, the system will respond with READY FOR INPUT. Modifications can be made as with a NEW program, and the program can be listed. Upon the control command RUN, the old program will be run. (The entry OLD file:name will bypass the request OLD NAME-.)

NOTE: The old program must be a dataBASIC program which has been saved at a previous session at the terminal. If the user types the control command DONE while the dataBASIC system is requesting input from the terminal, the time-sharing system will sign him off the dataBASIC system; but it will permit him to select another system within the confines of the time-sharing system, and continue his use of the computer.

GETTING OFF PROCEDURE

If the control command BYE is entered while the dataBASIC system is requesting input from the terminal, it will cause the time-sharing system to "log-off" the user and disconnect the terminal. The time-sharing system will then provide a summary of the amount of time and resources used for this run, along with the total amount of the user's resources expended to date.

AUTOMATIC TERMINATION FROM TERMINAL

The user will be automatically terminated from the system for any of the following reasons:

1. If he responds twice with an invalid user identification, the terminal will reply after the first invalid use with the message ILLEGAL ID-RETYPE--; if he responds with an invalid user identification a second time, he will be terminated.

2. If he responds twice with an invalid password, the terminal will reply after the first invalid use with the message ILLEGAL PASSWORD--RETYPE--; if he responds with an invalid password a second time, he will be terminated.

3. If he depresses the CLR button on the terminal.

4. If he leaves the terminal in an idle state for more than ten minutes.

5. If his user's resources are overdrawn by more than 10 percent, the message "RESOURCES EXHAUSTED. CANNOT ACCEPT YOU" will be printed by the terminal before termination takes place.

APPENDIX A
SUMMARY OF dataBASIC LANGUAGE STATEMENTS, EXPRESSIONS AND REPLACEMENTS

Table A-1. dataBASIC Language Statements

Functional Category	Control Word	Statement
DECLARATIONS	DATA	DATA literal ... literal
	FILE	FILE filename
DATA SELECTION	FOR	FOR ALL FOR name FOR relational:expression FOR FNAME dictionary:expression FOR FVALUE dictionary:expression
	NEXT	NEXT
DATA MANIPULATION	DELETE	DELETE RECORD DELETE name DELETE name value value ... value DELETE ALL value
	FIX	FIX name = value FIX name value = value
	LET	LET wsfield = assignment:expression LET wsfield = arithmetic:expression
	STORE	STORE RECORD STORE COPY STORE name value value ... value
INPUT/OUTPUT	INPUT	INPUT wsfield wsfield ... wsfield
	PRINT	PRINT RECORD PRINT print:expression print:expression ... print:expression PRINT print:expression print:expression ... print:expression; PRINT
	READ	READ wsfield wsfield ... wsfield

Table A-1. dataBASIC Language Statements (Cont.)

Functional Category	Control Word	Statement
CONTROL	END	END
	EXIT	EXIT
	GOSUB	GOSUB line:number
		GOSUB line:number wsfield wsfield ... wsfield
	GOTO	GOTO line:number
	IF	IF relational:expression THEN line:number
	RETREAT	RETREAT
	RETURN	RETURN
	STOP	STOP
DOCUMENTATION	REM	REM character:string (Note: Value references are restricted to wsfield and literal for "STORE", "FIX", and "DELETE" statements.)

Table A-2. dataBASIC Expression and Replacements

Expression	May Be Replaced By
relational:expression	compound:cond. compound:cond. OR compound:cond. . . . OR compound:cond.
compound:condition	condition condition AND condition . . . AND condition
condition	simple:condition NOT simple:condition
simple:condition	name ALL ALL value value relational:operator value value NULL value FROM value TO value
name	rfield wsfield &
value	literal wsfield rfield wsfield &
relational:operator	= (which means equal to) > (which means greater than) < (which means less than) = > (which means greater than or equal to) < = (which means less than or equal to) < > (which means not equal to)
dictionary:expression	ALL relational:operator value FROM value TO value
arithmetic:expression	value arithmetic:operator value ABS value RND value SUM value NEG value INT value

Table A-2. dataBASIC Expression and Replacements (Cont.)

Expression	May Be Replaced By
assignment:expression	value MAX value MIN value NULL FNAME FVALUE
arithmetic:operator	+ (which means add) - (which means subtract) * (which means multiply) / (which means divide) ∧ (which means raise to a power)
format:expression	" alphanumeric:format " " numeric:format "
print: expression	value value EDIT format:expression FNAME FVALUE

APPENDIX B

ALPHABETIC LIST OF TERMINAL COMMANDS

AUTOMATIC

ANALYZE

BYE

CATALOG

CREATE

DESTROY

DONE

LIST

NEW

OLD

PURGE

RESEQUENCE

RUN

SAVE

STATUS

TAPE

VERIFY

APPENDIX C

ALPHABETIC LIST OF RESERVED WORDS

ABS
ALL
COPY
EDIT
FNAME
FROM
FVALUE
HIER
INT
LOCK
MAX
MIN
NEG
NOT
NULL
PAIR
PHO
PRINTER
PUNCH
READER
RECORD
RND
RULE
SUM
TAPE
THEN
TO
UNTIL

APPENDIX D

TERMINAL ERROR MESSAGES AND SYSTEM MALFUNCTION MESSAGES

TERMINAL ERROR MESSAGE TYPES

Terminal error messages are printed by the dataBASIC system whenever a dataBASIC language rule is violated. These messages are printed at the terminal after the control command RUN is given.

Terminal error messages may be divided into three classes; namely, compiler error messages, routine error messages, and language processor error messages.

Compiler Error Messages

Compiler error messages may be printed during program compilation and may prevent execution. They may be subdivided into two groups. The first group references the statement printed on the line preceding the error message. An example of this group of error messages is

```
100A  LET A = "XXX"
            MISSING OR INVALID LINE NUMBER
```

The above example indicates that the statement contains a missing or invalid line number and is repeated in Table D-1 along with other compiler error messages.

Table D-1. Compiler Error Messages - Type 1

Message	Explanation
MISSING OR INVALID LINE NUMBER	Statement contains a missing or invalid line number.
INVALID STATEMENT	Statement is unrecognizable.
INVALID OPERATOR	Statement contains an invalid arithmetic or logical operator.
INVALID STATEMENT FORMAT	Statement has been specified incorrectly.
INVALID EDIT FORMAT	Statement contains an invalid edit format control word.
MISSING OR INVALID FIELD NAME	Statement is either missing a required field:name or contains an invalid field:name.
MISSING OR INVALID FIELD VALUE	Statement is either missing a required field:value or contains an invalid field:value.
FILE PREVIOUSLY DECLARED	A file statement has previously been encountered. dataBASIC allows reference to only one file during a run.
MISSING OR INVALID WORKING STORAGE REFERENCE	Statement is either missing a required working storage reference or contains an invalid working storage reference.
INVALID CHARACTER IN STATEMENT	Statement contains an invalid dataBASIC character for which a blank is substituted.
MISSING QUOTE MARK	Statement contains an alphanumeric literal or print format control word with a missing quote mark.
UNPAIRED NEXT STATEMENT	Statement has no corresponding FOR statement.
INVALID EXPRESSION	Statement contains an invalid arithmetic or logical expression.

Table D-1. Compiler Error Messages - Type 1 (cont.)

Message	Explanation
EXPRESSION EXCEEDS COMPILER LIMITATION	Statement has caused a compiler limitation to be exceeded. Compiler limitations are ● A maximum of 10 nested FOR statements. (This may cause invalid UNPAIRED NEXT STATEMENT error messages to be printed.) ● A maximum of 25 simple conditions in a relational:expression.

The second group of compiler error messages references the overall structure of the dataBASIC program. This group is tabulated in Table D-2.

Table D-2. Compiler Error Messages - Type 2

Message	Explanation
FILE NOT DECLARED	Program does not contain a file statement.
MISSING NEXT STATEMENT	Program contains one or more FOR statements with no corresponding NEXT statement.
LINE XXXXX UNDEFINED	Program references a line numbered statement which is undefined.

Runtime Error Messages

Runtime error messages may be printed during execution of a dataBASIC program. Each of these messages contains a reference to the statement being executed when the error occurred. Action taken after the error occurs is error-dependent.

An example of a runtime error message which indicates that a divide check occurred during the execution of statement 100 is

```
LINE 100 DIVIDE CHECK
```

Other runtime error messages are listed and explained in Table D-3 below.

Table D-3. Runtime Error Messages

Message	Explanation
LINE XXXXX FILE UNAVAILABLE	File is currently being updated by another user. Program is terminated with no retreat.
LINE XXXXX ALLOTTED FILE SPACE EXCEEDED	File space has been filled to capacity. Program is terminated and retreat performed.
LINE XXXXX INVALID INPUT	Terminal input has been incorrectly specified. Request for input is repeated.
LINE XXXXX NO CURRENT RECORD	No current record is available for a statement requiring a current record for execution. Program is terminated, and if the program contains a statement which modifies the file, retreat is performed.
LINE XXXXX INVALID FIELD VALUE	Field:name specified in wsfield is invalid. Program is terminated, and if the program contained a statement which modified the file, retreat is performed.
LINE XXXXX FILE NOT DECLARED	Statement requiring access to the file has been executed prior to execution of the file declaration statement. Program is terminated with no retreat.
LINE XXXXX OUT OF DATA	Attempt to read more data than specified in DATA statements. Program is terminated with no retreat.
LINE XXXXX DIVIDE CHECK	Divide check occurred during execution of statement specified. The result is set zero and the program continues.
LINE XXXXX OVERFLOW	Arithmetic overflow occurred during execution of statement specified. The result is set zero and the program continues.

Table D-3. Runtime Error Messages (cont.)

Message	Explanation
LINE XXXXX INVALID EXPONENT	Exponentiation error occurred during execution of statement specified. The result is set zero and the program continues.

Language Processor Error Messages

Language error messages may be printed during compilation or execution of a dataBASIC program. Each of these messages causes processing to stop. If an error occurs during execution and the program contains a statement modifying the file, then retreat will occur.

Table D-4 lists and explains the two language processor error messages printed by the dataBASIC system:

Table D-4. Language Processor Error Messages

Message	Explanation
LINE XXXXX MEMORY EXCEEDED	The memory allotted to compile and execute the dataBASIC program has been exceeded.
SYSTEM MALFUNCTION XXXXX (See also tables D-5 through D-12)	A system malfunction has occurred over which the user has no control. This error should be reported to persons responsible for maintaining the system. System malfunction messages may be classified according to the activity during which they originate.

SYSTEM MALFUNCTION MESSAGES

System malfunction messages may occur during the many activities of the dataBASIC system. Tables D-5 through D-12 provide a list of these messages in code form, along with a definition of each message and its associated activity source.

Table D-5 lists system malfunction messages which may occur during the compilation or execution of a program:

Table D-5. System Malfunction Messages During Compilation/Execution

Message Code	Definition	Activity Source
00004	COMPILER ERROR	During management routines
AA01	INVALID ERROR CODE-COMPILER ERROR	
AA02	BAD OFFSET	
AAF1	INVALID OP CODE	
AAF2	MEMORY FAULT	
AAF3	TAB FAULT	
BJ01	COMPILER ERROR IN SOURCE FILE (POSSIBLE NO SOURCE FILE)	During control routines
BJ03	COMPILER ERROR - BLOCK COUNT	
BJ02	COMPILER ERROR - READING INPUT	
BM01	COMPILER ERROR - INVALID SUBROUTINE NAME	During code generation routines
BM02	COMPILER ERROR - INVALID OPERATOR	
BM03	COMPILER ERROR - INVALID LOGIC OPERATOR	
BM04	COMPILER ERROR - CODE GENERATION	
BM05	COMPILER ERROR - CODE GENERATION	
BM06	COMPILER ERROR - CODE GENERATION	

Table D-5. System Malfunction Messages During Compilation/Execution (cont.)

Message Code	Definition	Activity Source
BM07	COMPILER ERROR - CODE GENERATION	During code generation routines
BM08	COMPILER ERROR - CODE GENERATION	
BM09	COMPILER ERROR - CODE GENERATION	
BM10	COMPILER ERROR - CODE GENERATION	
BK01	EXPANSION LEVEL GREATER THAN 8	During compiler expansion routines
BK02	EXPANSION ERROR	
BK03	ERROR IN FOR... EXPANSION	
BK04	ERROR IN FOR... EXPANSION	
BK05	ERROR IN FOR... EXPANSION	
BK06	ERROR IN FOR... EXPANSION	
BK07	ERROR IN FOR... EXPANSION	
BK08	ERROR IN FOR... EXPANSION	
BK09	ERROR IN FOR... EXPANSION	
BK10	ERROR IN FOR... EXPANSION	
BK11	ERROR IN FOR... EXPANSION	
BK12	EXPANSION ERROR	
BK14	ERROR IN FOR... OR FIX...EXPANSION	

Table D-5. System Malfunction Messages During Compilation/Execution (cont.)

Message Code	Definition	Activity Source
BK15	ERROR IN FIX... EXPANSION	During compiler expansion routines
BK16	ERROR IN FIX... EXPANSION	
BK17	ERROR IN LET... EXPANSION	
BK18	ERROR IN LET... EXPANSION	
BK19	EXPANSION ERROR	
BK20	EXPANSION ERROR	
BK21	EXPANSION ERROR	
BK22	EXPANSION ERROR	
BK23	EXPANSION ERROR	
BK24	EXPANSION ERROR	
BK25	EXPANSION ERROR	
BK26	EXPANSION ERROR	
BK27	EXPANSION ERROR	
BK28	EXPANSION ERROR	
BK29	EXPANSION ERROR	
BK30	EXPANSION ERROR	
BK31	EXPANSION ERROR	
BK32	EXPANSION ERROR	
BK33	EXPANSION ERROR	
BK34	EXPANSION ERROR	
BK35	EXPANSION ERROR	
BK36	EXPANSION ERROR	
BK37	EXPANSION ERROR	

Table D-5. System Malfunction Messages During Compilation/Execution (cont.)

Message Code	Definition	Activity Source
BK38	EXPANSION ERROR	During compiler expansion routines
BK39	EXPANSION ERROR	
BK40	EXPANSION ERROR	
BK41	EXPANSION ERROR	
BK50	ERROR IN FIX... EXPANSION	
BK51	ERROR IN FIX... EXPANSION	
BK52	ERROR IN FIX... EXPANSION	
0005	COMPILER SCAN ERROR	During compiler management
0006	COMPILER ERROR - PREVIOUSLY DEFINED LINE NO.	
0007	COMPILER ERROR - INVALID ERROR CODE	
0008	COMPILER ERROR - IN LINE NO. REFERENCE	

System malfunction messages are also possible at runtime, being generated during I-D-S routines. These messages are listed and defined in Table D-6.

Table D-6. System Malfunction Messages During I-D-S Routines

Message Code*	Definition
XANN	ANN is the I-D-S ERROR CODE (See CPB-1565.)
DMNN	M IS 1 OR 2, AND NN IS THE I-D-S ERROR CODE (See CPB-1565.)
*Note: The code NN can take on four values not defined as normal I-D-S error codes. These values are defined below and are generated by the QTDRL subroutine:	
76	CHECKSUM CHARACTER ALERT
77	END OF FILE CONDITION
78	END OF LOGICAL FILE CONDITION
79	UNDEFINED I/O ERROR CONDITION

System Malfunction Messages may also be generated by .SIDSO during a routine attempt to access the dataBASIC file .DATA0. These messages are tabulated and defined in Table D-7.

Table D-7. System Malfunction Messages During .DATA0 Accessing

Message Code	Definition
BB1B	4002 - I/O ERROR - CANNOT PROCEED
BB1F	4006 - LLINK SPACE EXHAUSTED
BB1G	4007 - DEVICE TYPE UNDEFINED
BB1H	4010 - LINK SPACE EXHAUSTED
BB1I	4011 - NON-UNIQUE NAME
BB1J	4012 - SIZE REQUESTED LESS THAN CURRENT SIZE
BB1K	4013 - REQUESTED SPACE EXCEEDS THAT ALLOWED

Table D-7. System Malfunction Messages During .DATAO Accessing (cont.)

Message Code	Definition
BB1M	4015 - I-D-S FILE IN ABORT STATUS
BB1N	4016 - I-D-S FILE IN RECOVERY STATUS
BB10	4017 - SEEK ADDRESS CALCULATION ERROR
BB1P	4020 - FAILURE IN NAME SCAN (IMP.)
BB1Q	4021 - UNDEFINED DEVICE (IMP.)
BB1R	4022 - DEVICE LINK TABLE CHECKSUM ERROR
BB1S	4023 - INCONSISTENT FDW BLOCK COUNT
BB1T	4024 - INTERNAL LINK TABLE CHECKSUM ERROR
BB1V	4037 - DUPLICATE NAME IN AFT
BB1W	4040 - NO PAT SPACE AVAILABLE UNDEFINED STATUS CODE RETURNED

System Malfunction Messages may also be generated by .SIDSO during a routine attempt to access the Retreat file .JOUR. These messages are listed and defined in Table D-8.

Table D-8. System Malfunction Messages During .JOUR. Accessing

Message Code	Definition
BB2A	4001 - NAME NOT IN SYSTEM MASTER CATALOG
BB2B	4002 - I/O ERROR - CANNOT PROCEED
BB2C	4003 - PERMISSIONS DENIED
BB2D	4004 - FILE BUSY
BB2E	4005 - INCORRECT CATALOG/FILE DESCRIPTION
BB2F	4006 - LLINK SPACE EXHAUSTED

Table D-8. System Malfunction Messages During .JOUR. Accessing (cont.)

Message Code	Definition
BB2G	4007 - DEVICE TYPE UNDEFINED
BB2H	4010 - LINK SPACE EXHAUSTED
BB2I	4011 - NON-UNIQUE NAME
BB2J	4012 - SIZE REQUESTED LESS THAN CURRENT SIZE
BB2K	4013 - REQUESTED SPACE EXCEEDS THAT ALLOWED
BB2L	4014 - REQUIRED OR INCORRECT PASSWORD
BB2M	4015 - I-D-S FILE IN ABORT STATUS
BB2N	4016 - I-D-S FILE IN RECOVERY STATUS
BB2O	4017 - SEEK ADDRESS CALCULATION ERROR
BB2P	4020 - FAILURE IN NAME SCAN (IMP.)
BB2Q	4021 - UNDEFINED DEVICE (IMP.)
BB2R	4022 - DEVICE LINK TABLE CHECKSUM ERROR
BB2S	4023 - INCONSISTENT FDW BLOCK COUNT
BB2T	4024 - INTERNAL LINK TABLE CHECKSUM ERROR
BB2V	4037 - DUPLICATE NAME IN AFT
BB2W	4040 - NO PAT SPACE AVAILABLE
BB2O	UNDEFINED STATUS CODE RETURNED

Other system malfunction message may occur during an attempt to RETREAT to restore the integrity of a data file. These messages are listed and defined in Table D-9.

Table D-9. System Malfunction Messages During RETREAT Attempt

Message Code	Definition
CP00	UNDEFINED FILSYS STATUS CODE RETURNED
CP01	4001 - NAME NOT IN SYSTEM MASTER CATALOG
CP02	4002 - I/O ERROR - CANNOT PROCEED
CP03	4003 - PERMISSIONS DENIED
CP04	4004 - FILE BUSY
CP05	4005 - INCORRECT CATALOG/FILE DESCRIPTION
CP06	4006 - LLINK SPACE EXHAUSTED
CP07	4007 - DEVICE TYPE UNDEFINED
CP10	4010 - LINK SPACE EXHAUSTED
CP11	4011 - UNIQUE NAME
CP12	4012 - SIZE REQUESTED LESS THAN CURRENT SIZE
CP13	4013 - REQUESTED SPACE EXCEEDS THAT ALLOWED
CP14	4014 - REQUIRED OR INCORRECT PASSWORD
CP15	4015 - I-D-S FILE IN ABORT STATUS
CP16	4016 - I-D-S FILE IN RECOVERY STATUS
CP17	4017 - SEEK ADDRESS CALCULATION ERROR
CP20	4020 - FAILURE IN NAME SCAN (IMP.)
CP21	4021 - UNDEFINED DEVICE (IMP.)
CP22	4022 - DEVICE LINK TABLE CHECKSUM ERROR
CP23	4023 - INCONSISTENT FDW BLOCK COUNT
CP24	4024 - INTERNAL LINK TABLE CHECKSUM ERROR

Table D-9. System Malfunction Messages During RETREAT Attempt (cont.)

Message Code	Definition
CP36	4036 - AFT IS FULL
CP37	4037 - DUPLICATE NAME IN AFT
CP40	4040 - NO PAT SPACE AVAILABLE
CP41	I/O CHECKSUM ERROR
CP42	I/O END-OF-FILE ERROR
CP43	I/O END-OF-LOGICAL-FILE ERROR
CP44	I/O UNDEFINED STATUS CODE RETURNED

System malfunction messages may occur during an attempt to CREATE a dataBASIC file. These messages are listed and defined in Table D-10.

Table D-10. System Malfunction Messages During CREATE Attempt

Message Code	Definition
CQ01	4001 - NAME NOT IN SYSTEM MASTER CATALOG
CQ02	4002 - I/O ERROR - CANNOT PROCEED
CQ03	4003 - PERMISSIONS DENIED
CQ04	4004 - FILE BUSY
CQ05	4005 - INCORRECT CATALOG/FILE DESCRIPTION
CQ06	4006 - LLINK SPACE EXHAUSTED
CQ07	4007 - DEVICE TYPE UNDEFINED
CQ12	4012 - SIZE REQUESTED LESS THAN CURRENT SIZE
CQ14	4014 - REQUIRED OR INCORRECT PASSWORD
CQ15	4015 - I-D-S FILE IN ABORT STATUS

Table D-10. System Malfunction Messages During CREATE Attempt (cont.)

Message Code	Definition
CQ16	4016 - I-D-S FILE IN RECOVERY STATUS
CQ17	4017 - SEEK ADDRESS CALCULATION ERROR
CQ21	4021 - UNDEFINED DEVICE (IMP.)
CQ22	4022 - DEVICE LINK TABLE CHECKSUM ERROR
CQ23	4023 - INCONSISTENT FDW BLOCK COUNT
CQ24	4024 - INTERNAL LINK TABLE CHECKSUM ERROR
CQ40	4040 - NO PAT SPACE AVAILABLE
CQ41	MORE THAN 100 DISC I/O ERRORS

System malfunction messages may also occur during an attempt to DESTROY a dataBASIC file. These messages are listed and defined in Table D-11.

Table D-11. System Malfunction Messages During DESTROY Attempt

Message Code	Definition
CS10	Subcatalog is present but cannot be purged. (Retreat and data files have been purged.)
CS20	Data file is present but cannot be purged. (Retreat file has been purged.)
CS30	Retreat file is present but cannot be purged.

A System Malfunction Message may occur during an attempt to ANALYZE a dataBASIC file. This message is entered and defined in Table D-12.

Table D-12. System Malfunction Messages During ANALYZE Attempt

Message Code	Definition
CR00	Unable to access the file specified.

Finally there are system malfunction messages which are possible during an attempt to verify a dataBASIC file. For a listing and definition of these errors, see Tables D-6 and D-7 of this Appendix.

APPENDIX E
PROGRAMMING CONSIDERATIONS

IMPLEMENTATION GUIDELINES

This appendix outlines some general considerations and guidelines to be followed when trying for higher dataBASIC programming efficiency and selecting more effective file loading techniques.

File Creation

A dataBASIC file consists of a subcatalog, referenced by the "dataBASIC file name," with two dependent random access files, .DATA0 and .JOUR. The .DATA0 file is the data file, and the .JOUR. file is the journal, or recovery, file. These files may be created by the CREATE subsystem, or they must be created through a direct file system activity when loading of a dataBASIC file is done with the Load/Unload system. Any password supplied to the CREATE subsystem will be assigned at the subcatalog level. The files .DATA0 and .JOUR. are without passwords. The .JOUR. file will contain 'before' images of all I-D-S pages modified during an update run. In the event of a system malfunction prior to completion of an update run, the pages written to the .JOUR. file during the current run will be restored on the .DATA0 file, thus obliterating all effects of this run.

To allow a user to quickly calculate the approximate number of links needed for a proposed dataBASIC file, the following formulae are provided:

$$L_D = \frac{R(F+4) \ + \ 7N(V+1)}{900}$$

$$L_R = \frac{50 + (L_D)}{10} \geq 5$$

where,

R = total records to be placed on the file

F = average number of fields per record

N = total unique field names on the file

V = average number of unique values per unique field name

L_D = size of the data file .DATA0 (in links)

L_R = size of the retreat file .JOUR. (in links) where L_R is never smaller than five links.

Normally a terminal user would not process such a large number of updates during any one run that the 'before' page images would exceed the allotted .JOUR. file space. In such an event, the user would be notified of a D177 System Malfunction and the contents of the file would be restored to their status at the beginning of the aborted run. In order to circumvent such a possibility, the user may wish to enlarge the size of the .JOUR. file. He may accomplish this by purging it, then recreating it through a file system activity with all attributes, other than the size, identical to those of the purged file. The file cannot merely be increased in size since the file system does not allow a random file to grow.

Loading Techniques

There are three methods for loading data onto a dataBASIC file. The first method involves use of the dataBASIC Load/Unload System. It is usually used when most of the data is to be loaded at file creation time when operation is in the batch world environment. For a full description of the Load/Unload system, see the dataBASIC Load/Unload System Implementation Guide, Document Number DA09.

The second method involves use of the Time-Sharing Media Conversion Program described in the GECOS Time-Sharing System General Information Manual, Document Number CPB-1643. Following this procedure, card-image input can be converted to time-sharing format stored on some pre-defined mass storage device. Using the TSS Editor, each line could then be converted to a dataBASIC-compatible DATA statement. Then these DATA statements could be appended to a dataBASIC program with appropriate READ and STORE statements which when executed would store the converted data as directed in a time-sharing environment.

The third method (usually used to store small amounts of data) allows for user-terminal interaction with the dataBASIC program while in execution. This is accomplished by use of the INPUT statement within the framework of the dataBASIC program, followed by appropriate READ and STORE statements.

Retrieval Techniques

Because of the nature of a dataBASIC data file, certain record selection techniques will be more efficient than others. It is an inverted file; that is, unique name:value pairs occur only once within the entire file in the portion of the file which is designated as the dictionary range. Whenever a record is stored containing an already used name:value pair, a pointer field is placed in the record range portion of the file linking the new record to the already present name:value pair in the dictionary range. It is important to realize that the name:value pair fields can be quickly accessed through a randomization algorithm used by the dataBASIC system. Once located, these fields point directly to their associated records stored in the record range. (Note that the dictionary and record ranges are physically placed at the lower and higher halves of the file respectively.) The records, however, appear in reverse order from that in which they are stored. Thus, if one wishes to access the first record stored on the file without first randomizing to some name:value pair appearing in that record, the dataBASIC system will have to follow a series of pointers, linking together all records on the file until it finds the one desired-in this case the last record to be encountered.

The following selection commands are those which will go to the dictionary range of the file first and will be most efficient:

```
FOR name  =value
FOR NOT  name = value
FOR name ALL
```

where name may be an actual field name or a working storage field suffixed by the character &, and value may be a literal or a working storage field.

In each of the above cases, an occurrence tally is maintained as the selected name: value pairs are found in the dictionary range and compared to a number equal to 70 percent of the total number of records on the file. If the occurrence tally is greater than this number, the record range portion of the file will be examined.

The FOR FNAME and FOR FVALUE commands, by definition, must always go to the dictionary range.

Examples of selection commands which interrogate the dictionary range include:

```
1)    100 FOR LAST:NAME = "SMITH"
2)    100 LET A = "SMITH"
      110 FOR LAST:NAME = A
3)    100 FOR LAST:NAME ALL
4)    100 FOR FNAME ALL
5)    100 FOR FNAME < 220
6)    100 FOR FNAME FROM  3 TO 110
7)    100 FOR FVALUE ALL
8)    100 FOR FVALUE  > 30
9)    100 FOR FVALUE FROM 50 TO 55
```

NOTE: The two examples which follow achieve the same results as the above examples but are considerably faster since they force interrogation of the dictionary range:

1)
```
         .
         .
         .
     200 FOR AGE FROM 21 TO 35
         .
         .
         .
     300 NEXT
         .
         .
         .
```

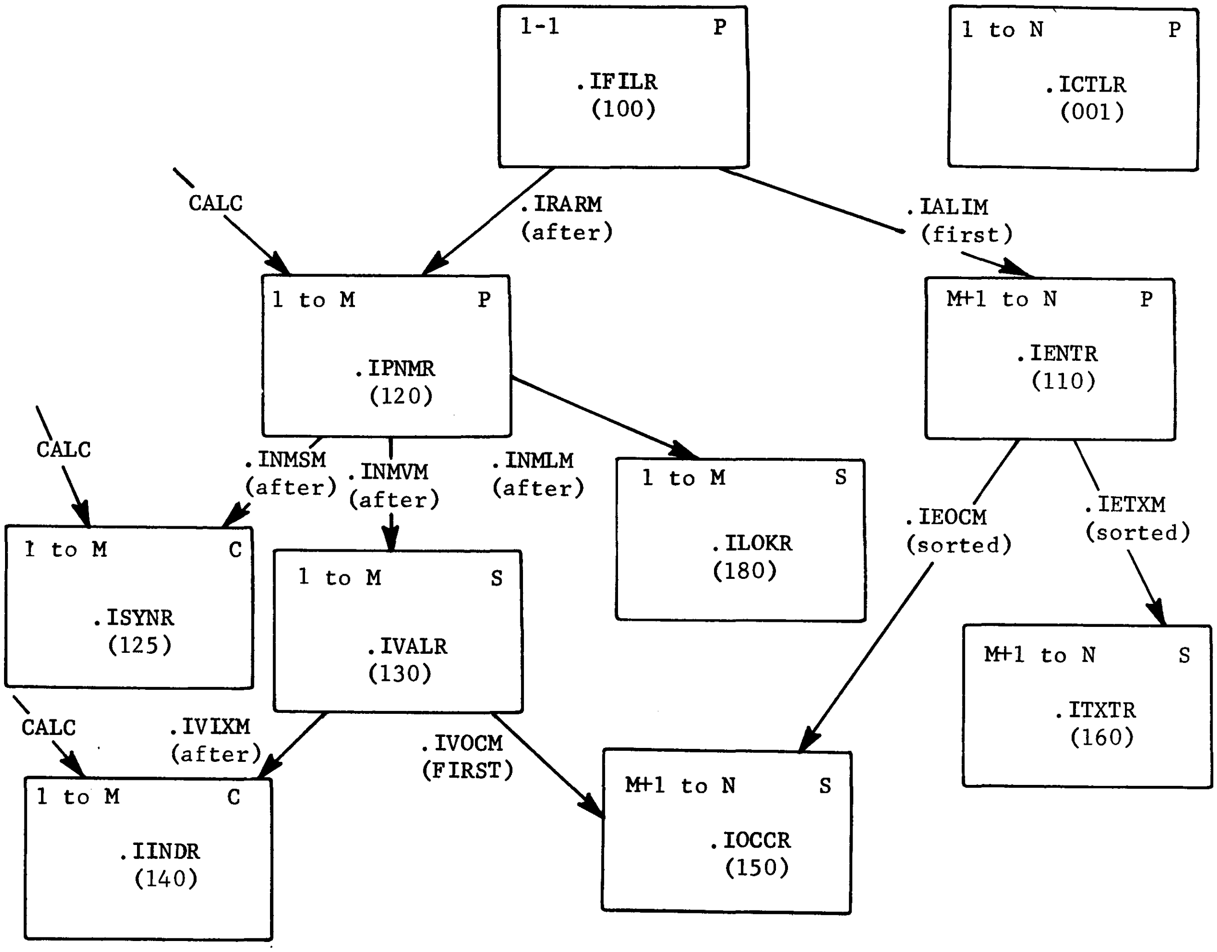
2)
```
         .
         .
         .
     200 LET A = 21
     210 FOR AGE = A
         .
         .
         .
     300 IF A < 35 THEN 320
     310 EXIT
     320 LET A = A+1
     330 NEXT
         .
         .
         .
```

SUBSYSTEM USAGE

The CREATE and DESTROY subsystems are to be used whenever a dataBASIC file is created or released. The ANALYZE and VERIFY subsystems are more specialized in that the usual user might never use them. It is important to realize that on large files the VERIFY subsystem may be very slow in terminal response time since it walks through all I-D-S chains on the file checking their integrity and tallying all associated records. When finished, a short summary report will be printed at the terminal.

APPENDIX F

dataBASIC FILE STRUCTURE



Records

.IFILR - file record
.IPNMR - prime name record
.ISYNR - synonym name record
.IVALR - value record
.IINDR - index record
.IENTR - entity record
.IOCCR - occurrence record
.ITXTR - text record
.ICTLR - control record

Chains

.IPARM - pair chain
.IALLM - all chain
.INMSM - name synonym chain
.INMVM - name value chain
.INMLM - name lock chain
.IVIXM - value index chain
.IVOCM - value occurrence chain
.IEOCM - entity occurrence chain
.IETXM - entity text chain

Figure F-1. dataBASIC File Structure

COMPUTER GENERATED INDEX

COMPUTER GENERATED INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form*

TITLE: SERIES 6000/600
dataBASIC SYSTEM
LANGUAGE MANUAL

ORDER NO: DA08, Rev. 0

DATED: MAY, 1971

ERRORS IN PUBLICATION:

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

(Please Print)

FROM: NAME ______________________ DATE ____________

COMPANY ______________________

TITLE ______________________

ADDRESS ______________________

____

*Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here. □

CUT ALONG LINE

CUT ALONG LINE

POSTAGE WILL BE PAID BY:

HONEYWELL INFORMATION SYSTEMS

60 WALNUT STREET
WELLESLEY HILLS, MASS. 02181

ATTN: U.S. GROUP PUBLICATIONS, MS 050

Honeywell

The Other Computer Company:

Honeywell

HONEYWELL INFORMATION SYSTEMS

2124
2671
Printed in U.S.A.