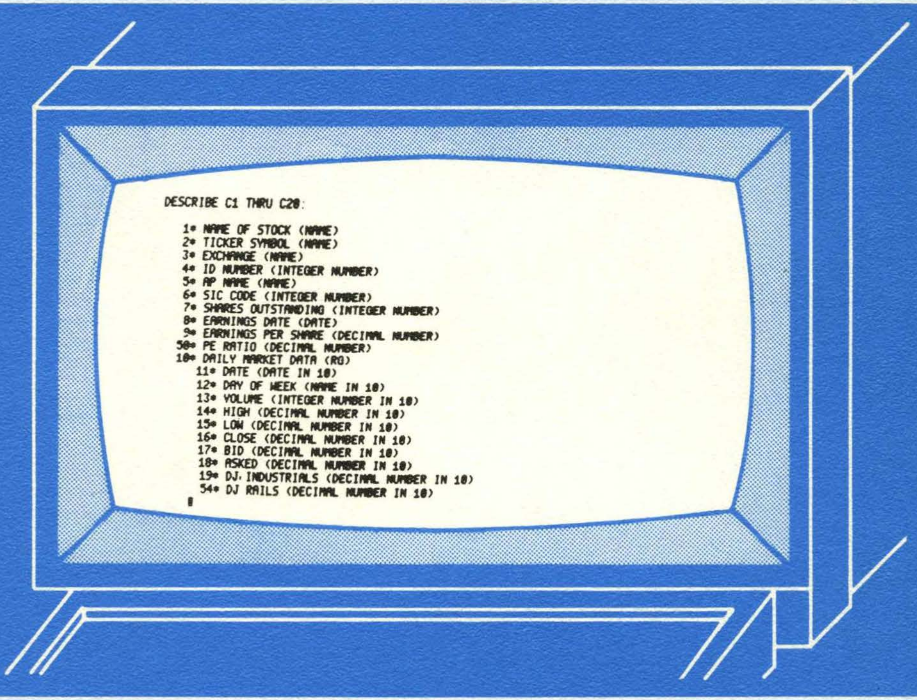
user information manual

Control Data



```
DESCRIBE C1 THRU C20:

   1* NAME OF STOCK (NAME)
   2* TICKER SYMBOL (NAME)
   3* EXCHANGE (NAME)
   4* ID NUMBER (INTEGER NUMBER)
   5* AP NAME (NAME)
   6* SIC CODE (INTEGER NUMBER)
   7* SHARES OUTSTANDING (INTEGER NUMBER)
   8* EARNINGS DATE (DATE)
   9* EARNINGS PER SHARE (DECIMAL NUMBER)
  50* PE RATIO (DECIMAL NUMBER)
  10* DAILY MARKET DATA (RG)
    11* DATE (DATE IN 10)
    12* DAY OF WEEK (NAME IN 10)
    13* VOLUME (INTEGER NUMBER IN 10)
    14* HIGH (DECIMAL NUMBER IN 10)
    15* LOW (DECIMAL NUMBER IN 10)
    16* CLOSE (DECIMAL NUMBER IN 10)
    17* BID (DECIMAL NUMBER IN 10)
    18* ASKED (DECIMAL NUMBER IN 10)
    19* DJ. INDUSTRIALS (DECIMAL NUMBER IN 10)
    54* DJ RAILS (DECIMAL NUMBER IN 10)
 ▮
```

SYSTEM 2000™

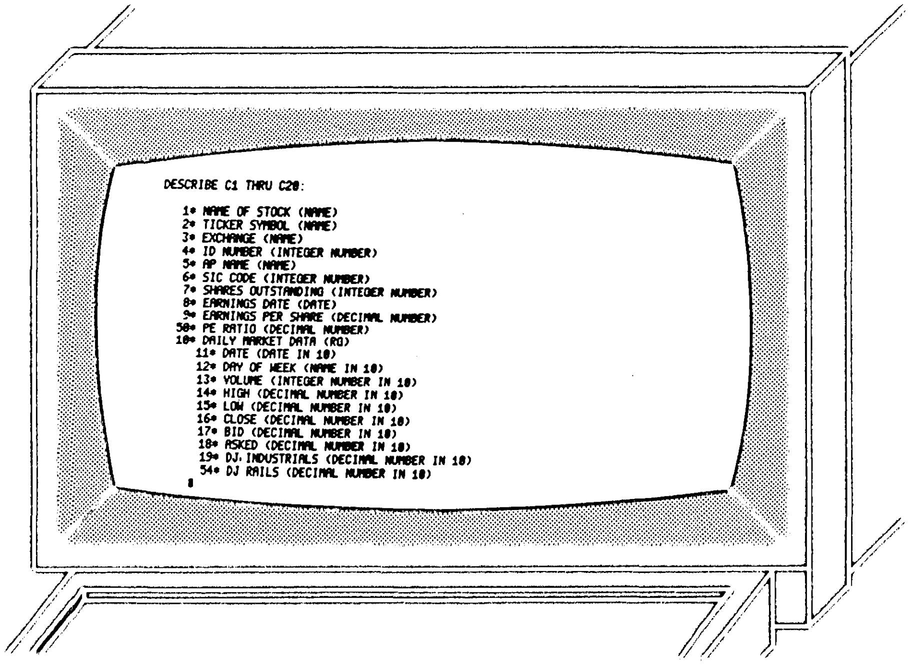
A MULTI-PURPOSE DATA MANAGEMENT SYSTEM FOR THE CDC® 6600 COMPUTER SYSTEM

(PRELIMINARY)

DATA SERVICES DIVISION

CONTROL DATA
CORPORATION

user information manual



```
DESCRIBE C1 THRU C20:

   1* NAME OF STOCK (NAME)
   2* TICKER SYMBOL (NAME)
   3* EXCHANGE (NAME)
   4* ID NUMBER (INTEGER NUMBER)
   5* AP NAME (NAME)
   6* SIC CODE (INTEGER NUMBER)
   7* SHARES OUTSTANDING (INTEGER NUMBER)
   8* EARNINGS DATE (DATE)
   9* EARNINGS PER SHARE (DECIMAL NUMBER)
  50* PE RATIO (DECIMAL NUMBER)
  10* DAILY MARKET DATA (RG)
     11* DATE (DATE IN 10)
     12* DAY OF WEEK (NAME IN 10)
     13* VOLUME (INTEGER NUMBER IN 10)
     14* HIGH (DECIMAL NUMBER IN 10)
     15* LOW (DECIMAL NUMBER IN 10)
     16* CLOSE (DECIMAL NUMBER IN 10)
     17* BID (DECIMAL NUMBER IN 10)
     18* ASKED (DECIMAL NUMBER IN 10)
     19* DJ. INDUSTRIALS (DECIMAL NUMBER IN 10)
     54* DJ RAILS (DECIMAL NUMBER IN 10)
  ▮
```

SYSTEM 2000™

A MULTI-PURPOSE DATA MANAGEMENT SYSTEM FOR THE CDC® 6600 COMPUTER SYSTEM

(PRELIMINARY)

DATA SERVICES DIVISION

CONTROL DATA
CORPORATION

FOREWORD

SYSTEM 2000^{TM} is a multi-purpose data management system that was developed by Management Research International, Inc. (Austin, Texas). It is designed to run on the CDC® 6600 Computer Systems within CDC Data Services' nationwide network of computers and terminals (CYBERNET Service).

This preliminary user manual is divided into two main sections: (1) a SYSTEM 2000 Reference Manual, copyrighted by Management Research in 1970, and (2) a summary of SYSTEM 2000 diagnostic messages, also copyrighted by Management Research in 1970. This material is republished by CDC Data Services with the permission of Management Research.

A postcard is included at the end of this manual for mailing to Management Research. This procedure will guarantee that the reader will be sent appropriate updates for the most recent version of SYSTEM 2000. Additionally, each CDC Data Center (that maintains a CDC 6600) employs an anlyst who is trained in the use of SYSTEM 2000. Consequently, the reader can contact his nearest 6600 Data Center if he has any questions regarding this manual or SYSTEM 2000.

[redacted] 9 June 1970 RM-S2K-1.3

SYSTEM 2000™

Reference Manual

TABLE OF CONTENTS

Page

LIST OF FIGURES

1.0 INTRODUCTION

SYSTEM 2000 is a multi-purpose data management system which has the capability and flexibility to provide a complete data management service to business and government. It contains the ability to perform data base development, maintenance, functional computations, and complex information retrieval activities.

SYSTEM 2000 operates on large volume, high speed computers under the control of the computer's operating system. SYSTEM 2000 operations are controlled by the Executive, which allows access to four functional modules: DEFINE, LOADER, RETRIEVAL and UPDATE. The Executive contains the system-wide commands used in the development and overall operational utilization of user defined data bases. The first functional module, DEFINE, allows the user to structure his own data base by defining the data base components. The LOADER module is used to load new or existing data files into the system. The RETRIEVAL module contains the command language providing the ability to gain access to any stored data item, to compare data, to perform functional computations and to output the data in either standard list format, or by use of a post processor, output the data in a user defined report format. The UPDATE module allows a complete range of data base maintenance activities including adding, deleting, changing and inserting of data.

The system provides throughout its varied capabilities a user oriented concept. All of the communications language within SYSTEM 2000 uses the English language, in all of the commands within all of the modules, as well as within the complete range of system diagnostics.

2.0 EXECUTIVE

The Executive contains all of the system-wide commands which allow access to all system capabilities. Any of these commands are available for use any time the user has access to the system. This chapter will discuss each of these commands which are divided into four job-related sections. In addition to the system-wide commands, the system-wide default settings are given.

2.1 SYSTEM-WIDE COMMANDS

SYSTEM 2000 system-wide commands are classified into four types, with each type including a number of specific commands. All of the system-wide commands have at least one thing in common as suggested by their title: they can be used at any time the user has access to the system, regardless of which functional module is active. Each functional module has its own module specific commands which are appropriate only when that module is active and are used to carry out the various operations of the module.

All SYSTEM 2000 commands have a common format appearance to the extent that each command ends with a colon.

2.1.1 Module Commands

There are four functional modules and each is called into service by a one-word command, as follows:

```
DEFINE:
LOADER:
RETRIEVAL:
UPDATE:
```

Each of these commands and their associated uses is discussed quite fully within the individual chapters reserved for the four functional modules. The system allows selecting more than one module in any one job session.

2.1.2 Change Parameter Commands

There are two system-wide commands in the parameter change group. They concern the ability to change the system separator and the entry terminator.

2.1.2.1 System Separator

Purpose To change the system separator.

Command SEPARATOR IS <separator symbol>:

Discussion SYSTEM 2000 uses the system separator in practically all of its operations and it is used to separate data values from other system items. The default system separator is the asterisk ("*"). Unless the asterisk will appear as a data value within a data base, the user need not change the system separator from the asterisk to some other symbol.

The purpose of the system separator is to separate data; therefore it cannot appear within the data values. If the user has a need to use the asterisk within his data, he may change the system separator at any time to any one of the characters listed below:

(1) ≡
(2) [
(3)]
(4) ≠
(5) →
(6) ∨
(7) ∧
(8) ↑
(9) ↓
(10) <
(11) >
(12) ≤
(13) ≥
(14) ¬

The system separator used by the user in the development of his data base is stored within the data base tables for system reference.

2.1.2.2 Entry Terminator

Purpose To change the entry terminator.

Command ENTRY TERMINATOR IS <entry terminator word>:

Discussion The entry terminator word is used when a user wants to signal the end of all data for a logical entry or a data string. As such, it is employed within the LOADER and UPDATE modules. The terminator word may be any combination of alphanumeric or special characters, up to a maximum of 10. Whenever the entry terminator is used, it is always associated with a double system separator preceding it with a mandatory blank prior to the system separators. By default, the standard entry terminator word is END. Therefore, by default the standard entry terminator is as follows:

Δ**END

Much freedom in the selection of the terminator word is possible because the same word may occur frequently within data bases without causing ambiguity.

2.1.3 User File-Name Commands

SYSTEM 2000 utilizes four files over which the user can exhibit some direct control.

2.1.3.1 Input Files to SYSTEM 2000

All commands, requests and data that the user sends to SYSTEM 2000 are given to the system on one of two files. The DATA FILE contains only the loader data input string, and the COMMAND FILE contains all user commands and requests. Commands are available to establish the location where SYSTEM 2000 should go to find these input files. The COMMAND FILE, by default, is named INPUT, which identifies the card reader or remote input device as the source of user commands and requests. The DATA FILE must be placed on a file other than INPUT before calling the LOADER module. If an operating system control card has requested either file to be a certain magnetic tape prior to calling SYSTEM 2000, then the system will read commands or data from that tape. The commands available to signal a change in either of the input file names are:

```
COMMAND FILE IS <file name>:
DATA FILE IS <file name>:
```

where <file name> is any standard 7-character file name of up to 7 characters, beginning with an alphabetic character, but may not be TAPExx, i.e., "TAPE3". To restore the COMMAND FILE dynamically back to the card reader during a job where many files are used, the user may give the following command:

```
COMMAND FILE IS INPUT:
```

File name changes are instantaneous upon command and remain in effect until another file name command for the file is encountered in the job.

2.1.3.2 Output from SYSTEM 2000

All output from SYSTEM 2000 including messages, comments and retrieval results are sent to the user on either the MESSAGE FILE or the REPORT FILE. The purpose, then, of the two output file commands is to specify the physical location or device where SYSTEM 2000 should send its output. These two output files, by default, are set to be the printer or remote output device for local batch or remote batch jobs. Messages and comments are returned to the user on the MESSAGE FILE; retrieval results are returned on the REPORT FILE. If both files are under the same default file or if the user sets both files to be under the same file name, then messages, echos of commands and retrieval results will appear in the order of their occurrence during processing. Either output file may be set to a file name associated with magnetic tape if a tape request card declaring the file is processed before calling SYSTEM 2000. Retrieval results are sent to the REPORT FILE under a standard SYSTEM 2000 format; if the REPORT FILE is specified (and saved as a tape, common, or permanent file externally by the operating system control cards), it can then be read by any external program. The commands available to signal a change in the output file names are:

```
MESSAGE FILE IS <file name>:
REPORT FILE IS <file name>:
```

where file name conforms to a legal operating system file name, and is not TAPE <xx>. If during a job, the user wishes to dynamically restore either output file back to the printer, he may give the following commands:

```
MESSAGE FILE IS OUTPUT:
REPORT FILE IS OUTPUT:
```

Often it is convenient or necessary to produce a "clean" report containing no messages or echoes of retrieval requests; by simply setting the MESSAGE FILE to some "dummy" file name, the REPORT FILE only is displayed in pure form on the printer. Another desirable use of diverting output often happens when unloading a data base; in this case, the REPORT FILE can be diverted to a user-specified file and saved for further processing.

SYSTEM 2000 is capable of interfacing with any I/O device such as an optical scanner, 252 display scope, etc., when these devices are available.

2.1.4 Data Base Control Commands

The fourth and final type of system-wide command deals with data base access, control and manipulation.

2.1.4.1 Password Command

Purpose To establish legal access to SYSTEM 2000.

Command USER,<Password>:

Discussion This is the first SYSTEM 2000 card in the job deck structure for local or remote batch operations. All preceding cards in the job deck structure are operating system control cards. Users are assigned passwords, which change from time to time for security purposes. The system checks the legality of the password, and, if honored, allows access to the system. The password is an assigned 5-character word and maybe one of two types. Certain passwords permit the user to Define, Load, Retrieve and Update a data base; other passwords are Retrieval-only passwords. Assignment of these passwords is generally the responsibility of the computer center management.

2.1.4.2 Access Data Base

Purpose To gain access to an established data base.

Command There are two separate and distinct system-wide commands that have this common purpose.

(1) DATA BASE NAME IS <data base name>:

The user has created a data base, named it, probably stored it and now wants to access it with this command. The data base name is originally assigned by a DEFINE module specific command, NEW DATA BASE <data base name>:. The data base name is limited to 20 characters or less including blanks when it is originally assigned. Thereafter, it is referred to by that same name. When the user issues this command, the system first checks to see if the data base is available on the disk and then checks the user password to see if the user should be allowed access to this data base. If not, a diagnostic message will be output. If access is available and legal, the system automatically assigns the data base and the RETRIEVAL module to the job.

(2) DATA BASE COPY IS <data base name>:

This command does exactly what the previous command did, except, it accesses the copy of the data base which was created by the system-wide command, CREATE COPY: (to be discussed).

2.1.4.3 Save Data Base

Purpose To copy the data base tables from the disk file to a tape file. Secondarily, to specify the Update File Tape visual reel number.

Commands SAVE DATA BASE ON <dbtvr#>:, or
SAVE DATA BASE ON <dbtvr#>/<uftvr#>:

where: (1) <dbtvr#> = data base tape visual reel number
(2) <uftvr#> = update file tape visual reel number (optional)

Discussion When a data base has been created by use of the DEFINE and LOADER module, the data base exists on disk as eight files associated with a ninth file, called the working update file (originally empty). The user can save the data base he is currently accessing by issuing this command, issued at any time in the life of the data base. The user must specify the data base tape visual reel number <dbtvr#>, but the specification of the update file tape visual reel number <uftvr#> is optional. If the <uftvr#> is not specified, the working update file is automatically suspended. (See Section 6.3 for a more detailed discussion of the use of the update file.)

2.1.4.4 Load Data Base

Purpose To load a data base from tape to disk.

Commands LOAD <data base name> FROM <dbtvr#>:, or
LOAD <data base name> FROM <dbtvr#>/<uftvr#>:

where: (1) <dbtvr#> = data base tape visual reel number
(2) <uftvr#> = update file tape visual reel number (optional)

Discussion The logic of this command follows directly from the logic of the Save Data Base command. Before a Load Data Base command is legal or logical, the data base must have been saved. This command must specify the <dbtvr#>, the same reel number specified when the associated Save Data Base command was issued. If the Save Data Base command indicated the <uftvr#>, the Load command need not repeat it. If, however, the Save command did not specify the <uftvr#>, then the Load command can include it as an optional item if the user wishes to record update segments on the working update file. If neither command specifies the Update File Tape Visual Reel Number, then the working update file is automatically in suspend mode.

2.1.4.5 Create Copy

Purpose To copy the nine data base tables of the accessed data base to nine new tables, creating two identical data bases both residing on disk.

Command CREATE COPY:

Discussion This command assumes the user has previously accessed a data base. The user may wish to create a copy of the data base which can be manipulated or tested so that such manipulation will not disturb normal use of the standard data base. The original data base is known as the standard data base and the copy is known as the test data base. The test data base concept lends itself to controlled update processing. As soon as the command has been processed and the test data base has been created, the user is attached to the test rather than the standard data base.

If the user submits a subsequent job, the test data base may be accessed by the command:

```
DATA BASE COPY IS <data base name>:
```

If the user desires to replace the standard copy with the test copy, the two following commands are given:

```
DATA BASE NAME IS <data base name>:
RELEASE:
```

Disk storage allocated to the standard data base is released and the test data base automatically becomes the standard data base.

2.1.4.6 Release

Purpose To purge the nine data base tables of the accessed data base residing on disk.

Command RELEASE

Discussion If the user wishes for whatever reason to release the data base tables from the disk, he merely needs to access the appropriate data base by one of two commands:

DATA BASE NAME IS <data base name>:, or
DATA BASE COPY IS <data base name>:

and issue the Release command. If he would like to keep an archival copy of the released data base, he must issue a Save Data Base command before giving the Release command.

2.1.4.7 Erase

Purpose To purge the nine data base tables of the accessed data base residing on disk (RELEASE), and remove from the system all reference to the name of the standard or test data base which was accessed.

Command ERASE:

Discussion If the user wishes to perform a RELEASE as well as erase the data base name from the system, this command should be performed.

2.2 SYSTEM-WIDE DEFAULT SETTINGS

The system-wide default settings for SYSTEM 2000 are listed here. Each functional module has additional default settings which are introduced within the appropriate module discussion.

1. The standard separator symbol is the asterisk, "*."
2. The separator symbol will be the standard separator, ("*"), or the symbol last associated with the data base unless the SEPARATOR IS <separator symbol>: command is given. The current separator symbol is saved with the data base when a SAVE DATA BASE command is given.
3. The standard entry terminator word is END.
4. The entry terminator will be the standard entry terminator or the entry terminator last associated with the data base unless the

ENTRY TERMINATOR IS <entry terminator>: command is given. The current entry terminator is saved with the data base when a SAVE DATA BASE command is given.

5. The COMMAND FILE (containing all SYSTEM 2000 requests and commands) is initialized to INPUT which means cards for local or remote batch.

6. The DATA FILE (containing the loader input string) is initialized to INPUT, meaning cards or keyboard input and must always be set by the user to the name of the file containing the loader input string when using the LOADER module. (See LOADER module chapter.)

7. The following is the set of commands which may legally follow the USER command:

```
DATA BASE NAME IS <data base name>
DATA BASE COPY IS <data base name>
LOAD <data base name> FROM <dbtur#>
LOAD <data base name> FROM <dbtur#>/<uftvr#>
DEFINE:
```

   The first four commands are used to establish access to an already-existing data base. The fifth command may be used to permit the definition of a new data base to be specified (See Section 3.4.1.2).

8. The user password used during the creation of a new data base is identified within the system as the only password authorized to access the data base. That password may be used to assign other valid passwords for the accessed data base, but only the creating password may use other than the RETRIEVAL module with that data base. The new valid passwords assigned by the creating password may only operate in the RETRIEVAL module.

9. A test data base, formed by the CREATE COPY: command, can only be created by the password which created the original or standard data base.

10. If an existing data base is loaded with a LOAD <data base> command, that data base name is not associated with the password unless the DATA BASE NAME IS <data base name> command has been given.

11. The RETRIEVAL module is automatically available to the user after loading or naming a data base.

12. The MESSAGE FILE (containing error messages and echoes of user commands) is initiated to the OUTPUT FILE which is the on-line printer or remote output device for local batch or remote batch operations.

13. The REPORT FILE (containing retrieval output and LOADER reports) is initiated to the OUTPUT FILE which is the on-line printer for local batch or remote batch operations.

14. The user's password must be correctly specified on every job.

15. The use of SAME across the RETRIEVAL and UPDATE modules:

    a. SAME implies use of the results of the last WHERE clause containing Boolean conditions.

    b. It does not mean process the previous WHERE clause again.

    Therefore, if the following series of requests are issued, then the expected results should be:

```
PRINT C1 WHERE C1 EQ JONES:
1* JONES
UPDATE:
CHANGE C1 EQ SMITH **END WHERE SAME:
RETRIEVAL:
PRINT C1 WHERE SAME:
1* SMITH
```

    (If the WHERE clause were reprocessed each time SAME occurred, then in the above example, the last result would be "No Output Found.")

3.0 DEFINE MODULE

3.1 INTRODUCTION

The DEFINE module has been designed to assist in solving the first direct question facing the user of SYSTEM 2000: How will I organize or structure my data which I wish to load into my data base(s) and later access in retrievals and updating? Learning how to optimize all of the DEFINE module powers within the particular environments in which it is used will take some study, but learning the mechanics of actually doing it is quite a simple task. The first and really the only step required is to define the data base.

What is a data base? One definition which may help is as follows: A data base is an organized collection of data about something. In SYSTEM 2000, we solve the problem by letting the user define what that organization will be. Since different types of data suggest different organizations, SYSTEM 2000 provides the user the flexibility to organize or structure his own data base.

3.2 DATA BASE STRUCTURE

The data base is structured by the user to solve the user's problems and to answer his questions. He defines his own data base using the tools of the DEFINE module. He does this by choosing appropriate words which will stand for the different types of data which he will store. The user will probably be storing numerous quantities of what we call logical entries. A logical entry is all the information about one of the major items being stored. Examples will help at this point. If a school is developing a data base of student records, a complete student record about one student would be a logical entry. If a government housing development has a data base, a housing project with all of the tenants might be the logical entry. If the courts are building a data base to assist their court calendar scheduling, an individual docket within the court system might be the best logical entry. A data base developed to assist in portfolio management might solve the problem of designing the logical entry to be about ORGANIZATIONS (large investment firms), which have several PORTFOLIOS, which have many STOCKS. In that case, a logical entry would be an ORGANIZATION.

A logical entry is what the name suggests, an entry of data logically related to solve the user's problem. A data base, then, consists of the total collection of all logical entries. Data bases may have many logical entries or only a few. It is strictly up to the user.

A graphic illustration of a portfolio logical entry would look something like Figure 1. Within this simplified graphic of a logical entry or data tree, the levels of information, i.e., the hierarchy of the user's source data is shown. Level zero is at the top, level one is next and so on such that our illustration demonstrates four levels. SYSTEM 2000 is so designed that it can accept and associate data nested hierarchically to 64 levels with numerous categories at each level.

These many levels of data and the detailed classification of the data within levels is done by the user through the data base definition, which consists of an orderly arrangement of component names or labels. These labels indicate the type of data which the user will be loading into the data base; they are not the data values, but, instead, become identification tags which the user employs in accessing his data. But, before we get too involved, let us discuss the different components available to the user.

3.3 DATA BASE COMPONENTS

There are four types of components within SYSTEM 2000: elements, repeating groups, user-defined functions, and strings.

3.3.1 Elements

Elements are components which permit the naming and grouping of data stored in the data base. We will be developing in this chapter the definition of a portfolio data base.

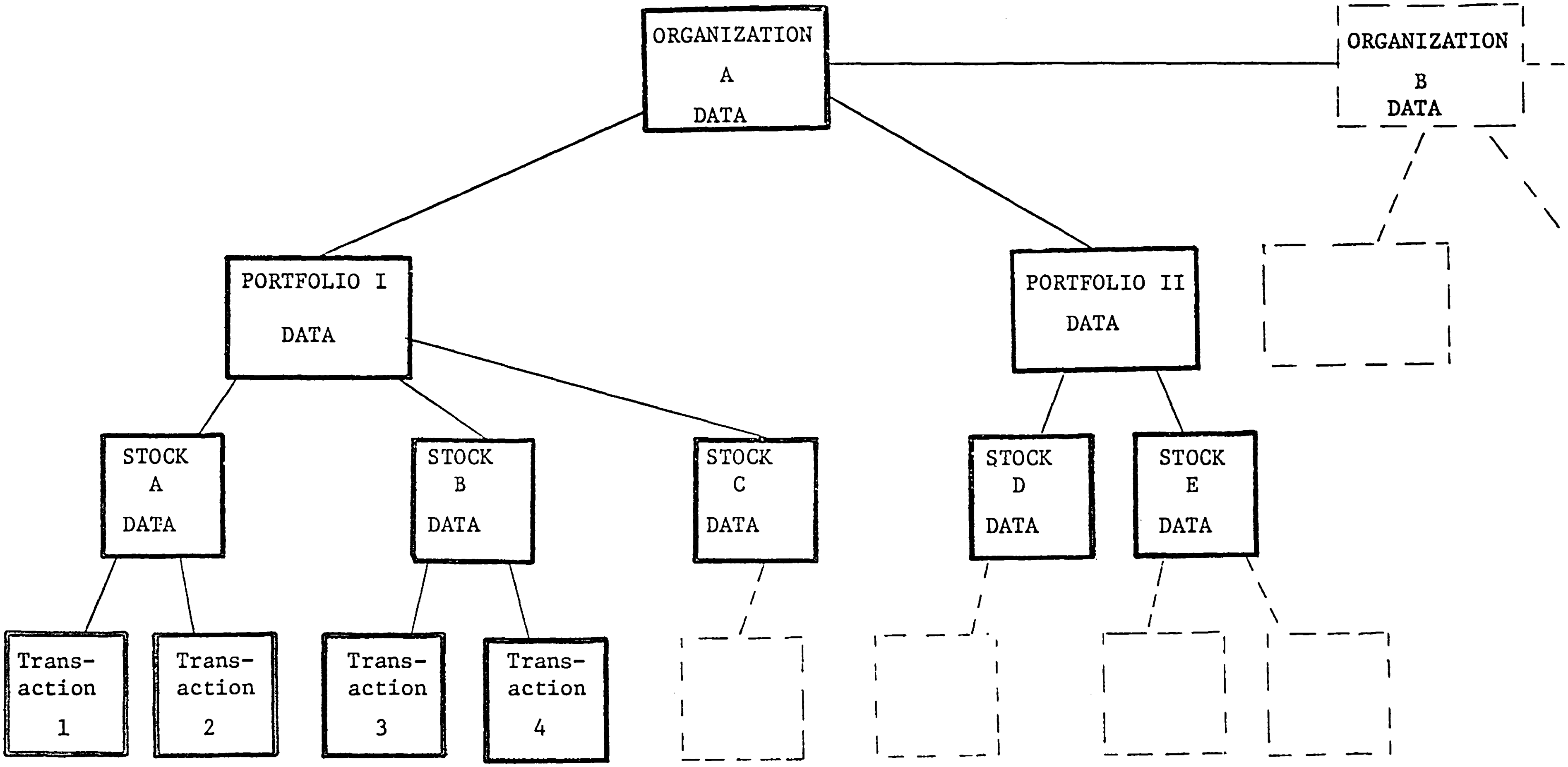


FIGURE 1

Logical Entry Graphic Illustration

One of the types of data we will be storing in this data base are stocks held in the portfolio. The names of the stocks held will be an element, which will be called NAME OF STOCK. As you can see, this component will undoubtedly contain many values, but we identify it by one element name, and a user-assigned number which appears before the element name. Elements can take on several different types of values. They are:

1. Name - Any alphanumeric data with all leading, trailing and extraneous (more than one) embedded blanks discarded when the data is stored. (255 characters maximum)

2. Text - Any alphanumeric data with all blanks being retained in the data. Text is the element type selected when storing reports or graphics where control of blank storage is imperative to maintain the original appearance of the material. (255 characters maximum)

3. Date - MM/DD/YYYY or MM/DD/YY, standing for month, day and year. If only two numbers are used for the year, the system assumes 1900, until the century changes. Dates before the advent of the Gregorian calendar, October 15, 1582, cannot be stored as type "date." Each date in the data base is stored as the number of days elapsed since 10/15/1582 (day zero).

4. Decimal Number[1] - A positive or negative string of numbers with a decimal point. No decimal number may exceed 15 characters, including the sign and the decimal point.

5. Integer Number[1] - Any string of numerals (0-9) up to 15 characters, including the sign.

6. Exponential Number[1] - The form for this element type is ± nn.nnE±mm where both n and m are any integers (0-9). The "E" is used to denote mm to be the base 10 exponent. Again, 15 characters are allowed including the sign.

[1] When the sign is omitted, it is assumed to be positive.

Actual examples of defined elements are as follows:

1* ORGANIZATION (NAME)
6* ZIP CODE (INTEGER NUMBER)
7* CURRENT DATE (DATE)

The preceding number in each case is a user-defined number which is called the component number. Therefore, the first element may be referred to by ORGANIZATION or C1. The asterisk is the default system separator.

All component names, which are always user-defined, may contain 1 to 50 characters. The user may wish to avoid lengthy component names, as they may become cumbersome during retrieval and update operations. Also, the following list of words and symbols cannot be used in any component name:

1. Logical Operators (AND,NOT,OR)
2. Relational Operators (EQ,NE,LT,LE,GT,GE)
3. AT, SAME
4. SPAN, SPANS, SPANNING
5. FAIL, FAILS, FAILING
6. EXIST, EXISTS, EXISTING
7. HAS, HAVE, HAVING
8. Connectors (WHERE, BEFORE, AFTER)
9. Symbols (comma, colon, system separator, parentheses)

Each of the above words or symbols occur somewhere within the system commands syntax. If these words or symbols were allowed within a component name and that component name were used within a RETRIEVAL or UPDATE command, the system could not distinguish between these words and would assume that they were part of a system command, which would result in an error diagnostic message.

The parenthetical expression following the element name designates the type of data which will be given to the element as previously discussed. It is within this parenthetical expression that SYSTEM 2000 keeps track of the relationships of elements to each other and between the elements and the entire logical entry. This will be explained in the next section dealing with repeating groups.

In summary, the general form of an element definition, or for any component for that matter, is as follows:

- component number (numbers from 1 to 9999)
- system separator
- component name
- component type description (in parentheses)

3.3.2 Repeating Groups

The second component type is the repeating group (sometimes abbreviated RG[1]) which associates elements or other lower level repeating groups, and cannot take on a data value. This is the component which allows elements to take on multiple values. In the portfolio data base mentioned earlier, we indicated the element, NAME OF STOCK. Each stock in a portfolio is going to have several buy and sell transactions. In order to keep these organized, we will assign a repeating group component the name of TRANSACTIONS and assign several elements to be in that TRANSACTIONS repeating group, namely TRANSACTION TYPE (which will take on data values like BUY and SELL), DATE (which will be the date of the buy or sell), SHARES (which will be the actual number of shares bought or sold) and the PRICE (which will be the buy or sell price). Like elements, repeating groups are assigned identification numbers as are all components.

Actual examples of defined repeating groups combined with the defined elements shown previously are as follows:

```
 1* ORGANIZATION (NAME)
 6* ZIP CODE (INTEGER NUMBER)
 7* CURRENT DATE (DATE)
 8* PORTFOLIOS (RG)
    9* PORTFOLIO NAME (NAME IN 8)
   11* MANAGER (NAME IN 8)
   12* STOCKS (RG IN 8)
       13* NAME OF STOCK (NAME IN 12)
```

[1] If REPEATING GROUP is used within a SYSTEM 2000 command, the abbreviation RG may always be used in place of the complete term.

In the examples given, repeating groups have been defined as components 8 and 12. Also note the numbers now appearing in the parenthetical expressions. Components which are located at level zero in the hierarchical structure of SYSTEM 2000 have no linkage numbers because they reside at the top. In addition, those elements which reside at level zero are said to be in an assumed repeating group, called ENTRY or CØ (zero). All descendants from the level zero repeating group require numbers in the right-hand parenthetical statement to show their related linkage. In this example, components 9, 11 and 12 are in repeating group C8, PORTFOLIOS. Additionally, component 13 is an element within the repeating group C12, STOCKS. Elements and repeating groups below level zero are always linked with or tied to repeating groups. Components may be declared "in" any repeating group at any point in the definition as long as the "parent repeating group" has previously been declared. Indentation by level is controlled by the system.

3.3.3 User-Defined Functions

The third type of component is the user-defined function. Functions, defined by the user, provide problem solving capabilities. A function is any arithmetic expression consisting of element numbers, element names and constants separated by arithmetic operators and parentheses. The function may be used only during retrieval operations and permits the result of the defined computation to be output to the user. Since such computation takes place at retrieval time, it is not necessary to store the computed functional value in the data base as an element.

The allowable arithmetic operators in their indicated order of operation associated with the symbol used in the definition, are as follows:

ORDER	SYMBOL	OPERATION
First	↑	exponentiation (e.g., C3↑2 = $C3^2$)
Second	*	multiplication (e.g., C1*C2 = C1 x C2)
	/	division (e.g., C4/C20 = $\frac{C4}{C20}$)
Third	+	addition
	-	subtraction

All of the preceding operations may be defined within nested parenthetical expressions as necessary. Nested operations are processed first, before any check is made of the normal order of processing.

Functions are described by the user in his data base definition, and apply only to that data base. The general form of a function definition is identical to the format for other component definitions, i.e., component (function) number, system separator, component (function) name, (maximum of 50 characters) and type description. An example function description is:

30* DOLLAR AMOUNT (DECIMAL FUNCTION (C28 * C29 / 100))

where

30 is the user defined function number,

* is the system separator,

DOLLAR AMOUNT is the user defined function name,

DECIMAL FUNCTION is the component type, and

(C28 * C29 / 100) is the arithmetic expression, with 100 being a defined constant. C28 and C29 are component numbers standing for component names. Either component names or numbers may be used.

The component types which can be used in defining a function are an INTEGER, DECIMAL, or EXPONENTIAL NUMBER, or a DATE. The referenced components (e.g., C28 and C29) must exist in the definition before they can be referenced. The type of function (INTEGER, etc.) determines the output format of the function value. If the numeric function type is not declared, DECIMAL is assumed. Output formats for the three numeric types of functions are:

DECIMAL FUNCTION	xxxxxxxxxx.xxxxxxxxxx (10 significant digits plus decimal point)
INTEGER FUNCTION	xxxxxxxxxxxxxxx (15 digits max)
EXPONENTIAL FUNCTION	x.xxxxxxxxx E ± nn (10 digits max)

Functions are numbered independently of other components; therefore, a function may be assigned the same number as other components within the data base definition. To distinguish between other components, the functions are referred to by their F number rather than their C number. Therefore, function number 30 would be referred to as F30, and not C30. They, of course, may be referred to by name also. When describing a function, no association with a repeating group is explicitly defined. In other words, all functions are defined as if they were level Ø components. In the retrieval output, however, functions have an implied level and RG association. The implied level and RG association of a function is dictated by the deepest level element appearing in the function definition. For example, if a function contains a level Ø and a level 2 element, the function will be evaluated for every occurrence of the level 2 element and thus will be output as a level 2 function associated with the RG parent of the level 2 element.

3.3.4 Strings

A string is a user defined component which can be assigned to identify a long string of components used frequently in the RETRIEVAL module; a string may also contain several complete commands which are to be used in sequence during retrievals. This is particularly useful when the user has a repetitive need for a lengthy retrieval and would like to refer to this lengthy retrieval by some shortened code word.

The format for the strings definition is similar to the other components, as indicated below:

- ° component (string) number (can be a duplicate of any other component number, because when referred to, it is used with an S prefix, i.e., S30.)
- ° system separator
- ° component (string) name (maximum of 50 characters)
- ° component type (STRING (maximum of 700 characters))

An example of the use of strings would be the case of the user who needed to frequently retrieve information using the following command:

```
PRINT C1, C2, C3, C4, C10, C25, C26, C27, F30
WHERE PORTFOLIO NAME EQ INCOME:
```

We are going to define this retrieval statement in a string and assign the code word REPORT A, as follows:

```
30* REPORT A (STRING (PRINT C1,C2,C3,C4,C10,C25,C27,F30 WHERE
PORTFOLIO NAME EQ INCOME:)):
```

Later, when using the string in the RETRIEVAL module, the code word would be used between system separators, as in the following:

REPORT A

which would be interpreted by the system the same as submitting the long retrieval statement.

3.4 DATA BASE CONSTRUCTION

This section will identify those procedures for the actual construction of the data base. We will draw heavily upon the discussions already presented concerning the structure and components of the data base.

3.4.1 Basic DEFINE Procedures

After the user has worked out on paper an optimum data base definition, he is ready to load his definition into his data base using the DEFINE module. This activity will most often be done on a high speed terminal with a card reader, so that commands and component statements will be on cards. As a general rule, all SYSTEM 2000 commands and statements are concluded by a colon (:). The system is able to interpret this as an end of statement. Therefore, it is possible to put one or more commands or statements on one card.

The DEFINE module has a complete set of user diagnostics both for error checking and progress notification. It usually takes several job submissions before the user arrives at a finalized definition that is error free. There are two main types of errors, neither of which are serious, which usually require multiple job submissions: structural errors and card errors. Structural errors are of several types such as duplicate component names and/or numbers, inappropriate repeating group indicators, character length of data base name, etc. Card errors include keypunch errors, incorrect card deck arrangement, etc. With the complete SYSTEM 2000 set of diagnostics, the user is told exactly what his problem is and in most cases it is cleared up by a simple card change. During these re-submitted jobs, the user can make minor or major revisions to his definition with little effort. Once the user has developed an error free definition, he is ready to turn his attention to loading his data by use of the LOADER module after using the MAP: command.

3.4.1.1 Password and Module Command

The first two required SYSTEM 2000 commands are system-wide commands. They are used to first gain access to the system and secondly, to call the DEFINE module. For example purposes, we will assign the user password as ABCDE.

```
USER,ABCDE:
DEFINE:
```

3.4.1.2 Declare Data Base Name

The third command is the one which names the user's data base, NEW DATA BASE <data base name>:. Any data base name is limited to 20 characters or less including blanks. In the example definition we will be building, we are going to name the new data base, PORTFOLIO, so the first three commands would be:

```
USER, ABCDE:
DEFINE:
NEW DATA BASE PORTFOLIO:
```

The results of these commands, after the data base creation process is complete, is to establish ABCDE as the authorized, and only, password for the PORTFOLIO data base. If, however, the new data base name duplicates any other existing data base name, the error message returned would be - DATA BASE NAME ALREADY USED - <data base name> and the remainder of the DEFINE job would be aborted. (This is an example of a FATAL type error.)

3.4.1.3 Component Declarations

As we learned in section 3.3, there are four types of components. Any definition would contain the first two discussed, elements and repeating groups, but user-defined functions and strings are both used as required and definitions may or may not contain them. As we will learn, components may be added to an existing definition, so functions, and, in particular, strings are frequently added to a definition after a specific need is identified. The maximum number of components which may be declared in a data base is 127.

The PORTFOLIO data base definition, which is given in Figure 2, is the example data base definition which will be used throughout most of this document. Two additional data base definitions are given in Figures 3 and 4 as comparative samples of different types of data bases. All three are used throughout this document, with the later two being used in the RETRIEVAL module chapter. As previously indicated, each of these commands and/or component declarations would probably be submitted on a separate card. The previously presented commands are again listed for the sake of continuity.

3.4.1.4 Mapping the Definition

The MAP: command need not and should not be used until an error free definition is achieved. An error free definition is achieved by submitting a job deck as just defined through the DEFINE module. This module will provide all of the required tests of acceptability and issue appropriate error diagnostics. When no more errors are detected by the module, the error free definition is displayed at the conclusion of the job. The user is now ready to load data into the data base using the LOADER module.

The MAP: command is now entered, following the component declarations, and before the LOADER module is called; which results in the following:

(1) The internal SYSTEM 2000 definition tables are created.
(2) The data base name is set.
(3) The definition version number is set to one (this is a sequential number indicating the number of times the definition has been modified),
(4) The data version number is set to Ø (because no data has been loaded at this time), and
(5) The system considers the data base to exist for the life of the job.

At this point in time, the user has accomplished all of the production activities required of the DEFINE module and is now ready to call the LOADER module to load data into the newly defined data base. The DEFINE module has set up the data base

```
 USER,ABCDE:
 DEFINE:
 NEW DATA BASE PORTFOLIO:


 1* ORGANIZATION (NAME):
 2* COGNIZANT OFFICIAL (NAME):
 3* ADDRESS (NAME):
 4* CITY (NAME):
 5* STATE (NAME):
 6* ZIP CODE (INTEGER NUMBER):
 7* CURRENT DATE (DATE):
 8* PORTFOLIOS (RG):
    9* PORTFOLIO NAME (NAME IN 8):
   10* PORTFOLIO CODE (NAME IN 8):
   11* MANAGER (NAME IN 8):
   12* STOCKS (RG IN 8):
       13* NAME OF STOCK (NAME IN 12):
       14* TICKER SYMBOL (NAME IN 12):
       15* EXCHANGE (NAME IN 12):
       16* INDUSTRY NAME (NAME IN 12):
       17* INDUSTRY CODE (INTEGER NUMBER IN 12):
       18* SHARES OUTSTANDING (INTEGER NUMBER IN 12):
       19* LATEST EARNINGS (NAME IN 12):
       20* LATEST EARNINGS DATE (DATE IN 12):
       21* ESTIMATED EARNINGS (NAME IN 12):
       22* ESTIMATED EARNINGS DATE (DATE IN 12):
       23* DIVIDEND (DECIMAL NUMBER IN 12):
       24* CURRENT PRICE (DECIMAL NUMBER IN 12):
       25* TRANSACTIONS (RG IN 12):
           26* TRANSACTION TYPE (NAME IN 25):
           27* DATE (DATE IN 25):
           28* SHARES (INTEGER NUMBER IN 25):
           29* PRICE (DECIMAL NUMBER IN 25):
30* BONDS (RG IN 8):
    31* NAME OF ISSUER (NAME IN 30):
    32* ASKED PRICE (DECIMAL NUMBER IN 30):
    33* PURCHASE PRICE (DECIMAL NUMBER IN 30):
    34* MATURITY DATE (DATE IN 30):
    35* PURCHASE DATE (DATE IN 30):
    36* FACE AMOUNT (DECIMAL NUMBER IN 30):

 MAP:
```

FIGURE 2

Sample Portfolio Data Base Definition

```
USER,ABCDE:
DEFINE:
NEW DATA BASE HOUSING AUTHORITY:


1* PROJECT NAME (NAME):
2* PROJECT NUMBER (NAME):
3* PROJECT TYPE (NAME):
4* PROJECT ADDRESS (NAME):
5* HOUSING MANAGER NAME (NAME):
6* HOUSING MANAGER PHONE (NAME):
7* ACCOUNTS (RG WITH NULLS):
     8* ACCOUNT NUMBER (INTEGER NUMBER IN 7):
     9* TENANT NAME (NAME IN 7 WITH MANY FUTURE ADDITIONS):
    10* DATE ADMITTED (DATE IN 7):
    11* FAMILY SIZE (INTEGER NUMBER IN 7):
    12* RACE (NAME IN 7):
    13* BIRTH COUNTRY (NAME IN 7):
    14* BASIS FOR SELECTION (RG IN 7):
        15* BASIS (NAME IN 14):
    16* TOTAL ASSETS (NAME IN 7):
    17* SOURCES OF INCOME (RG IN 7):
        18* SOURCES (NAME IN 17):
    19* EMPLOYMENT OF PRINCIPAL WAGE EARNER (RG IN 7):
        20* OCCUPATION (NAME IN 19):
        21* INDUSTRY (NAME IN 19):
    22* INITIAL NET INCOME FOR RENT (DECIMAL NUMBER IN 7):
    23* PREVIOUS ADDRESS (NAME IN 7 WITH 45 PERCENT PADDING):
    24* PREVIOUS BOROUGH (NAME IN 7):
    25* PREVIOUS PROJECT (NAME IN 7):
    26* PREVIOUS HOUSING OCCUPANCY (NAME IN 7):
    27* SIZE OF PREVIOUS APARTMENT (INTEGER NUMBER IN 7):
    28* GROSS MONTHLY RENT FOR APARTMENT (DECIMAL NUMBER IN 7):
    29* PRIOR PERCENTAGE OF INCOME FOR RENT (NAME IN 7):
    30* LENGTH RESIDENCE LAST APARTMENT (NAME IN 7):
    31* CURRENT DATA REPORTS (RG IN 7 WITH NULLS):
        32* BASIS FOR REPORT (NAME IN 31):
        33* REPORT DATE (DATE IN 31):
        34* APARTMENT SIZE (INTEGER NUMBER IN 31):
        35* GROSS ANTICIPATED INCOME (DECIMAL NUMBER IN 31):
        36* NET INCOME FOR RENT (DECIMAL NUMBER IN 31):
        37* MONTHLY GROSS RENT (DECIMAL NUMBER IN 31):
        38* PERCENTAGE OF INCOME FOR RENT (INTEGER NUMBER IN 31):
        39* RENT ADJUSTMENT MADE (NAME IN 31):
        40* CLASSIFICATION OF NEW RENT (NAME IN 31):
        41* ELIGIBILITY FOR CONTINUED OCCUPANCY (NAME IN 31):
        42* SIZE OF FAMILY (INTEGER NUMBER IN 31):
        43* FAMILY COMPOSITION (NAME IN 31):
        44* NUMBER OF CHILDREN UNDER 21 (INTEGER NUMBER IN 31):
        45* PERSONS CURRENTLY EMPLOYED (INTEGER NUMBER IN 31):
        46* EMPLOYMENT OF WIFE OR MOTHER (NAME IN 31):
        47* MINORS CURRENTLY EMPLOYED (INTEGER NUMBER IN 31):
```

FIGURE 3

Sample Housing Authority Definition

(continued)

```
        48* SOURCES OF CURRENT INCOME (RG IN 31):
            49* CURRENT SOURCES (NAME IN 48):
        50* AGE OF HEAD OF HOUSEHOLD ( INTEGER NUMBER IN 31):
        51* SEX OF HEAD OF HOUSEHOLD (NAME IN 31):
        52* AGE OF SPOUSE (INTEGER NUMBER IN 31):
        53* MILITARY SERVICE RECORD (RG IN 31):
            54* MILITARY RECORD (NAME IN 53):
            55* DATES OF SERVICE (NAME IN 53):
        56* SERVICE-CONNECTED DISAB+DEATH COMP (DECIMAL NUMBER IN 31):
     57* CHARGES (RG IN 7):
        58* DATE OF CHARGE (DATE IN 57):
        59* NAME OF CHARGE (NAME IN 57 WITH FEW FUTURE ADDITIONS):
        60* DESCRIPTION OF CHARGE (NAME IN 57 WITH FEW FUTURE ADDITIONS):
        61* AMOUNT OF CHARGE (DECIMAL NUMBER IN 57):
     62* PAYMENT RECORD (RG IN 7):
        63* PAYMENT DATE (DATE IN 62):
        64* PAYMENT AMOUNT (DECIMAL NUMBER IN 62):
        65* ITEMIZED STATEMENT (RG IN 62):
            66* ITEM (NAME IN 65):
            67* ITEM AMOUNT (DECIMAL NUMBER IN 62):
     68* TERMINATION REASON (NAME IN 7):
     69* DATE TERMINATED (DATE IN 7):
     70* TRANSFER REASON (NAME IN 7):
     71* DATE TRANSFERRED (DATE IN 7):
     72* BALANCE DUE (DECIMAL NUMBER IN 7):
 73* ARREARS (FUNCTION(C37+C61+C72-C64)):

PAD FOR SOME VALUE DUPLICATIONS:

MAP:
```

FIGURE 3

Sample Housing Authority Definition

(continued from previous page)

```
   USER, ABCDE:
   DEFINE:
   NEW DATA BASE HEALTH PLANNING:


   1* HEALTH PLANNING AREA (NAME):
   2* COUNTIES (RG):
      3* COUNTY (NAME IN 2):
      4* CITIES (RG IN 2):
         5* CITY (NAME IN 4):
         6* BIRTHS (RG IN 4 WITH NULLS):
            8* DATE OF BIRTH (DATE IN 6 WITH SOME FURTHER ADDITIONS):
            9* SEX OF BIRTH (NAME IN 6):
           48* HOSPITAL NAME (NAME IN 6):
           10* THIS BIRTH (NAME IN 6):
           11* ORDER OF BIRTH (NAME IN 6):
           12* RACE OF BIRTH (NAME IN 6):
           14* FATHERS RACE (NAME IN 6):
           15* FATHERS AGE (INTEGER NUMBER IN 6):
           16* FATHERS BIRTHPLACE (NAME IN 6):
           18* MOTHERS RACE (NAME IN 6):
           19* MOTHERS AGE (INTEGER NUMBER IN 6):
           20* MOTHERS BIRTHPLACE (NAME IN 6):
           21* OTHER LIVING CHILDREN (INTEGER NUMBER IN 6):
           22* CHILDREN NOW DEAD (INTEGER NUMBER IN 6):
           23* CHILDREN BORN DEAD (INTEGER NUMBER IN 6):
           24* ATTENDANT AT BIRTH (NAME IN 6):
           25* LEGITIMATE (NAME IN 6):
           26* LENGTH OF PREGNANCY (INTEGER NUMBER IN 6):
           27* WEIGHT AT BIRTH (DECIMAL NUMBER IN 6):
           28* CONGENITAL OR OTHER ABNORMALITY (NAME IN 6):
           29* MONTH OF PRENATAL CARE (INTEGER NUMBER IN 6):
        30* DEATHS (RG IN 4 WITH NULLS):
           32* DATE OF DEATH (DATE IN 30 WITH SOME FUTURE ADDITIONS):
           33* SEX OF DECEASED (NAME IN 30):
           34* RACE OF DECEASED (NAME IN 30):
           35* MARITAL STATUS (NAME IN 30):
           37* AGE OF DECEASED (INTEGER NUMBER IN 30):
           39* BIRTHPLACE (NAME IN 30):
           40* CITIZEN (NAME IN 30):
           45* PRIME CAUSE OF DEATH (NAME IN 30):
           46* AUTOPSY PERFORMED (NAME IN 30):
           47* UNNATURAL DEATH (NAME IN 30):
205* VITAL STAT REPORT (STRING(PRINT COUNT BIRTHS, COUNT DEATHS WHERE HEALTH
                        PLANNING AREA EQ)):

MAP:
```

FIGURE 4

<u>Sample Health Planning Definition</u>

structure as defined by the user.

This need not be the final use of the DEFINE module. As we will see, definitions can be later modified. Also, if the user would like to see his definition, he can call the RETRIEVAL module, following a mapped definition, and give the DESCRIBE command (discussed in the RETRIEVAL module chapter).

3.4.2 Padding Options

The user has the capability within SYSTEM 2000 while defining his data base of preparing for future data storage characteristics which might affect his retrieval times. Padding options allow the user to reserve space in the internal tables for consecutive blocks of pointers for all data values.

If the user is going to have a data base containing less than 7000 logical entries, he need not concern himself with any of the considerations discussed within this section. Padding options should only be used when dealing with very large data bases and large, fairly frequent data modifications or updates are anticipated.

If these options have not been exercised, and massive updates have taken place, creating excessively slow response time during retrievals, the user has the ability to improve his response time by exercising his previous options and performing a simple RELOAD operation (see RETRIEVAL module). These operations would compact existing data and reserve space for future update operations.

Padding options can be divided into three main types, each of which are discussed in the following sections.

3.4.2.1 Null Options - Repeating Groups

As we have seen, repeating groups are usually made up of elements which contain values. When repeating groups are defined with a null option, it causes the LOADER module to allow a complete and consecutive block of pointers for all data values in the associated repeating group, whether or not data values were entered for all or only a few of the elements. Later updates would store these values in their reserved location, associated with their appropriate data set. If the user would ignore this option, data values entered at different update sessions would be chained together over the table partitions, widely scattered and the overall effect will probably cause an increase in retrieval time when the user employs the RETRIEVAL module. This can be averted by assigning the null option to those repeating groups which might be most affected. Null options only apply to repeating groups (remember that level zero elements are considered to be members of the level zero repeating group).

Null options may be declared during the initial definition phase or sometime later. Those actions which create the null options are as follows:

(1) Level Zero Repeating Group

Before the MAP: command, insert the following:

```
ADD NULL OPTION TO LEVEL ZERO ELEMENTS:
```

To remove this option at some later time, the next command would be:

```
REMOVE NULL OPTION FROM LEVEL ZERO ELEMENTS:
```

(2) All Non-Zero Level Repeating Groups

Within the repeating group declaration the component type is contained in a parenthetical expression to the right of the component name. The null option is given within this expression during the initial definition phase, as follows:

```
 8* PORTFOLIOS (RG WITH NULLS):
```

or

```
12* STOCKS (RG IN 8 WITH NULLS):
```

Later, if the user would like to add or remove a null option to a repeating group, he can issue, for example, the following commands:

```
ADD NULL OPTION TO REPEATING GROUP <component number>:
REMOVE NULL OPTION FROM REPEATING GROUP <component number>:
```

3.4.2.2 Padding for Unique Element Values

Each element in the data base has a set of unique values. This option is a valuable one where an element is expected to have many unique values. Typically, these unique values will be entered over a period of time during many different loading and updating sessions. The option may be quite useful because it permits the user to insert unique values consecutively in their appropriate storage tables regardless of when they are inserted. The extra space or padding leaves the necessary space. There are two general ways in which this type of padding can be assigned to an element.

(1) Assigned In the Element Declaration

When the element is initially defined in the definition, the padding option may be specified within the element type specification in one of two ways:

```
     (a) . . . WITH <nn> PERCENT PADDING
or   (b) . . . WITH <amount> FUTURE ADDITIONS
```

where:

```
     <nn> = integer number from 0 to 60
     <amount> = [NO  ]                NO = 0%
                [FEW ]              FEW = 15%
                [SOME]    where:   SOME = 30%
                [MANY]             MANY = 50%
```

A few examples using the foregoing options in the PORTFOLIO data base definition would be as follows:

```
 7* CURRENT DATE (DATE WITH 60 PERCENT PADDING):
13* NAME OF STOCK (NAME IN 12 WITH MANY FUTURE ADDITIONS):
24* CURRENT PRICE (DECIMAL NUMBER IN 12 WITH 40 PERCENT PADDING):
```

Where the user assigns a padding option amount to an element the system counts the number of unique values stored via the initial load and then reserves that percentage of additional space in the appropriate tables for the to-be-added values.

(2) Assigned After the Element Declarations

This same padding option may be specified by another set of commands after an element has been declared:

```
     (a) PAD [component number][nn] PERCENT:
or   (b) PAD [component number] FOR [NO  ] FUTURE ADDITIONS:
                                    [FEW ]
                                    [SOME]
                                    [MANY]
```

The values for these commands are the same as the ones just given. These commands stand alone and can be used during the initial definition declaration before the MAP: command or can be used at some future time, as will be shown in a later section with the REMAP: command discussion.

3.4.2.3 Padding for Multiple Occurrences of Each Unique Value

The third and last padding option allows the user to pad for multiple occurrences of each unique element value, across the entire data base for all elements. Two alternate forms are available:

```
     (a) PAD VALUE DUPLICATIONS <nn> PERCENT:
or   (b) PAD FOR ┌NO  ┐ VALUE DUPLICATIONS:
                 │FEW │
                 │SOME│
                 └MANY┘

     where:  <nn> = integer number from 0 to 60

             NO = 0%
             FEW = 15%
             SOME = 30%
             MANY = 50%
```

These last padding options may be taken during the initial definition, or they can be used at some later time, as will be shown in a later section with the REMAP: command discussion.

3.5 DATA BASE MODIFICATION

Data base definitions may be modified at will prior to loading the data. It is at these early stages of definition development that most of the bugs should be worked out. But, after data has been loaded into the data base, it is still possible to modify the definition in several different ways, adding another powerful problem-solving tool to SYSTEM 2000.

3.5.1 Adding New Components

Any of the four component types can be added to an existing data base. It is accomplished through use of the REMAP: command. The procedure is almost identical to the initial declaration except that the new component declarations are the only ones that need to be given. The actual procedure is shown in the following steps.

(1) Call DEFINE Module

The user must issue the system-wide command

DEFINE:

(2) Declare Old Data Base Name

Once a data base has been given a name, a definition version number, and a data version number and has been mapped into a definition, it becomes an "old" data base. If the user wants to refer to his "old" data base by name while in the DEFINE module, he must issue the following command:

OLD DATA BASE <data base name>:

which, in our continued example, would be;

OLD DATA BASE PORTFOLIO:

In response, the DEFINE module makes that definition available for modification.

(3) Component Declarations

The desired component declarations are now specified. All components added to a data base through these procedures are added physically to the end of the definition; although logically they exist within the repeating group (a level 0) of the definition. Any combination of components may be added so long as the structural convention rules are followed.

(4) REMAP:

When the initial definition tables were constructed, the definition was MAPPED. Now, when accomplishing a definition modification, the command is REMAP: . This action will now cause the DEFINE module to modify the existing definition residing on disk, e.g., increment the definition version number by one and incorporate the specified changes and additions into the definition tables. In summary, the commands required to add new components are as follows:

```
DEFINE:
OLD DATA BASE PORTFOLIO:
Component Declarations:
REMAP:
```

3.5.2 Changing Padding Options

Padding options were previously discussed in section 3.4.2 as they pertained to the initial definition declaration. Padding options can be declared during the initial definition declaration or can be added at a later time in exactly the same manner as new components are added. As discussed in the previous section the certain initial commands are required. Examples are shown below with all of the available padding option commands which might be used.

```
REQUIRED     DEFINE:
             OLD DATA BASE  PORTFOLIO :

OPTIONAL     ADD NULL OPTION TO LEVEL ZERO ELEMENTS:
             REMOVE NULL OPTION FROM LEVEL ZERO ELEMENTS:

             ADD NULL OPTION TO RG 25:
             REMOVE NULL OPTION FROM RG 25:

             ADD NULL OPTION TO RG 25:
             REMOVE NULL OPTION FROM RF 25:

             PAD 13 20 PERCENT:
             PAD 13 FOR NO FUTURE ADDITIONS

             PAD VALUE DUPLICATIONS 35 PERCENT:
             PAD FOR NO VALUE DUPLICATIONS:

REQUIRED     REMAP:
```

3.5.3 Specific DEFINE Commands

There are two additional commands in the DEFINE module repertoire which have yet to be introduced. Both are used after the initial definition declaration and require the use of the OLD DATA BASE <data base name>: and the REMAP: commands.

3.5.3.1 Change Single Component Number

If the user desires to change the component number of a component within his old data base he may use the following command.

```
     CHANGE NUMBER OF <component type><integer number> TO <integer number>:
where:        (1) component type = ELEMENT
                                   REPEATING GROUP
                                   FUNCTION
                                   STRING
              (2) integer number = 1-9999
```

An example of this command would be:

```
DEFINE:
OLD DATA BASE PORTFOLIO:
CHANGE NUMBER OF ELEMENT 13 TO 75:
REMAP:
```

3.5.3.2 Renumber Components

There are four commands available to the user to change the numbering scheme of the data base components. All of them may be utilized during the initial definition declaration, followed by the MAP: command, or at some later time, using the OLD DATA BASE <   >: and REMAP: commands. One word of caution. If your definition contains strings and/or user-defined functions, the component numbers referred to within these components are not affected by any of the renumber commands, which creates internal inconsistencies which cannot be tolerated by the system. Therefore, strings and functions should be defined with component names rather than component numbers.

```
(1) RENUMBER:
(2) RENUMBER STARTING WITH <n>:
(3) RENUMBER INCREMENTING BY <k>:
(4) RENUMBER STARTING WITH <n> AND INCREMENTING BY <k>:
```

In all these commands the renumbering affects every component number in the definition, including the component numbers identifying strings and user-defined functions. The RENUMBER directive causes all components to be renumbered in their current order with the first component's number set equal to one and incrementing by one for each successive component. Commands 2,3, and 4 above are similar to the RENUMBER: command, except that the user can designate any integer number <n>, $1 \le n \le 9999$ as the beginning number for the first component number and a suitable integer increment <k>. When either the second or third form of this command are used, then <k> or <n>, (whichever is omitted), respectively, is considered to be one.

3.6 DATA BASE PASSWORD CONTROL

The accessing of a data base is controlled by use of user passwords. The system will contain in its data base directory two types of passwords. These passwords will be assigned to users. By definition, some of them will be authorized to create data bases as well as all other SYSTEM 2000 activities and the second type will only be authorized to retrieve from a data base. The original mating of a password to a data base is accomplished during the creation phase via the New Data Base command. It is this command that takes the user password utilized within the job deck and assigns it to the data base named in the creation command. That is the only password authorized to access the data base until additional passwords are assigned. Also, it is the original password which has the sole access to all modules while accessing that data base. If additional passwords, which can only use the RETRIEVAL module while accessing the data base, are desired or they are no longer desired, special commands are required.

3.6.1 Assign Password

Purpose - To assign a valid password to a data base for retrieval capability only.
Command - VALID PASSWORD IS <password>:
where: password = 5 characters

Discussion - The system checks to see if the password exists in the list of available passwords. This list must be established and maintained by the computer center's Operations Department. If it does not exist, an error diagnostic - ILLEGAL PASSWORD - will be generated. If it does exist, the password is assigned and given the retrieve only permission while using the data base. The command may be issued during the original definition activity or during a remap action.

3.6.2 Delete Password

Purpose - To make a valid password invalid for a particular data base.

Command - INVALID PASSWORD IS <password>:

where: password = 5 characters

Discussion - This command is used when undesirable passwords are authorized access to a data base. This command is related, of course, to a previously accessed data base. This action would logically be used during a remap activity. This command has the possibility of creating either of two error messages. If the password used in the command never existed, the diagnostic would be - ILLEGAL PASSWORD. If the password existed but was not valid for the accessed data base the error message created would be - PASSWORD NOT VALID FOR DATA BASE.

3.7 DEFAULT CONDITIONS IN THE DEFINE MODULE

(1) If any syntactic errors occur while using the DEFINE module:
   a) the job continues if defining a new data base
   b) the job terminates if redefining an old data base

(2) Functions are "DECIMAL" functions if not otherwise defined by the user.

(3) Illegal data base names or illegal passwords designation are always fatal to the rest of the job.

(4) The password in use when the definition is mapped (and when the LOADER module has successfully entered data into a new data base) is the only password that can automatically access the new data base.

4.0 LOADER MODULE

4.1 INTRODUCTION

Once the data base has been defined, data may be loaded into it. This is done using the LOADER module. The user simply lays out the data in a string, associating with each piece of data the element number (e.g., 1* for ORGANIZATION) to which that piece has been assigned. A typical loader string would look like the example given in Figures 5 and 6.

All of the complex coding usually associated with the somewhat tedious task of loading data into a data base is no longer necessary. If the data is already in machine-readable format, like most of the stock exchange data, a simple conversion to loader string format can be accomplished using an external proprietary program language developed by MRI called RE/FORM-I; then SYSTEM 2000 will load the data into the user's data base.

The LOADER module enters data values for new logical entries into the data base. The user calls the LOADER module initially to build a new data base; he may call LOADER to do incremental loads whenever he has additional logical entries to be added to the data base. The LOADER module does not add data values to existing logical entries.

The user may call upon the LOADER module whenever he has collected a "batch" of "new" logical entries. In other words, the user may enter source data for "this year," for example, and immediately begin to use that data. Then he may enter a "batch" of last year's data and so on until all source data he may have on file from a backlog of files has been put into the data base.

The source data need not be complete, nor need it be completely accurate or "standardized." The UPDATE module can modify or insert any piece of data individually or whole sets of data across the data base. Therefore, the user need not decide beforehand what key terms are important or if they

	Loader Data Input String for Example Definition	LOADER Analysis
Logical Entry 1	1* CITY TRUST COMPANY 2* J.B. WISER 3* 303 WEST 52ND ST. 4* NEW YORK 5* NEW YORK 6* 10022 7* 10/22/69	Data set 1 (level 0)
	8* 9* INCOME 10* A 11* J.B. PACE	Data set 2 (level 1)
	12* 13* AMERICAN CYANAMID 14* ACY 15* NYSE 16* CHEMICALS 17* 2899 18* 44509000 19* 2.00 20* 06/30/69 21* 2.05 22* 12/31/69 23* 1.25 24* 29.00	Data set 3 (level 2)
	25* 26* BUY 27* 01/14/69 28* 12000 29* 31.50	Data set 4 (level 3)
	25* 26* SELL 27* 05/15/69 28* 5000 29* 33.25	Data set 5 (level 3)
	25* 26* SELL 27* 08/15/69 28* 3500 29* 27.50	Data set 6 (level 3)
	12* 13* GENERAL MOTORS 14* GM 15* NYSE 16* MOTOR VEHICLES	Data set 7 (level 2)
	25* 26* BUY 27* 03/28/69 28* 2000 29* 81.75	Data set 8 (level 3)
	12* 13* UPJOHN	Data set 9 (level 2)
	30* 31* GAC CORP. 32* 101.25 33* 98.50 34* 01/01/98 35* 05/04/70 36* 100.	Data set 10 (level 2)
	8* 9* TRUST 10* E 11* W.D. GARDNER	Data set 11 (level 1)
	**END	
Logical Entry 2	1* GOOD LIFE INSURANCE CO. 2* L.G. OGDEN 4* SAN FRANCISCO 5* CALIFORNIA	Data set 12 (level 0)
	**END **END	

FIGURE 5

Sample Loader String for PORTFOLIO Data Base

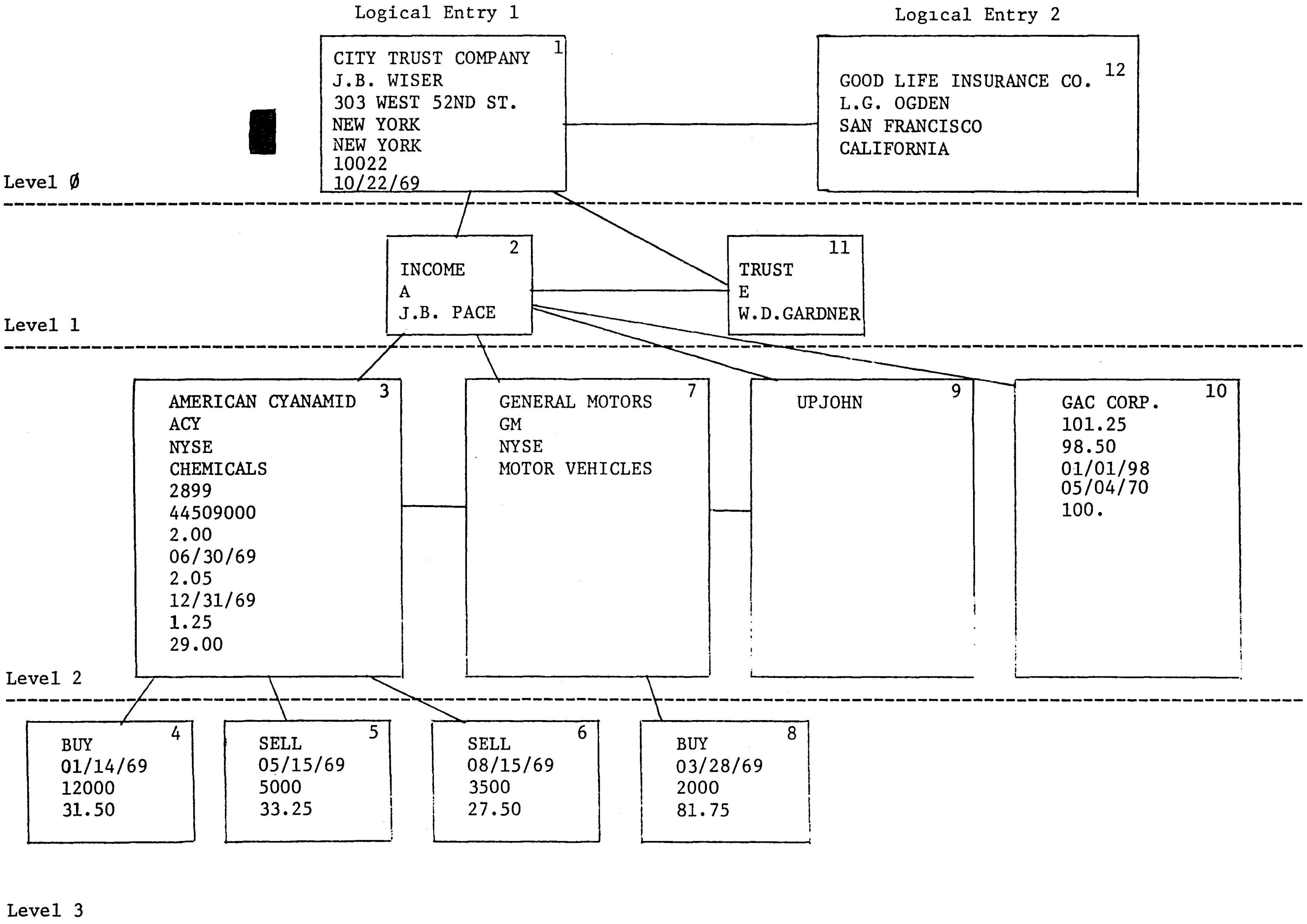


FIGURE 6

Data Base Structure for Example Portfolio Loader String are appropriate and identical across the entire source data. The RETRIEVAL module will allow him to browse through the data base and the UPDATE module will allow him to modify at will. All of the above capabilities afford the user early use of his data base whether or not his source data is complete or absolutely accurate.

The LOADER module has three general functions:

1) to enter large volumes of data into a data base.
2) to pre-edit the user's raw data by checking the data values and data structure against the user's definition of the data base.
3) to give the user statistical counts at various checkpoints throughout the loading process.

First, a <u>definition</u> must exist before the LOADER module is called. Secondly, a <u>DATA FILE</u> must be prepared which contains the data input string of values to be entered into the data base. Lastly, a set of job related commands must be given to the LOADER module to direct its services in loading the user's data.

An example set of job related commands are given below. This illustration assumes that the DEFINE module has been called prior to issuing the call to LOADER.

```
.
.
.
LOADER:
ISSUE REPORT WHEN ALL CHECKPOINTS OCCUR:
DATA FILE IS <file name>:
ASSUME NO ERRORS:
SCAN:
.
.
.
```

After the first batch of loader data values have been successfully entered into the data base, the data base automatically is available for access on disk storage. After the data base is created on the disk, it is available for permanent external storage on tape by the Save Data Base command.

4.2 LOADER DATA INPUT STRING FORMAT

The overall design of the data input string consists of groups of data values each preceded by an element number; each group (or data set) is preceded by a repeating group number. The end of a data value is signaled by the occurrence of a component number of the next value, etc., until finally an entry terminator word is encountered; at that point, data begins for the next "logical entry." The end-of-file at the end of the DATA FILE preceded by _two_ occurrences of a double system separator followed by the entry terminator is the signal to LOADER that the end of the string has occurred.

The LOADER module reads the DATA FILE for the data input string of data values. This means that before calling SYSTEM 2000, the user has gathered his source data and prepared a DATA FILE. The data may have simply been converted from another machine readable source or may have been constructed directly from raw data. The DATA FILE may be a disk file, magnetic tape or punched cards that were "copied" onto a disk file.

The data input string is one, long continuous string of characters up to an "end-of-file." The format is absolutely free field within the file, meaning that blanks may be used freely between data values and within data values. LOADER ignores all extraneous blanks and retains only one embedded blank between words if the type of data is NAME; if the data is type TEXT (see DEFINE module) blanks will be retained. Characters are read consecutively so that if a word spreads across the end of one record to the beginning of the next, then the user should consider the effect of blanks.

Loader data input string is composed from the source material at hand according to the categories of data which he has outlined in his definition of the data base. The data is first organized by logical entry. Next, the user arranges the data values within each logical entry according to the repeating groups which he has defined. There is no maximum number of data sets. Each repeating group at any level may contain as many data sets as necessary to contain the available data. In composing the data input string, the user is forming an implied "data tree" of data sets and also, an implied order of entrance into the data base. If functions or strings have been defined, they are ignored during the loading process. The values of functions are not stored in the data base; they are calculated dynamically by the RETRIEVAL module.

A data set may contain one value for each element that the user has defined in the particular repeating group under consideration. A data set, then, may contain many data values associated one-to-one with the appropriate elements in the repeating group. If the user's source data does not have a data value for an element in a particular data set, then that element should be omitted from the input string for that specific data set. Thus, a data set may contain values for all elements in the defined repeating group, for a few of the elements, or perhaps for none of the elements. If no values exist for a data set, then that data set is called a non-valued data set; generally speaking, a non-valued data set will not occur unless the user has source data for descendant data sets. The user may give level 0 values together at the beginning or end of each logical entry or he may scatter them throughout the logical entry -- as long as no family of repeating group data sets is broken. If no values exist for any of the level 0 elements, then the LOADER module automatically creates a non-valued level 0 data set for the logical entry.

A repeating group data tree begins with the topmost parent repeating group and extends vertically to the lowest level repeating group of that family tree. One or many data sets may be entered at any level depending entirely upon the available source data. Each descendant data set in every subtree must have a parent data set somewhere above it in the same subtree. If data values are to be entered at any level below a data set having no values, then a non-valued data set must be specified in the input string before entering the descendants.

SYSTEM 2000 retains the original order of entrance of values and data sets in the data base. This persistent knowledge of ordinal position of data sets is true for any data base, but may not be of any significance to the user unless he specifically wants to utilize it in retrievals or updates. The system then can follow any branch of any tree or any chain of subtrees across the entire data base without a sequential search of the data base.

4.2.1 Data Value Assignments

Each data value is preceded by its associated element number. The element number must be preceded by a blank and must be followed by the separator symbol. Therefore, the first column on the first loader string card, by definition, must be blank. The data value must conform to the type of value which the user has categorized for the element number specified, that is, if an element is a type DATE, then the value must be in DATE format or if type INTEGER NUMBER, then an integer number. Each element number must be followed by a value; if an element has no value in a particular instance, then the number for that element must be omitted from the string. Using the * as the system separator symbol, an example of a correct data value assignment is shown below.

```
     1*    CITY TRUST COMPANY
```

There may be no embedded blanks within the numerals of the element number and a blank must not occur between the element number and the separator. A blank may, but need not follow the separator. In the example, the several blanks before "CITY TRUST COMPANY" will be ignored by the LOADER module because component 1 is type NAME (see DEFINE module).

Acceptable data value assignments:

```
     1*    GOOD LIFE INSURANCE 2* J.B. WISER
    25*
    27* 03/28/69
```

Unacceptable data value assignments:

27	*03/28/69	(blanks cannot be embedded between component number and the separator)
1*	2*J.B. WISER	(no data value was given for component 1 -- if no value exists, then omit the element number)
22*	12/1/1935	(incorrect format for type date -- day field should contain 2 digits)

The user must position his DATA FILE (rewind or whatever) before calling LOADER. The "blank" preceding a component number is always associated with the number. Thus, while the input string is free field regarding use of blanks, it is especially noteworthy that if an element is type TEXT and trailing blanks are to be included for that element's data values (for instance, in graphs, pictures, or columnar material), then care should be taken to form the "next" component number beyond the exact last blank of TEXT.

4.2.2 Data Set Assignments

The level 0 elements and their associated data values always become the level 0 data set in each logical entry; therefore, the user never "assigns" the level 0 data set. Data sets below level 0, that is, for all repeating groups must be furnished by the user. Each data set is formed by giving a repeating group number followed by the data value assignments for that particular data set. Thus, any repeating group number found in the data input string signals the end of the previous data set and the beginning of the next data set. All elements assigned data values in a data set must belong to the repeating group specified. Within a data set, the user may enter the actual data values for the elements in any order. If any element does not belong to the repeating group specified, an error message will be given to the user and the data will be rejected. All values for a data set (for all elements having values in that repeating group) must be given before another repeating group number is given.

The repeating group number must be preceded by a blank, have no embedded blanks, and must be followed immediately by the separator symbol. For example,

```
8* Δ 12* Δ 25* Δ 26* BUY
```

Blanks may be used freely after a system separator; the LOADER module ignores them up to the blank preceding the first non-blank character. At Least one blank must exist after the separator; it is part of the next component number.

Acceptable data set assignments:

```
8* 9* INCOME 10* A 11* J.B. PACE
12* 13* AMERICAN CYANAMID 14* ACY 15* NYSE
```

Unacceptable data set assignments:

12* 13* UPJOHN

9* TRUST 10* E 11* W.D. GARDNER (repeating group 8* is missing for the data set TRUST, E, W.D. GARDNER)

25* BUY 27* 03/28/69 28* 2000 (a value cannot follow a repeating group number -- 26* has been omitted)

4.2.3 Special Labels and Non-Data User Messages

A special label is two adjacent separator symbols preceded by a blank, for instance, **. The special label may be used anywhere in the input string. It is used as a signal to the LOADER module that the message immediately following the special label is not to be put into the data base. One of the main functions of the special label, when used with the entry terminator, is to signal the end of each logical entry. As discussed above, the end of a value is signalled by the occurrence of another component number; the end of a data set is signalled by the occurrence of a repeating group number. The end of the data values belonging to each logical entry is signalled by a special label followed by the entry terminator word. The end of the entire string is signalled by two entry terminators and then an end-of-file mark. The standard entry terminator word by default is END; it is short and meaningful. The user may choose his own entry terminator word, but must give the system-wide command ENTRY TERMINATOR IS <entry terminator word> so that LOADER will recognize the non-standard terminator.

Another use of the special label is to allow the user to insert "comments" throughout his data input string. These comments are useful only to the user. They allow him to annotate any item he wishes. A comment here and there may be useful in locating errors in his source material or in telling him what data has been formed. All comments are displayed on the MESSAGE FILE unless the user has told the LOADER module to SUPPRESS COMMENTS. A comment is given in the following way:

```
**COMMENT This transactions data set is the 4th one ....**COMMENT
we do not have data for the portfolio on growth.
```

A comment may occur anywhere in the data string between values or data sets. Blanks may be used freely within a comment; a maximum of 255 characters, not including extraneous blanks, may be given in a single comment.

4.3 USER DIRECTIVES

The following sequence of commands is the minimal and mandatory set of commands for using the LOADER module.

```
.
.     (previous commands to other modules, perhaps)
.
LOADER:
DATA FILE IS <file name>:
SCAN
.
.
.
```

The LOADER command calls in the LOADER module. The data input string must be pre-constructed on the DATA FILE prior to calling LOADER: then, that DATA FILE's name must be furnished to LOADER. The SCAN directive is local to the LOADER module; it tells LOADER to scan the data input string and build the data base. No other commands are necessary. If the user thinks there may be errors in his data input string, he can ask LOADER to "pre-edit" his data and STOP AFTER SCAN.

Several local directives are available to let the user tell LOADER what options to perform during that particular loading process. Local directives must be given after calling LOADER and before telling LOADER to SCAN. These local commands are discussed in detail below. Depending upon the nature of the job, the user can combine one or more of these directives to produce the desired results.

4.3.1 NOTIFY Directive

The NOTIFY directive is as follows:

```
                                                      ┌            ┐
                                                      │ENTIRE ENTRY│
NOTIFY MESSAGE FILE IF ANY ERROR(S) OCCUR(S), DISPLAY │LEVEL 0 ONLY│ :
                                                      │ERRORS ONLY │
                                                      └            ┘
```

The NOTIFY directive concerns the display of error messages. If the data string contains no errors, the NOTIFY command does nothing. The first error in a logical entry causes all data values after the error to be excluded from the data base. The next logical entry, however, is treated as though it were error free, and so on. The LOADER module tries to enter as much acceptable data as it can into the data base unless the user has told the LOADER module to STOP .... The NOTIFY directive allows the user to designate what he wants displayed in case of errors. If he chooses the ENTIRE ENTRY option, all accepted data values are displayed before the excluded values. This might produce a lengthy list unnecessarily; if the user can locate the source of error in his input by knowing a few unique values for the level 0 elements only, he can choose the LEVEL 0 ONLY option. If errors are very easily isolated in the source data, he may want to obtain a display of ERRORS ONLY. The default option if the NOTIFY command is not given, is a display of ERRORS ONLY.

4.3.2 ISSUE REPORT Directive

The ISSUE REPORT directive is as follows:

```
                  ┌                                     ┐
                  │ALL LEGALITY CHECKED                 │
                  │VALUES FOR EACH ELEMENT ARE ENTERED  │
ISSUE REPORT WHEN │SELECTION TABLES ARE COMPLETE        │ :
                  │FINAL SORT IS COMPLETE               │
                  │LOADING IS COMPLETE                  │
                  │ALL CHECKPOINTS OCCUR                │
                  └                                     ┘
```

The user may specify any combination of reports to be issued by giving one or more of the alternatives above; each alternative must be separated from the next by a comma or the word "and." All checkpoint reports are issued if the last alternative is given. If the ISSUE REPORT command is not given to the LOADER module, then by default no reports are given to the user.

ALL LEGALITY CHECKED - means that LOADER has scanned the entire data input string (or was told to STOP) and the user wishes a report of the number of entries accepted, number of accepted data values, number of rejected data values, etc.

VALUES FOR EACH ELEMENT ARE ENTERED - means that while LOADER is building the internal Selection Tables, those containing the data values, the user wishes to have a report of the number of unique values being entered for each element.

SELECTION TABLES ARE COMPLETE - means that a report will be given after all data values have been successfully entered into the data base.

FINAL SORT IS COMPLETE - means that a checkpoint message will be given as the LOADER module is beginning to build the Retrieval Tables, those that contain data set and data tree associations.

LOADING IS COMPLETE - means that a final summary report containing statistics and counts of values, etc., will be displayed.

ALL CHECKPOINTS OCCUR - means that all of the above checkpoint reports will be given to the user.

An example of each report is presented below.

```
16.39.10
10/28/69

        ALL LEGALITY IS CHECKED FOR THIS LOADER CALL
              NUMBER OF ACCEPTABLE ENTRIES =   36
              NUMBER OF VALUES ACCEPTED =   1562
              NUMBER OF ENTRIES WITH REJECTED VALUES =   0
              NUMBER OF VALUES REJECTED =   0
              NUMBER OF VALUES EXCLUDED =   0
              NUMBER OF DATA SETS ACCEPTED =   343
              NUMBER OF NON-VALUED DATA SETS ACCEPTED =   30

16.39.40.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C  1 =    4
16.39.40.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C  2 =    4
16.39.41.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C  3 =    2
16.39.42.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C  5 =    4
16.39.43.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C  6 =    4
16.39.43.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C  9 =   36
16.39.44.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C 10 =   31
    .             .                            .          .
    .             .                            .          .
    .             .                            .          .
16.40.07.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C 26 =   34
16.40.08.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C 27 =   30
16.40.08.     TOTAL NUMBER OF UNIQUE VALUES FOR ELEMENT C 28 =   27

     16.40.10.

              SELECTION TABLES ARE COMPLETE

     16.40.39.

              FINAL SORT IS COMPLETE

     16.40.48.

              LOADING COMPLETED
                 NUMBER OF ACCEPTABLE ENTRIES =   36
                 NUMBER OF VALUES ACCEPTED =   1562
                 NUMBER OF ENTRIES WITH REJECTED VALUES =   0
                 NUMBER OF VALUES REJECTED =   0
                 NUMBER OF VALUES EXCLUDED =   0
                 NUMBER OF DATA SETS ACCEPTED =   343
                 NUMBER OF NON-VALUED DATA SETS ACCEPTED =   30
                 NUMBER OF NULLS CREATED IN THIS JOB =   0
                 TOTAL NUMBER OF UNIQUE VALUES IN DATA BASE =   691
                 TOTAL SIZE OF CURRENT DATA BASE =   68 PARTITIONS OR
                    174080 CHARACTERS

     PORTFOLIO       DEFINITION VERSION       1      DATA VERSION       1

          16.40.48.      10/28/69
```

4.3.3 STOP AFTER Directive

The STOP AFTER directive is as follows:

```
                    ┌                  ┐
                    │ ONE ERROR        │
                    │ 1 ERROR          │
        STOP AFTER  │ 0 ERROR          │:
                    │ <integer> ERRORS │
                    │ SCAN             │
                    └                  ┘
```

The user may tell LOADER to stop after the scan no matter how many errors occurred or did not occur. This option is used effectively for a "pre-edit" of a new input string. The ASSUME ERRORS directive should also be given for a pre-edit. The user can then correct the errors and resubmit the input string for another pre-edit, etc., until all errors disappear.

The other alternatives indicate that the LOADER module is to scan the input string until the specified number of errors occur at which point LOADER is to stop the scan. If Ø ERRORS is specified, it means STOP AFTER SCAN. If LOADER is not told to stop after the scan, all acceptable values will automatically be entered into the data base whether or not errors occurred. ERRORS ARE COSTLY -- IN DISPLAY TIME, IN LATER UPDATE SESSIONS TO CORRECT A DATA BASE AND IN ACTUAL COMPUTER TIME.

The user should always pre-edit all data input strings and correct all errors by telling the LOADER module to STOP AFTER SCAN before letting LOADER build a data base with ASSUME NO ERRORS.

4.3.4 SUPPRESS COMMENTS Directive

The SUPPRESS COMMENTS directive simply tells the LOADER module that if it finds any user comments within the data input string, the user does not want them displayed. If the directive is not given, comments are displayed for the user. He will usually want comments displayed during a pre-edit run, but in general they should be suppressed when he wants LOADER to go ahead and build the data base.

4.3.5 ASSUME Directive

The ASSUME directive is as follows:

$$\text{ASSUME} \begin{bmatrix}\text{NO ERRORS} \\ \text{ERRORS}\end{bmatrix}:$$

The ASSUME directive allows the user to let the LOADER module know whether the data input string should be checked for errors or not. Scanning for all legality and isolating errors, giving error messages, etc., takes more time than knowing beforehand that the string is error free. After the user has assumed errors, corrected any errors, then he may issue an ASSUME NO ERRORS directive. This will be timesaving in that LOADER assumes all error checking has been done on a previous run. Most of the error checking will not be performed when assuming no errors and the user runs the risk of entering bad data into the data base if the user has not made a "pre-edit" run previously. Some structural error checking is performed at all times and if an error occurs, the LOADER module will issue a message to the user that it encountered an error while "assuming no errors," and then halt. The default for assume directives is to ASSUME ERRORS always unless otherwise directed. The ASSUME directive and the STOP AFTER directives may be issued effectively in combination.

If the user ASSUMEs NO ERRORS and undetected errors occur within the loader string, the result will be indeterminate.

4.3.6 SCAN Directive

The SCAN command causes the LOADER module to start scanning the data input string. If no STOP command was given, the data base will be built. No more commands can be given to LOADER after the SCAN command.

4.4 OUTPUT FROM THE LOADER MODULE

The only output from the LOADER module (aside from error messages) is the data base itself, newly constructed or reconstructed, with all data internally ready for retrievals, updates, or report generation. A secondary and optional output at the user's choice is a display of checkpoint reports produced during the loading operation.

The user may call any task module immediately after using LOADER. If an archival tape of the data base is desired, a Save Data Base command should be given.

4.5 ERROR HANDLING

The LOADER module always tries to scan the entire data input string and will always enter all acceptable data values into the data base unless the user has told LOADER to stop after the scan or after a certain number of errors have occurred. If errors occur in the user's data, they will always be detected during the scanning process -- not while the data base is being constructed. Therefore, all errors will be known to the LOADER module before it begins to enter the data into the data base. If the user has specified that he wishes to see the first checkpoint report and that LOADER should stop after the scan, then all errors are displayed according to the NOTIFY directive (or default if none was issued) that the user chose and a summary of the number of rejected and accepted values, etc., will be displayed.

The data values along with their corresponding element numbers are displayed with error messages if any occur. The display of errors may include three types of displays:

1. accepted data values within a logical entry
2. rejected data values
3. excluded data values

Accepted data values are those values whose type and structure are legal and acceptable for the data base. Rejected data values are those values that are erroneous either because of legality tests, wrong type of data, incorrect repeating group membership, data set structure, or any one of the many errors that may possibly occur in the data input string. Excluded data values are those data values that occur within the logical entry containing rejected data values after the point of the first error in the logical entry. Excluded data values are those values which by themselves are "acceptable" (that is, all error checking has been performed upon them), but they may be dependent upon rejected data values and therefore are excluded from the data base until all errors have been corrected.

The LOADER module always tries to read to the end of the DATA FILE. Each new logical entry is treated as though no errors exist. Acceptable data values are not displayed for logical entries having no errors. Thus, if not told to stop after the scan, the LOADER module could, for instance reject 50 values because of illegal data, exclude 1500 data values because they occurred after errors in logical entries having rejected data, and then still go ahead building the data base with 50,000 accepted data values. The user should pre-edit until all data is error free or costly time will be spent using the UPDATE module in correcting, inserting and adding the corrected values to the existing logical entries.

An example of a display from the LOADER module illustrating the occurrence of errors, and illustrating the "DISPLAY ENTIRE ENTRY" option is given below.

```
    -ACC-     1* CITY TRUST COMPANY
    -ACC-     2* J.B. WISER
    -ACC-     4* NEW YORK

         3 ACCEPTED DATA VALUES FOR LOGICAL ENTRY 2

    -REJ-     9* INCOME

              NO PRECEDING PARENT DATA SET        REJECTIONS = 1

      ⋮              ⋮                ⋮

   -REJ-      27* 12/5/66

              ILLEGAL DATE DATE VALUE             REJECTIONS = 2

   -EXC-      27* 05/01/66
   -EXC-      16* BUY
   -REJ-      25* SELL

              VALUE GIVEN AFTER A DATA SET LABEL  REJECTIONS = 3
         TOTAL REJECTED DATA VALUES FOR LOGICAL ENTRY 2 = 3
```

NOTE: Logical entry 1 had no errors, evidently, so it was not displayed.

4.6 DEFAULT CONDITIONS IN THE LOADER MODULE

Minimally the following commands must be given to the LOADER module:

```
LOADER:
DATA FILE IS <file name>:
SCAN:
```

If no other commands are given, then by default:

1. All user comments are displayed.
2. A display of all rejected and excluded data values (if any error occurred) is sent to the MESSAGE FILE. No accepted data values are displayed. The "ERRORS ONLY" option is employed.
3. No accepted data values are ever displayed for any logical entry which was error free. The user may use the RETRIEVAL module to print any or all logical entries in the data base.
4. The LOADER module ASSUMES ERRORS and performs all legality and data set structure testing unless directed otherwise. Error checking takes more time and could be avoided if the data is error free.
5. If <u>any</u> acceptable data values exist after the scanning process, the LOADER module <u>will enter them into the data base</u>. No matter how many hundreds of errors occur or reoccur in the string, fragments of all accepted values are entered into the data base unless the user issues a STOP AFTER SCAN directive. This could be costly and dangerous if performing an incremental load.
6. LOADER stops scanning the loader input string when it encounters two consecutive entry terminators. No data will be read beyond that point whether or not an end-of-file is detected.
7. All padding and null options as given in the definition for the current data base are used in loading.
8. Any error in a LOADER command terminates the job.

5.0 RETRIEVAL MODULE

5.1 INTRODUCTION

The DEFINE and LOADER modules discussed in the two previous sections are used to create a data base; the RETRIEVAL and UPDATE modules are used to interact with the data base.

The UPDATE module, discussed in a following section, loads the most recent data values into the data structure and keeps the data current. The RETRIEVAL module, discussed below, consists of a number of access tools to get information from data bases.

The data bases used in the examples are those from the DEFINE module section 3.4.1.3.

5.2 RETRIEVAL COMMANDS

RETRIEVAL commands can take on many forms. The commands normally take on the format of PRINT information:, PRINT information WHERE certain conditions exist:, or DESCRIBE something:. The commands can be quite simple or as complex as the user requires. The only SYSTEM 2000 commands required prior to one or more RETRIEVAL commands are the User and Data Base Name commands. The next several RETRIEVAL commands assume the data base accessed is the HEALTH PLANNING data base, presented in section 3.4.1.3.

5.2.1 PRINT Request

Command

```
PRINT MARITAL STATUS:
```

Printout

```
PRINT MARITAL STATUS:

     35* DIVORCED
     35* MARRIED
     35* NEVER MARRIED
     35* WIDOWED
```

Explanation This is a request for all the unique values -- not multiple occurrences -- for a single component, which in this case is Component 35. Even though every death recorded in the Health Planning Data Base has a value for this component, only the unique values are printed. It will be shown later that when a WHERE clause is used, all of the values that qualify are printed out, not just the unique ones.

In the resulting printout, the command is always given with the printout, so that the question and answer can be clearly associated. The number 35 indicates the component number. The adjacent asterisk is the system separator or the changeable character which separates the component numbers from their values.

5.2.2 PRINT Request With Two Components

Command

```
PRINT HEALTH PLANNING AREA, COUNTY:
```

Printout

```
PRINT HEALTH PLANNING AREA, COUNTY:

  1* ALAMO PLANNING REGION
  1* CENTRAL TEXAS PLANNING REGION
     3* ATASCOSA
     3* BELL
     3* BEXAR
     3* BOSQUE
     3* COMAL
          .
          .
          .
     3* ZAVALA
```

Explanation This example is quite similar to the first, except that it requests the unique values of two components; the two are set off by commas. The two components, numbers one and three respectively, are not associated with each other, but are listed separately. The output for Component 3, COUNTY, is indented from the output for Component 1 because of the organization of this data base definition: Counties exist within Health Planning

Areas, which is referred to in SYSTEM 2000 as an indication of the hierarchical structure.

5.2.3 PRINT Request With Format Instructions

Command

```
PRINT BLOCK, STUB SUPPRESS, DOUBLE SPACE, HEALTH PLANNING
AREA, COUNTY:
```

Printout

```
PRINT BLOCK, STUB SUPPRESS, DOUBLE SPACE, HEALTH PLANNING

AREA, COUNTY:

  ALAMO PLANNING REGION

  CENTRAL TEXAS PLANNING REGION

  ATASCOSA

  BELL

      .

      .

      .

  ZAVALA
```

Explanation There are three types of format instructions available to SYSTEM 2000 users. The first, as used above, is the control of the left hand margin by the command of INDENT or BLOCK. The second format instruction determines whether the stub - the component number and system separator to the left of the component values - appears or is suppressed. The two commands are STUB or STUB SUPPRESS. The last format instruction is the control of the spacing by the command of SINGLE SPACE or DOUBLE SPACE. If no format instructions are given, the conditions which prevail by default are: INDENT, STUB and SINGLE SPACE.

5.2.4 PRINT Request Using Component Numbers

Command

```
PRINT STUB, SINGLE SPACE, INDENT, C1, C3:
```

Printout

```
PRINT STUB, SINGLE SPACE, INDENT, C1, C3:

  1* ALAMO PLANNING REGION
  1* CENTRAL TEXAS PLANNING REGION
     3* ATASCOSA
     3* BELL
          .
          .
          .
     3* ZAVALA
```

Explanation Component numbers, e.g., C1, C2, etc., can be substituted for component names in any SYSTEM 2000 command. This is a convenient shorthand for the operator after he gains familiarity with the data base definition. The User-Defined Functions and Strings, discussed later, can be referred to by their component numbers, e.g., F1, F2, or S1, S2, etc.

5.2.5 Accessing Different Data Bases

Command

```
DATA BASE NAME IS HOUSING AUTHORITY:
```

Printout

```
DATA BASE NAME IS HOUSING AUTHORITY:
```

Explanation The standard data base can be changed at any time by use of the command DATA BASE NAME IS <data base name>: . Operationally, the user usually issues one or more commands following this one, as in the next example.

5.2.6 DESCRIBE

Command

```
DATA BASE NAME IS HOUSING AUTHORITY:
DESCRIBE:
```

Printout

```
DATA BASE NAME IS HOUSING AUTHORITY:

DESCRIBE:
SYSTEM 2000, VERSION 19
DATA BASE NAME IS HOUSING AUTHORITY:

DEFINITION NUMBER:      1
DATA BASE VERSION NUMBER:      1

   1* PROJECT NAME (NAME)
   2* PROJECT NUMBER (NAME)
                 .
                 .
                 .
```

```
7* ACCOUNTS (RG)
   8* ACCOUNT NUMBER (INTEGER NUMBER IN 7)
   9* TENANT NAME (NAME IN 7)
                .
                .
                .
  72* BALANCE DUE (DECIMAL NUMBER IN 7)
```

Explanation   The command changing the working data base is repeated here along with the DESCRIBE command. It is the DESCRIBE command which produces a printout of the entire data base definition. The present operating version of SYSTEM 2000 and the name of the data base are given first. The current Definition and Data Base Version Numbers are then output with this command. If the definition or the data base is modified or changed, the Definition and Data Base Version Numbers are incremented appropriately. The DESCRIBE command has several forms. The one used above produces a full printout of all elements and repeating groups (not Functions and Strings which are discussed later). The other forms of the DESCRIBE command that relate to components are as follows:

```
DESCRIBE C#:
DESCRIBE C# THROUGH C#:
DESCRIBE C# THRU C#:
DESCRIBE C# TO END:
```

5.2.7 PRINT ENTRY

Command   PRINT ENTRY:

Printout   PRINT ENTRY:

(Entire Data Base would be printed here)

Explanation   This is the command which prints the entire data base. All SYSTEM 2000 data bases are organized around logical entries of data. The Housing Authority Data Base uses a housing project as a logical entry and each of the housing projects contains accounts or tenants. The command, PRINT ENTRY, asks for the printout of each logical entry, and therefore is rarely used. The use of the term ENTRY is more normally used with a WHERE clause, which will usually limit the entries qualifying, such as:

```
PRINT ENTRY WHERE PROJECT NAME EQ ADAMS HOUSES:
```

The WHERE clause will be introduced later, but this example shows that the only entry to be output is the ADAMS HOUSES entry.

5.2.8 The WHERE Clause

The WHERE clause is used to subject retrievals to specific criteria. The PRINT portion of the command asks for all information qualified by the criteria listed in the WHERE clause. The WHERE clause conditions are specified using the general format shown below:

```
PRINT information WHERE certain conditions exist:
```

5.2.9 Relational Operators

Command

```
PRINT TENANT NAME, DATE ADMITTED WHERE TENANT NAME EQ BAFFORD J E:
```

Printout

```
PRINT TENANT NAME, DATE ADMITTED WHERE TENANT NAME EQ BAFFORD J E:

 9* BAFFORD J E
10* 06/01/60
```

Explanation

The SYSTEM 2000 relational operators are:

EQ	EQUAL
NE	NOT EQUAL
LT	LESS THAN
LE	LESS THAN OR EQUAL TO
GT	GREATER THAN
GE	GREATER THAN OR EQUAL TO

They are used within the condition statement of the WHERE clause and are formulated as follows:

```
                          ┌  ┐
                          │EQ│
                          │NE│
... WHERE <element name>  │LT│  <specific value>:
                          │LE│
                          │GT│
                          │GE│
                          └  ┘
```

5.2.10 SPAN, SPANS or SPANNING

Command

```
PRINT TENANT NAME, NET INCOME FOR RENT WHERE NET INCOME
FOR RENT SPANS 8900., 10000.:
```

<u>Printout</u>

```
PRINT TENANT NAME, NET INCOME FOR RENT WHERE NET INCOME

FOR RENT SPANS 8900., 10000.:

     9* ALCANTARA FIDEL
        36* 8971.
     9* BETER D M
        36* 9600.
```

<u>Explanation</u> The SPANS command can be considered one of the relational operators. The command allows spanning of any data inclusive of the values given. This example also shows the related pieces of data presented together, which is the result of the WHERE clause usage. Recall that a simple PRINT command did not result in any association of values. Any data value can be used in the SPAN command. The common format for the SPANS command is as follows:

```
                           ┌         ┐
                           │SPAN     │
... WHERE <element name>   │SPANS    │ <A>,<B>:
                           │SPANNING │
                           └         ┘
```

where A $\leq$ B.

5.2.11 EXISTS, FAILS

<u>Command</u> PRINT TENANT NAME, SERVICE-CONNECTED DISAB+DEATH COMP WHERE SERVICE-CONNECTED DISAB+DEATH COMP EXISTS:

<u>Printout</u>

```
PRINT TENANT NAME, SERVICE-CONNECTED DISAB+DEATH COMP WHERE

SERVICE-CONNECTED DISAB+DEATH COMP EXISTS:

9* ACKERBAUM EDW I
   56* 0252.
9* CARRUTHERS BEN F
   56* 0598.
9* CARTAGENA JOSE
   56* 0612.
9* ADAMOWITZ ELENA
   56* 0612.
```

<u>Explanation</u> The EXISTS and FAILS commands are used in the WHERE clause with the following format:

```
WHERE <element name> ┌EXIST, EXISTS, or EXISTING┐ :
                     └FAIL, FAILS or FAILING    ┘
```

EXISTS will limit retrieval to data sets whose specified element is valued. FAILS limits retrieval to data sets whose specified element has no data value. They may be used to arrange data in a specified format, as the example above indicates. These commands may also be used to check data integrity:

```
PRINT ACCOUNT NUMBER WHERE TENANT NAME FAILS:
```

5.2.12 System Functions, With WHERE Clause

Commands

```
PRINT COUNT TENANT NAME, SUM PERSONS CURRENTLY EMPLOYED,
AVERAGE MONTHLY GROSS RENT, MAXIMUM MONTHLY GROSS RENT
WHERE PROJECT NAME EQ ADAMS HOUSES:
PRINT MINIMUM MONTHLY GROSS RENT WHERE PROJECT NAME
EQ ADAMS HOUSES:
PRINT SIGMA MONTHLY GROSS RENT WHERE PROJECT NAME EQ
ADAMS HOUSES:
```

Printout

```
PRINT COUNT TENANT NAME, SUM PERSONS CURRENTLY EMPLOYED,
AVERAGE MONTHLY GROSS RENT, MAXIMUM MONTHLY GROSS RENT
WHERE PROJECT NAME EQ ADAMS HOUSES:

COUNT 9*     38
SUM 45*      18
AVG 37*      58.44736842
MAX 37*     125.

PRINT MINIMUM MONTHLY GROSS RENT WHERE PROJECT NAME
EQ ADAMS HOUSES:

MIN 37*     026.

PRINT SIGMA MONTHLY GROSS RENT WHERE PROJECT NAME EQ
ADAMS HOUSES:

SIGMA 37*    22.94847032
```

Explanation The six system functions used above are an integral part of SYSTEM 2000. Only one function may be specified with a component at one time. In the above example, the functions COUNT, SUM, AVERAGE and MAXIMUM are shown for illustrative purposes only in a single print request. The other two functions are shown, again for illustrative purposes only, in individual print requests. The output format between the two methods is different. COUNT is the function used to count the number of occurrences of element values, repeating groups, or whole entries. Examples of each, in the order just given, are as follows:

```
PRINT COUNT TENANT NAME WHERE . . . .
PRINT COUNT CHARGES WHERE . . . .
PRINT COUNT ENTRY WHERE . . . .
```

Since the function COUNT merely counts the number of times something occurs, it can be used to count alphabetical and/or numeric data. SUM is the function used to add the numeric value of elements which are requested.

Therefore, SUM must be applied to numeric data, in the same manner as the functions AVERAGE and SIGMA (Standard Deviation). The SIGMA formula used in SYSTEM 2000 is as follows:

$$\text{SIGMA} = \sqrt{\frac{\Sigma X^2 - \frac{(\Sigma X)^2}{n}}{n - 1}}$$

The functions, MAXIMUM and MINIMUM, can be applied to alphabetical as well as numeric data. The set of values which will be used to obtain the function's value is determined by the presence or absence of a WHERE clause.

If a WHERE clause exists, then only that part of the data base which satisfies the WHERE clause is used to evaluate the function. If no WHERE clause exists, then the entire data base qualifier and each unique value (but not their multiple occurrences) for the element determines the function's output.

The six system functions cannot be referenced in the WHERE clause.

5.2.13 System Functions, Without WHERE Clause

Command PRINT COUNT BIRTH COUNTRY:

Printout

```
PRINT COUNT BIRTH COUNTRY:
COUNT 13* 7
```

Explanation As indicated in earlier examples, without a WHERE clause on the print request, only unique values are considered by the functions. This example illustrates how this system characteristic may be used to good advantage when the user wants only unique values considered. Each tenant has an element called BIRTH COUNTRY. If the user wanted to know how many different countries of birth exists within all projects, he would use this command. The answer 7 indicates that even though there are 538 countries of birth stored in this data base, there are only 7 different ones.

5.2.14 Logical Operators

Command

```
PRINT PROJECT NAME, TENANT NAME, NET INCOME FOR RENT WHERE
NET INCOME FOR RENT GT 8000. AND CLASSIFICATION OF NEW RENT
EQ WELFARE:
```

Printout

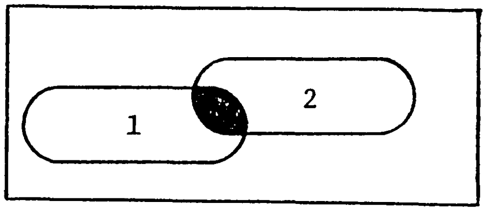
```
PRINT PROJECT NAME, TENANT NAME, NET INCOME FOR RENT WHERE

NET INCOME FOR RENT GT 8000. AND CLASSIFICATION OF NEW RENT

EQ WELFARE:

  1* GOMPERS HOUSES
     9* ALCANTARA FIDEL
        36* 8971.
  1* EASTCHESTER GARDENS
     9* BAILLY RENE M
        36* 8330.
  1* ROOSEVELT HOUSES
     9* BOSLER GEO W
        36* 8441.
  1* ROOSEVELT HOUSES
     9* BETER D M
        36* 9600.
  1* ST. NICHOLAS HOUSES
     9* LAW RUSSELL
        36* 8487.
```

Explanation The logical operators AND, OR, and NOT are used to indicate the relation between the conditions found in the WHERE clause. When AND is used, the data set values eligible for output must satisfy both conditions. When OR is used, the WHERE clause is satisfied by those data sets for which either condition is satisfied. When NOT is used, the WHERE clause is satisfied by all data sets which do not satisfy a condition. The format of the three Logical Operators is as follows:

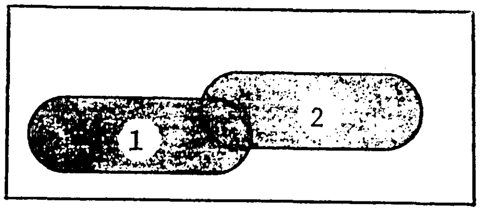
PRINT ... WHERE condition_1 AND condition_2:
PRINT ... WHERE condition_1 OR condition_2:
PRINT ... WHERE NOT condition_1:

The shaded areas in each of the diagrams below illustrate the logic of the three operators:

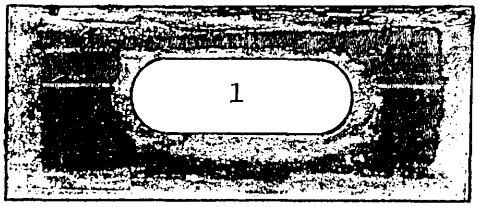
AND:



OR:



NOT:



5.2.15 Logical Operator Combinations

Command PRINT TENANT NAME, GROSS ANTICIPATED INCOME, NET INCOME FOR RENT, CLASSIFICATION OF NEW RENT WHERE GROSS ANTICIPATED INCOME SPANS 5000., 6000. OR NET INCOME FOR RENT SPANS 4000., 5000. AND CLASSIFICATION OF NEW RENT EQ WELFARE:

Let us examine this PRINT command to see what the user wants to retrieve. He wants a listing of tenant names, along with each of their gross anticipated incomes, net income for rent, and classification of new rent subject to the following criteria:

(a) Their gross anticipated income falls between $5000 and $6000 annually, OR

(b) Their net income for rent falls between $4000 and $5000 AND the classification of new rent is welfare.

Printout

```
PRINT TENANT NAME, GROSS ANTICIPATED INCOME, NET INCOME
FOR RENT, CLASSIFICATION OF NEW RENT WHERE GROSS ANTICIPATED
INCOME SPANS 5000., 6000. OR NET INCOME FOR RENT SPANS
4000., 5000. AND CLASSIFICATION OF NEW RENT EQ WELFARE:

  9* AURIEMMA FRANK L
     35* 5340.
     36* 4968.
     40* WELFARE
  9* BREINDEL SAML
     35* 4752.
     36* 4052.
     40* WELFARE
  9* CENCHEK MICHL
     35* 5290.
     36* 4531.
     40* SURCHARGE
               .
               .
               .
  9* BECKER LOUIS I
     35* 5300.
     36* 5028.
     40* SURCHARGE
```

5.2.16 Nested Logical Operators

Command

```
PRINT TENANT NAME, GROSS ANTICIPATED INCOME, NET INCOME FOR
RENT, CLASSIFICATION OF NEW RENT WHERE (GROSS ANTICIPATED
INCOME SPANS 5000., 6000. OR NET INCOME FOR RENT SPANS
4000., 5000.) AND CLASSIFICATION OF NEW RENT EQ WELFARE:
```

Printout

```
PRINT TENANT NAME, GROSS ANTICIPATED INCOME, NET INCOME FOR
RENT, CLASSIFICATION OF NEW RENT WHERE (GROSS ANTICIPATED
INCOME SPANS 5000., 6000. OR NET INCOME FOR RENT SPANS
4000., 5000.) AND CLASSIFICATION OF NEW RENT EQ WELFARE:

  9* AURIEMMA FRANK L
     35* 5340.
     36* 4968.
     40* WELFARE
  9* BUCKY GERALD
     35* 5956.
     36* 3861.
     40* WELFARE
  9* BREINDEL SAML
     35* 4752.
     36* 4052.
     40* WELFARE
```

Explanation This example illustrates nesting, within parentheses, of the OR connective. This causes the statement in which the OR is embedded to be processed before the AND operator.

There is an order of processing implicit with the use of these logical operators. Without nesting, as in the first example, the order of processing is NOT, AND and OR. The user can further control his retrieval by nesting the operators in one of the following formats, keeping in mind that statements enclosed by paired parentheses are processed before unnested conditions.

PRINT ... WHERE (condition$_1$ AND condition$_2$):

Since no conditions lie outside of the parentheses, they are ignored and the WHERE clause is processed as though the parentheses had been omitted.

PRINT ... WHERE NOT (condition$_1$ AND condition$_2$):

Because of nesting, the statement within the parentheses is processed first; all data which meets both condition$_1$ and condition$_2$ is selected. The NOT connective causes this data to then be excluded. Remove the parentheses and see how this changes the criteria.

PRINT ... WHERE (condition$_1$ OR condition$_2$) AND (condition$_3$ AND condition$_4$):

First, the OR of (condition$_1$ OR condition$_2$) is found; next, the AND of (condition$_3$ AND condition$_4$) is found, and finally the AND of these two results is found.

> PRINT ... WHERE (condition$_1$ OR condition$_2$) AND (NOT (condition$_3$ AND condition$_4$)):

(a) The AND of condition$_3$ and condition$_4$ is found; the NOT of that result is then found.

(b) The OR of condition$_1$ and condition$_2$ is then found.

(c) The AND of results (a) and (b) is then used for output.

Conditions 3 and 4 are nested two levels deep while condition 1 and 2 are nested one level deep. The request contains <u>two nestings</u> of conditions.

5.2.17 The Use of AT

Command PRINT PROJECT NAME, TENANT NAME, CHARGES WHERE AMOUNT OF CHARGE EXISTS AT 2:

Printout PRINT PROJECT NAME, TENANT NAME, CHARGES WHERE AMOUNT OF CHARGE EXISTS AT 2:

```
1*  BARUCH HOUSES
  9*  CADOGAN EDITH
    58*  11/15/69
    59*  LEGAL FEES
    60*  P + P NOTICES
    61*  2.00
    58*  12/10/69
    59*  SERVICFS
    60*  LOCKOUTS
    61*  .50
1*  BARUCH HOUSES
  9*  CAFFREY PATK
    58*  11/15/69
    59*  LEGAL FEES
    60*  P + P NOTICES
    61*  2.00
    58*  12/10/69
    59*  SERVICES
    60*  LOCKOUTS
    61*  .50
         .
         .
         .
```

```
1*  CARVER HOUSES
  9*  BRYANT KATHERINE
    58*  11/15/69
    59*  LEGAL FEES
    60*  P + P NOTICES
    61*  2.00
    58*  12/12/69
    59*  FINES
    60*  STOVE OVEN DOOR - COMPLETE
    61*  5.00
```

Explanation The format of the AT phrase is as follows:

```
... WHERE <condition> AT <n>:
```

The AT phrase means that the nth data set in a sequence of data sets is to be tested for the condition. "n" must be a positive integer. The example above indicates the AMOUNT OF CHARGE is a component in a repeating group, CHARGES, having multiple values associated with it. Each logical entry that has two or more occurrences will qualify the right hand side of the WHERE clause for possible output. When "AT 0" is used, the user is requesting the last occurrence of the condition.

5.2.18 The Use of HAS, HAVE, HAVING

Command

```
PRINT TENANT NAME WHERE ACCOUNTS HAS DESCRIPTION OF CHARGE
EQ P+P NOTICES AND ACCOUNTS HAS DESCRIPTION OF CHARGE EQ
LOCKOUTS:
```

Printout

```
PRINT TENANT NAME WHERE ACCOUNTS HAS DESCRIPTION OF CHARGE
EQ P+P NOTICES AND ACCOUNTS HAS DESCRIPTION OF CHARGE EQ
LOCKOUTS:

  9*  CADOGAN EDITH
  9*  CAFFREY PATK
```

Explanation One of the basic structural rules of SYSTEM 2000 is that an element can only assume one value within a single data set. If, however, it is desired to retrieve something which can only be qualified on the basis that it has different values for the same element occurring in different data sets, then the HAS command must be used.

The standard format for the use of HAS is as follows:

```
... WHERE <repeating group>HAS<condition>:
```

The system logic associated with the use of the HAS command is in giving the user the ability to choose the level at which a data set will become a Qualified Data Set (see Concept of Normalizing discussion). The <repeating group> given to the left of the HAS in the WHERE clause statement actually specifies the data sets which can qualify and become Qualified Data Sets.

When constructing the WHERE clause, the user must be aware of the data base tree structure sufficiently to select a <repeating group> which is an ancestor to the location of the <condition> being constructed. Choosing any ascending data set node above the conditional location will accomplish the stated purpose of the retrieval. If level zero data sets are to become the qualified data sets, the user may choose ENTRY.

However, the user must choose a <repeating group> at or above the data tree nodal location of the Specified Data Sets or else output redundancies will occur. This, then is the secondary use of the command HAS which allows the control of output redundancies. In a WHERE clause statement, the various conditions can create multiple Qualified Data Sets. If more than one data set qualifies as a Qualified Data Set and any of them occur at a level below the Specified Data Sets, they will "nominate" the Specified Data Sets for output as Selected Data Sets as many times as they qualified. An example of this secondary use will assist understanding at this point.

Command

```
PRINT PROJECT NAME WHERE AMOUNT OF CHARGE EXISTS AT 2:
```

Printout

```
PRINT PROJECT NAME WHERE AMOUNT OF CHARGE EXISTS AT 2:

   1*  BARUCH HOUSES
   1*  BARUCH HOUSES
   1*  BARUCH HOUSES
   1*  CARVER HOUSES
   1*  CARVER HOUSES
```

Explanation The output indicates that Baruch Houses had three accounts which had two charges and Carver Houses had two accounts which had two charges. This is true since the request asks that the qualification occur at the lower level 2 where AMOUNT OF CHARGE exists and each data set which qualified would select the PROJECT NAME to be output. If the user wished only to know which projects had two charges per tenant then he would phrase the request in the following manner:

Command

```
PRINT PROJECT NAME WHERE ENTRY HAS AMOUNT OF CHARGE EXISTING
AT 2:
```

Printout

```
1*  BARUCH HOUSES
1*  CARVER HOUSES
```

It is possible to construct a very complicated WHERE clause with many conditions. If there is a combination of HAS and non-HAS conditions in a WHERE clause and the logical operator AND ties them together, then the repeating group component specified in the HAS condition must be ancestorily related to the element appearing in the non-HAS condition to produce output results. That is, both the repeating group component and the non-HAS element must be in the same data tree. If different repeating group components are used in the WHERE clause, they also must be ancestorily related, or belong to the level Ø repeating group ENTRY. This does not hold if the logical operator used is OR.

If the repeating group name happens to be plural in the definition or proper English would dictate, the user may use HAVE instead of HAS; HAVING is also acceptable. The AT n phrase may be used with the HAS option, which has already been demonstrated.

5.2.19 The Use of DITTO

Command

```
PRINT PROJECT NAME, TENANT NAME, DESCRIPTION OF CHARGE WHERE
PROJECT NAME EQ BARUCH HOUSES AND DESCRIPTION OF CHARGE EXISTS
AT 2:
DITTO WHERE PROJECT NAME EQ BLAND HOUSES AND DESCRIPTION OF
CHARGE EXISTS AT 2:
```

Printout

```
PRINT PROJECT NAME, TENANT NAME, DESCRIPTION OF CHARGE WHERE

PROJECT NAME EQ BARUCH HOUSES AND DESCRIPTION OF CHARGE EXISTS

AT 2:
 1*  BARUCH HOUSES
     9*  BOOKSTAVER BURTON J
         60*  MARSHAL
 1*  BARUCH HOUSES
     9*  ARLINSKY REUBEN
         60*  CLOTHES DRYER - COMPLETE
 1*  BARUCH HOUSES
     9*  ADAMO LUIGI
         60*  LIGHT FIXTURE GLOBE
 1*  BARUCH HOUSES
     9*  CADOGAN EDITH
         60*  LOCKOUTS
 1*  BARUCH HOUSES
     9*  CAFFREY PATK
         60*  LOCKOUTS
 1*  BARUCH HOUSES
     9*  BROMFELD MAX
         60* LOCKOUTS
```

```
DITTO WHERE PROJECT NAME EQ BLAND HOUSES AND DESCRIPTION OF CHARGE
EXISTS AT 2:
```

```
 1*  BLAND HOUSES
     9*  CARSON J B
         60*  TOILET SEAT WITH COVER
 1*  BLAND HOUSES
     9*  CELLA E R
         60*  LOCKOUTS
```

Explanation Instead of repeating the conditions listed in the PRINT ... specifications, use of DITTO regenerates the same specifications used in the PRINT command immediately preceding, which can then be subjected to different retrieval criteria in the WHERE clause, as follows:

```
DITTO WHERE <conditions exist>:
```

The use of DITTO requires a WHERE clause in the previous request.

5.2.20 The Use of SAME and SAME AND

Command

```
PRINT TENANT NAME, GROSS ANTICIPATED INCOME, NET INCOME
FOR RENT, CLASSIFICATION OF NEW RENT WHERE (GROSS ANTICIPATED
INCOME SPANS 5000., 6000. OR NET INCOME FOR RENT SPANS 4000.,
5000.)  AND CLASSIFICATION OF NEW RENT EQ WELFARE:

PRINT PROJECT NAME, TENANT NAME, MILITARY RECORD WHERE SAME:

DITTO WHERE SAME AND MILITARY RECORD EQ KOREAN VETERAN:
```

Printout PRINT TENANT NAME, GROSS ANTICIPATED INCOME, NET INCOME FOR RENT, CLASSIFICATION OF NEW RENT WHERE (GROSS ANTICIPATED INCOME SPANS 5000., 6000. OR NET INCOME FOR RENT SPANS 4000., 5000) AND CLASSIFICATION OF NEW RENT EQ WELFARE:

```
9*  AURIEMMA FRANK L
    34*  5340.
    36*  4968.
    40*  WELFARE
9*  BUCKY GERALD
    35*  5956.
    36*  3861.
    40*  WELFARE
9*  BREINDEL SAML
    35*  4752.
    36*  4052.
    40*  WELFARE
```

PRINT PROJECT NAME, TENANT NAME, MILITARY RECORD WHERE SAME:

```
1*  GOMPERS HOUSES
    9*  AURIEMMA FRANK L
           54*  NO VETERAN OR SERVICE MAN
1*  MARKHAN GARDENS
    9*  BUCKY GERALD
           54*  KOREAN VETERAN
1*  POMONOK HOUSES
    9*  BREINDEL SAML
           54*  NO VETERAN OR SERVICEMAN
```

DITTO WHERE SAME AND MILITARY RECORD EQ KOREAN VETERAN:

```
1*  MARKHAN GARDENS
    9*  BUCKY GERALD
           54*  KOREAN VETERAN
```

Explanation SAME means that the contents of the WHERE clause of the immediately preceding retrieval request are to be used in the WHERE clause of the next request. SAME does for the right-hand portion of the WHERE clause what DITTO does for the left-hand side. If there was an error in the previous request, then no results will be given for a SAME request. A record is kept of WHERE clause results; thus, SAME never involves reprocessing any WHERE clause. The command SAME can be used with any of the three logical operators; AND, OR, or NOT followed by a condition statement. A modification of the immediately preceding request WHERE clause can be expressed by using SAME and any number of additional conditions. SAME must immediately follow WHERE. A modified SAME clause becomes the "SAME" clause of the next request, and so on, allowing the user to browse through the data base without repeating WHERE clause conditions. As shown, DITTO can be used in these combinations.

5.2.21 Strings

SYSTEM 2000 users have the ability to associate long strings of characters together, name the string, and subsequently recall the whole set of characters with only the name. This provides the user with an ability to reference long strings of characters, as in a request with multiple components, merely by naming them. The name given a character string must be preceded by a system separator, as in the following example.

Assume that the user wishes to reference the stub series contained in the following PRINT command:

```
          PRINT C1, C2, C3, C4, C5, C6 WHERE C5 EXISTS:
```

The user creates the string, PRINT C1 ... C6 as follows:

```
          DEFINE:
          OLD DATA BASE data base name:
          1*  PROJECT REPORT (STRING(PRINT C1, C2, C3, C4, C5, C6)):
          REMAP:
```

Later, the string may be used in the following way:

```
          RETRIEVAL:
          *PROJECT REPORT*  WHERE C5 EXISTS:
```

Note that in the PRINT command, the name of the character string, PROJECT REPORT, must be bracketed by the system separator, *. The general format for associating strings is:

```
          nnnn ss string name (STRING (any arbitrary string of characters)):
```

where: nnnn = an integer number between 1 and 9999,
ss = system separator, and
string name = any user chosen name.

The commands used to describe Strings are as follows:

```
DESCRIBE STRINGS:
DESCRIBE S#:
DESCRIBE S# THROUGH S#:
DESCRIBE S# THRU S#:
DESCRIBE S# TO END:
```

5.2.22 User-Defined Functions

The SYSTEM 2000 user may define his own functions to support his unique operational requirements. This may be done when the data base is originally defined or may be accomplished during its operational utilization. Operating with the Portfolio I Data Base, an example is given showing the defining of a function and its use in retrieval after defining. The user must request the DEFINE Module to initiate this action.

Command

```
DEFINE:
OLD DATA BASE PORTFOLIO:
77* DIVIDEND YIELD (FUNCTION(C23/C24*100)):
REMAP:
```

Later, the function may be used in the following way:

```
RETRIEVAL:
PRINT PORTFOLIO NAME, NAME OF STOCK, F77 WHERE PORTFOLIO NAME
EQ SPECIAL AND DIVIDEND EXISTS:
```

Printout

```
PRINT PORTFOLIO NAME, NAME OF STOCK, F77 WHERE PORTFOLIO NAME
EQ SPECIAL AND C23 EXISTS:

   9*  SPECIAL
     13*  NATOMAS
    F77*       .2680965147
   9*  SPECIAL
     13*  KAISER STEEL
    F77*       2.076124567
```

```
9*  SPECIAL
  13*  SYNTEX
 F77*       .5228758170
9*  SPECIAL
  13*  ELECTRONIC DATA SYSTEMS
 F77*       0.
9*  SPECIAL
  13*  INFORMATICS, INC.
 F77*       0.
```

Explanation The newly defined function is used in the PRINT command just like a component. In this case, however, the function, F77, DIVIDEND YIELD, did not request the retrieval of stored values, but the results of a calculation based upon the retrieval of stored values. The actual function is the quotient of the DIVIDEND, C23, divided by the CURRENT PRICE, C24, multiplied by the constant, 100.

The commands used to describe Functions are as follows:

```
DESCRIBE FUNCTIONS:
DESCRIBE F#:
DESCRIBE F# THROUGH F#:
DESCRIBE F# THRU F#:
DESCRIBE F# TO END:
```

The user-defined functions cannot be referenced in the WHERE clause.

5.2.23 Maintenance Function Commands

All of the foregoing RETRIEVAL commands have had some direct relevance in retrieving data base information for the purpose of collecting and using the information in support of some operational need. There are two other commands available in the RETRIEVAL Module which are used for what might be called a maintenance function: they manipulate the data base.

UNLOAD and UNLOAD WHERE

The UNLOAD and UNLOAD WHERE commands are used to extract data from the data base, like any of the other commands discussed. However, these commands produce LOADER string input suitable for loading into a defined data base. The format of the UNLOAD commands is as follows:

```
          UNLOAD:

          UNLOAD WHERE <conditions exist>:
```

The results of an UNLOAD operation are sent to the REPORT FILE, like any other retrieval. However, the loader input string produced by an UNLOAD contains no carriage control for vertical spacing on the printer and would lead to an uncontrollable print. Therefore, the user must specify a REPORT FILE other than the default setting by use of the REPORT FILE IS < >: command.

RELOAD

The RELOAD request causes the entire data base to be unloaded and automatically rewritten into the same data base of the same name without data change. Should the internal file structures be made less efficient by a voluminous number of data base updates, this technique will regain the original retrieval efficiency. The format of this command is simply RELOAD:.

5.2.24 Concept of Normalizing

SYSTEM 2000 uses the concept of normalization, or normalizing, within the RETRIEVAL and UPDATE modules. The normalizing concept has to do with the problem of selecting the appropriate elements or repeating groups for retrieval or update operations. This is important to the user because, understanding the concept, he can construct his command to achieve the unique results desired.

Three new terms are required at this point to assist in the understanding of this new concept.

(1) Specified Data Sets - the data sets implicitly identified by the components listed to the left of the WHERE clause statement in a retrieval request and located to the left

of the EQ of the update request. The components may be identified by name or by number. The data sets are implicitly identified by requesting elements within the data sets or naming the repeating group which generates the Specified Data Sets. The level within the data base structure where these Specified Data Sets occur is of great importance. If elements are used to identify the Specified Data Sets, then the elements and the data sets will be at the same level, since the elements come from within the data set. If a repeating group is used to identify the data sets, then the Selected Data Sets will occur at one level below the listed repeating group.

(2) Qualified Data Sets - data sets that satisfy the <u>entire</u> WHERE clause condition(s). When the system starts processing a command it processes the WHERE clause first. Those data sets which satisfy all the conditions in the WHERE clause are temporarily "collected" and tested to see if any of the Qualified Data Sets are named or implied by the Specified Data Sets. If they are, they become:

(3) Selected Data Sets - data sets produced from the Qualified Data Sets. That is, those data sets named by the Specified Data Sets and selected by qualification for retrieval or update modification. Within the UPDATE module only, a full or partial tracing to the logical placement of a data set is possible. When this capability is used, the Selected Data Set is selected due to its position within the data base structure.

The remainder of this discussion will deal only with normalizing within the RETRIEVAL module. Some retrieval requests do not use the WHERE clause. If the request contains no WHERE clause, then the entire data base <u>qualifies</u> for output. If a WHERE clause is used, but no data sets satisfy the conditions imposed by the WHERE clause, no action is taken for the request. If a list of Qualified Data Sets is produced, then each Qualified Data Set is examined to see if it is also a Specified Data Set. If it is, then it becomes a Selected Data Set. If all components in the WHERE clause reference only the Specified Data Sets, then all Qualified Data Sets are also the Selected Data Sets. The following examples use the Sample PORTFOLIO Data Base Definition, Figure 2, and the Data Base Structure ... PORTFOLIO Loader String, Figure 6.

```
PRINT PORTFOLIO NAME, MANAGER WHERE PORTFOLIO NAME EXISTS:
```

Effect: The Qualified Data Sets are the Selected Data Sets. The Qualified Data Sets are data sets numbers 2 and 11, because PORTFOLIO NAME has a value in both data sets. The Specified Data Sets indicate the same two data sets (the data sets were implied by use of the element, PORTFOLIO NAME, which is a member of the repeating group, PORTFOLIO.

The WHERE clause may frequently contain components which produce Qualified Data Sets which are not the same as, or do not occur at the same level as Specified Data Sets. If this happens, a level adjustment up or down must be made for each Qualified Data Set that does not belong to the Specified Repeating Group.

Level adjustment may be upward or downward or both depending upon the Specified Data Sets and the relationship of the particular Qualified Data Set being adjusted. Upward level adjustment takes place when the Qualified Data Set is a descendant (at whatever level) lying below the Specified Data Set according to the hierarchy of the data base definition. The adjustment follows an upward path from the Qualified Data Set to its ancestral data set, the Specified Data Set. No additional data sets are ever selected by upward level adjustment; one and only one Selected Data Set can be found and substituted for the Qualified Data Set. If ENTRY or CO is the Specified Data Set and the Qualified Data Sets lie below level 0, then the upward level adjustment selects entire logical entries and the Selected Data Sets are all level 0 data sets.

PRINT STOCKS WHERE DATE GT 07/01/69:

Effect: The Qualified Data Sets are those sets at level 3 where date is greater than 07/01/69. The Specified Data Sets are at level 2. The Qualified Data Sets are adjusted upward one level to the Specified Data Set level for retrieval of the Selected Data Sets. In this example, data set 6 is the only Qualified Data Set; it is normalized upward to data set 3, the Specified Data Set; and then the entire contents of data set numbers 3, 4, 5 and 6 are output. This illustrates that when a repeating group number is specified in the PRINT clause not only Selected Data Sets are output, but all of their descendants as well, unless otherwise qualified within the retrieval request.

If the Specified Data Sets are named by an element from within a repeating group, then the Selected Data Sets will contain the elements specified and descendants are not displayed. If, however, the Specified Data Sets are named by a repeating group, then not only data sets from the same level as the repeating group are output as Selected Data Sets, but all descendants from the Selected Data Sets are output as well.

Downward level adjustment takes place when the Qualified Data Sets occur at a higher level than do the Specified Data Sets. Another way of stating the same situation is to say that the Qualified Data Set is an ancestor of the Specified Data Set. All paths are followed downward from each Qualified Data Set to all descendant Specified Data sets and these become the Selected Data Sets. Downward level adjustment may produce none, one, or many Selected Data Sets from one Qualified Data Set depending upon how many descendants exist. Again, unless otherwise qualified, all descendants from the Selected Data Sets are output, if, the Specified Data Sets are named by use of a repeating group.

PRINT NAME OF STOCK WHERE TRANSACTION TYPE EQ BUY:

Effect: In this example, there are two Qualified Data Sets, data set numbers 4 and 8; number 4 is normalized up to data set 3 where the NAME OF STOCK is AMERICAN CYANAMID, and number 8 is normalized up to data set 7 where the NAME OF STOCK is GENERAL MOTORS. The output results in two Selected Data Sets and no descendant data sets. The Specified Data Sets were named by an element which calls for an output of the selected elements:

```
13*  AMERICAN CYANAMID
13*  GENERAL MOTORS
```

When the Qualified Data Set does not belong to the same family repeating group represented by the Specified Data Set, the level adjustment follows an upward path from the node occupied by the Qualified Data Set to the first intersection with a node in the data tree of the disjoint repeating group data sets, then downward from that common intersection. All Specified Data Sets and thus all Selected Data Sets are selected below the common intersection. Repeating groups always have a meet at level 0, but the intersection might occur at lower levels, depending upon the logical entry definition.

PRINT NAME OF STOCK WHERE NAME OF ISSUER EXISTS;

Effect: Even though the two repeating groups, STOCKS and BONDS occur at the same level, you cannot get from one to the other without normalizing up and then back down. The Qualified Data Set is number 10. To get to the Specified Data Sets the system normalizes up to the common node, data set number 2, and then back down to data sets 3, 7 and 9 which become the Selected Data Sets. The output would be:

```
13*  AMERICAN CYANAMID
13*  GENERAL MOTORS
13*  UPJOHN
```

One very important facet of normalizing remains: the frequent occurrence of multiple Qualified Data Sets. When the right hand side of a WHERE clause creates more than one Qualified Data Set, the system may produce duplicate values in the output, unless otherwise controlled. This happens because the normalization process is repeated from each Qualified Data Set to every Specified Data Set, to finally arrive at Selected Data Sets. An example will illustrate this point.

```
PRINT NAME OF STOCK WHERE TRANSACTION TYPE EQ SELL:
```

Effect: This WHERE clause produces two Qualified Data Sets, numbers 5 and 6. The system normalizes data set number 5 up to data set number 3, where NAME OF STOCK exists as AMERICAN CYANAMID. But Qualified Data Set number 6 also normalizes up to the same data set number 3. Each normalizing action is unique and results in the following output:

```
13*  AMERICAN CYANAMID
13*  AMERICAN CYANAMID
```

If the user desires to control the level at which qualification takes place, he may further constrain the WHERE clause statement by use of the HAS command. The secondary use of HAS actually allows the user to specify the level at which qualification takes place. (See the discussion on the use of HAS, HAVE.)

5.3 OUTPUT FORMAT

The current effective options regarding indentation, spacing and inclusion or exclusion of component numbers rule the overall output display.

5.3.1 Output Format for Data Values

All Dates, Name and Text type data values are displayed exactly as they were entered into the data base. All numeric data are output in the format of the first occurrence of each unique numeric value within a component. If, for example, a component defined as a type integer element has as its first value the number 10 expressed in the format of 00010, additional occurrences of the unique value 10, regardless of the number of leading zeros,(e.g., 10, 010, 0010, 00000010) will be displayed as a five digit number, 00010. Similarly, occurrences of decimal and exponential numbers will be displayed in the format of the first occurrence of each unique value for each element, regardless of leading and trailing zeros in later occurrences of the value.

5.3.2 Output Format For System Functions

The standard output format for all system functions (SUM, COUNT, MAX, MIN, AVG or SIGMA) includes the function name as part of the stub display (e.g., SUM 1* 297). If, however, the stub suppress option has been used, the function name as well as the component number is suppressed (e.g., 297).

The results of COUNT are always displayed as integer numbers. The SUM and AVG of integer numbers are displayed as integer numbers. The SUM and AVG of decimal and exponential numbers are displayed in ten place decimal format, if values permit, and 20 places decimal format if values dictate. SIGMA or Standard Deviation of all numeric values are displayed in ten digit decimal format. The output of MIN and MAX follows the format of stored data values described earlier.

5.3.3 Output Format For User-Defined Functions

When a user-defined function is requested under the normal STUB display, the output associates an F before the component number to differentiate between functions and elements (e.g., F75* 200). The type of function (integer, etc.) determines the output format of the function value. If the numeric function type was not defined by the user, DECIMAL format is assumed. Output formats for the three numeric types of functions are:

DECIMAL FUNCTION	-	10 significant digits plus decimal point
INTEGER FUNCTION	-	up to a maximum of 15 digits
EXPONENTIAL FUNCTION	-	up to a maximum of 10 digits plus the exponential suffix

5.4 DEFAULT CONDITIONS IN THE RETRIEVAL MODULE

1. Output Specifications

a. Format Control Defaults - Standardly initialized to SINGLE SPACE, STUB, INDENT

Once a user has chosen his set of format controls for a given sequence of requests, he does not have to re-specify the format in subsequent requests in the same job. Format control options are retained and applied to subsequent requests as long as no other task is called. If another task module is called, then the next call to RETRIEVAL reinitiates all standard defaults.

b. Output Commands

The PRINT, DITTO and UNLOAD results are always sent to the REPORT FILE. The REPORT FILE is by default the on-line printer or the remote device unless changed by the user. (See System-Wide defaults.)

2. Output Display Order

a. Absence of WHERE Clause

1) Elements

When a WHERE clause does not exist in the request, element values requested are displayed by element with each set of values in ascending order.

2) Repeating Groups

If a repeating group is to be displayed, then the data values are displayed by the order of the entrance of the data set and data values into the data base. Elements of a repeating group may be displayed in any order but that order must be explicitly stated by the left-to-right order of elements specified in the output list; furthermore, if the elements of a given repeating group are grouped together in the request, that order overrides the order given in the definition of that repeating group.

b. Presence of WHERE Clause

1) If the request specifies PRINT ENTRY WHERE < >:, or PRINT <repeating group> WHERE < >:, then logical entries or

repeating group data sets are displayed in order of entrance into the data base according to the conditions that were satisfied in the WHERE clause.

2) If the PRINT clause names elements and repeating groups specifically, then for each data set satisfying the WHERE clause the values associated with that data set for those components specified in the PRINT clause are displayed in the order specified fully before displaying the values for the next qualified data set.

The order of qualified data sets is determined partly by the data base structure and partly by the WHERE clause conditions.

3) If any error occurs in a retrieval request, the job continues to try to process the next request.

6.0 UPDATE MODULE

6.1 INTRODUCTION

After the data base has been defined by the DEFINE module, and at least one logical entry loaded into the data base by the LOADER module, any additional data can be inserted into the data base or modified within the data base by use of the UPDATE module.

The UPDATE module modifies data values, data sets, or data trees. To modify means to add, change, remove, assign or insert. A data set may be defined as each set of values associated with a repeating group at a given level within a data base. A data tree is a data set at a given level plus all its direct descendant data sets.

An illustration of a data base structure will assist the understanding of data sets and data trees. The loader string example as shown in the LOADER module section, Figure 5, is expressed in a graphic manner as shown in Figure 6, Section 4.1. Figure 6 indicates that the data base contains two logical entries; City Trust Company and Good Life Insurance Company. Logical entry number one contains much more data than does logical entry number two. The data within number one is logically distributed into four levels, indicating the hierarchical relationship of data within SYSTEM 2000. Each box in Figure 6 is a data set. A data tree was just defined as any data set at a given level plus all its direct descendant data sets. An example of a data tree would be data set three, the one containing the information on AMERICAN CYANAMID, and the descendant data sets 4, 5 and 6, or the BUY-SELL information pertaining to transactions of AMERICAN CYANAMID. One way of describing the structure of logical entry number one is to say that the ORGANIZATION contains two PORTFOLIOS. The first PORTFOLIO, INCOME, contains three STOCKS and one BOND; the first STOCK contains three TRANSACTIONS; the second STOCK contains one transaction and the third STOCK contains no transactions. The second PORTFOLIO, TRUST, contains no STOCKS or BONDS and thus, no TRANSACTIONS. The second logical entry has no descendant data sets.

The varied capabilities within UPDATE include:

- * Adding or modifying information within existing data sets.
- * Removing information from existing data sets.
- * Adding or modifying partial or total logical entries (data trees).
- * Removing partial or total logical entries.

The user must request the UPDATE module before any UPDATE commands are legal. He does this by giving the system-wide command, UPDATE:, followed by one or more UPDATE commands for every UPDATE job submission. During any one job, the UPDATE module could be called several times by its system-wide command, if the UPDATE requests are interspersed with other module activity, such as DEFINE or RETRIEVAL. Every time the UPDATE module is called and an UPDATE command or commands results in successful modification of the data base, the system increments the data base version number by one so as to advise the user of the number of times his data base has been changed. If the specific UPDATE commands which follow are not legal, or fail to select any data sets for modification, the data base version number does not get incremented because there was no change.

There is no output as a result of UPDATE commands similar to the output resulting from RETRIEVAL commands. There are only four outputs that can ever occur as the direct result of an UPDATE command. They are:

(1) An Error Message, or

(2) Ø Selected Data Sets
Data Base Unaltered, or

(3) The New Data Base Version Number
The Number of Data Sets Selected for Modification.

(4) Update File Informative Messages

If the user desires verification of the UPDATE command results, he can request appropriate retrievals following his updates.

6.2 UPDATE COMMANDS

The UPDATE commands used to modify the working data base are discussed within this section. When preparing for an UPDATE request, the user decides:

(a) which UPDATE operation is desired, and

(b) which data sets are to be selected for the operation.

UPDATE commands are categorized as operations which either modify existing data values, data sets and data trees, or create new data sets and new logical entries. Operations modifying existing data sets may be further classified as either single level operations or multiple level operations. Single level UPDATE operations modifying existing data sets are:

(a) ADD
(b) CHANGE
(c) REMOVE
(d) ASSIGN

Multiple level (TREE) UPDATE operations modifying existing data sets are:

(a) REMOVE TREE
(b) ASSIGN TREE

The INSERT TREE operation creates new data sets before or after existing data sets. Though insertion implies a multiple level operation, the new tree might only contain only a single data set.

In all TREE operations, the TREE concept or structure must be kept in mind such that a TREE, by definition, has only one data set at whatever level it starts and can descend downward with multiple data sets occurring below the nodal data set.

Knowledge of the RETRIEVAL module will prove helpful when constructing the various UPDATE commands. Practically all UPDATE commands utilize the WHERE clause which is used and constructed exactly as discussed in the RETRIEVAL module. The WHERE clause qualifies data sets within the data base for possible modification. If a WHERE clause would be inappropriate, then a trace notation or a combination of trace notation with a WHERE clause can be used. A request containing no WHERE clause or trace notation is meaningless and illegal.

6.2.1 General Format for UPDATE Commands

The general format for all UPDATE commands is given in Figure 7. There are three different formats indicated. The initial word or words in each command give a direct indication of its power and capability as suggested in the previous section. The capability of each will be discussed in a separate section devoted to the command itself.

Descriptions of terms used in the general formats in Figure 7 are as follows:

Component Identification

Each of the commands must contain the component identification of the element or repeating group(s) to be modified. The legal values to be inserted here can be:

* an element or repeating group name, or
* an element or repeating group number, or
* the word ENTRY or its equivalent, CØ.

Data String

The data string contains the data values to be inserted into the data base. Its format consists of the following:

data values $_{\Delta}$[two (2) system separators][entry terminator word]

The data values consist of a single value when modifying a single element or a loader string when more than one element value is being modified. The entry terminator word is an optional item. The component identification and the data string must logically agree as follows:

Component Identification	Data String
element name or number	single value
repeating group name or number	loader string

Retrieval Conditions

The retrieval conditions following the WHERE clause are the same con-

Format 1

```
[ADD          or  AD]
[CHANGE       or  CH]   <component identification>[1] EQ <data string> WHERE <update conditions>:
[ASSIGN       or  AS]
[ASSIGN TREE  or  AT]
```

Format 2

```
[REMOVE       or  RE]   <component identification>[1] WHERE <update conditions>:
[REMOVE TREE  or  RT]
```

Format 3

```
    INSERT TREE  or  IT   <component identification> EQ <data string> [BEFORE] <update conditions>:
                                                                      [AFTER ]
or  INSERT TREE  or  IT   <component identification>[1] EQ <data string>[WHERE ] <update conditions>:
```

FIGURE 7

General Format for UPDATE Commands

[1] Immediately following the component identification, a trace notation may be inserted. Trace notation is discussed in section 6.2.11.

ditions as discussed in the RETRIEVAL module. The WHERE SAME and WHERE SAME modified commands may be used within the UPDATE module or across the UPDATE and RETRIEVAL modules.

6.2.2 The ADD or AD Command

Purpose To add data within existing data sets where no data currently exists.

General Format

ADD <component identification>[1] EQ <data string> WHERE <update conditions>:

Add One Element If a data value is to be added to a single element in all of the selected data sets, then the component identification is the user-defined name or number of the element and the data string is the single data value terminated by two system separators and, optionally, the entry terminator word. The following examples utilize the PORTFOLIO Data Base as defined in section 3.4.13.

```
ADD CURRENT DATE EQ 02/25/70  **END WHERE C1 EQ GOOD LIFE INSURANCE CO:
ADD C14 EQ UPJ **WHERE C13 EQ UPJOHN:
```

Adding Multiple Elements If a data value is to be added to more than one element, then the component identification is the name or number of the repeating group which associates the elements. The data string is composed of the elements with their data values in data string format. The order of the data values to be added is immaterial as long as each value is preceded by its appropriate element number. ENTRY or CØ signify the level 0 data sets.

```
ADD STOCKS EQ 14* UPJ 15* NYSE 17* 2834 16* PHARMACEUTICAL **
WHERE C9 EQ INCOME AND C13 EQ UPJOHN:

ADD C8 EQ 10* E 11* W.D. GARDNER **END WHERE C9 EQ TRUST:
```

6.2.3 The CHANGE or CH Command

Purpose To change data within existing data sets where data exists.

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see Section 6.2.11.

General Format

```
CHANGE <component identification>[1] EQ <data string> WHERE <update conditions>:
```

Change One Element Identifying the element(s) whose value is to be changed is like that of the ADD operation. If a data value is to be changed for a single element in all selected data sets, the component identification specifies the element and the data string contains the data value.

```
CHANGE MANAGER EQ B.J. DILLARD  **END WHERE C9 EQ INCOME:
CHANGE C24 EQ 27.25  **WHERE C13 EQ AMERICAN CYANAMID:
```

Changing Multiple Elements If several elements in a repeating group are to have their values changed, then the specified repeating group is identified. Order of data values in the data string is immaterial as long as each is preceded by the correct element number. Elements not mentioned in the data string are not affected.

```
CHANGE STOCKS EQ 14* TWA 15* NYSE 17* 4511 **WHERE C13
EQ TRANS WORLD AIRLINES:

CHANGE C25 EQ 36* BUY 27* 02/05/70 **END WHERE C9 EQ INCOME AND
C13 EQ CHUBB CORPORATION AND C26 EQ SELL AND C27 EQ 02/06/70:
```

Discussion The CHANGE command has exactly the same format as does the ADD command. The CHANGE command, however, can only change data values if the data values exist.

For each selected data set, the CHANGE operation looks at the status of each element specified in the request. If the element has a value, it is changed to the new value specified in the data string. If the element has no value, no action is taken. The CHANGE operation changes existing values; it never adds new values (see ADD or ASSIGN).

6.2.4 The REMOVE or RE Command

Purpose To remove data from selected data sets.

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see Section 6.2.11.

General Format

REMOVE <component identification>[1] WHERE <update conditions>:

Remove One Element The REMOVE operation removes the data value of either a single element or all elements in each selected data set. If the specified component is an element, then for each selected data set, if that element has a value, the value is removed. If it has no value, no action occurs.

```
REMOVE MANAGER WHERE MANAGER EQ B.J. DILLARD:
REMOVE C17 WHERE C13 EQ BOEING CO.:
```

Removing Multiple Elements or Data Sets If the specified component is identified as a repeating group (or ENTRY) then every value is removed from every selected data set. Remember, this can occur at only a single level.

```
REMOVE TRANSACTIONS WHERE C9 EQ RESERVE AND C13 EQ AMERICAN CYANAMID:
REMOVE C8 WHERE C9 EQ BALANCED:
```

Discussion The REMOVE operation only removes data at a single level, but it may affect the data set structure at upper levels. The system attempts to discard non-valued data sets. A data set will be discarded from the structure within the data base if it passes two tests:

(1) If it is non-valued (contains no data), and
(2) It possesses no valued descendant data sets.

If all of the data has been removed from the specified data set, the data set may be removed from the data base. If the subject data set has valued descendants hanging from it, then it is retained within the structure as a non-valued data set. If it does not, the data set is removed and its parent is located for potential removal. All data sets, upward from the subject data set, are inspected for contained values and for valued descendants.

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see section 6.2.11.

6.2.5 The ASSIGN or AS Command

Purpose To assign data within existing data sets whether or not the existing data sets contain data.

General Format

```
ASSIGN <component identification>1 EQ <data string> WHERE <update conditions>:
```

Assign One Element If the request specifies a single element and its single new data value, then each selected data set is examined to determine if the specified element has a value in that data set. If it has a value, the value is changed to the new value; if it does not have a value, then the new value is added.

```
ASSIGN DATE EQ 02/25/70 **END WHERE C1 EXISTS:
ASSIGN C24 EQ 28.75 **WHERE C13 EQ AMERICAN AIRLINES:
```

Assign Multiple Elements If the component identification specifies a repeating group by name or number, then each selected data set is entirely emptied of all values and then filled with as many new values as are found in the data string in the request. For a specified repeating group, a total REMOVE operation is done for each selected data set, then a total ADD operation. If a repeating group is specified and if any original data is still to be retained along with new assignment of values, the new data string must contain that original data also or it will be lost in the total removal. All values in the data string must belong to elements associated with the singularly specified repeating group. The order is immaterial as long as the value is preceded by its proper element.

```
ASSIGN ENTRY EQ 1* NEW FUND 2* BOB JONES 7* 02/25/70 **
WHERE C1 EQ PILFER FUND:

ASSIGN C8 EQ 9* MIXED 10* M-1 11* DAVE SMITH **END WHERE C9
EQ BALANCED:
```

Discussion The ASSIGN operation always alters the contents of all selected data sets. Unlike ADD and CHANGE operations which are conditional depending upon the

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see section 6.2.11.

status of the values in the data set, ASSIGN is unconditional in that it always assigns the new value(s). The ASSIGN operation never changes the structure of the data sets in the data trees. No matter how much the selected data set is emptied, it is always filled again with whatever is in the data string; the data set contents are altered, but the data set never disappears.

6.2.6 The REMOVE TREE or RT Command

Purpose To remove each selected data set and all of its descendant data sets.

General Format

```
    REMOVE TREE <component identification>[1] WHERE <update conditions>:
```

Specific Examples

```
    REMOVE TREE PORTFOLIO WHERE C9 EQ GROWTH:
    REMOVE TREE C12 WHERE C13 EQ HILTON HOTELS AND C9 EQ GROWTH:
```

Discussion The REMOVE TREE operation removes each selected data set and all of its descendant data sets. By specifying and selecting the parent data sets, entire data trees are removed without regard to the status of their contents. After each data tree has been removed, the remaining data trees in the data base are relinked to close the gap at the topmost level created by the removal. For example, if the second logical entry was removed, the first is then linked to the original third logical entry which now becomes the second logical entry in the data base.

As indicated earlier, the REMOVE command only removes the data from the specified data sets and does not remove data from data sets lying above or below the specified ones. The REMOVE TREE command as just shown, not only removes the data from the specified data sets but also removes the data and the data sets lying below the specified ones. Neither command contains a data string within the formatted command due to the fact that data strings within any command indicate the data values to be put into and not removed from the data base.

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see section 6.2.11.

6.2.7 The ASSIGN TREE or AT Command

Purpose To replace current data trees with new data trees.

General Format

```
ASSIGN TREE <component identification>[1] EQ <data string> WHERE <update
conditions>:
```

Specific Examples

```
ASSIGN TREE PORTFOLIOS EQ 9* PRIVATE 10* XYZ 12* 13* CONTROL DATA
15* NYSE 24* 56.25 25* 26* BUY 27* 02/25/70 28* 10000 29* 56.25
12* 13* LITTON INDUSTRIES 12* 13* COMPUTER TERMINAL CORPORATION
**END WHERE C9 EQ STOCK:

ASSIGN TREE ENTRY EQ 1* NEW-NAME FUND 2* J.J. GRANT 8* 9* WILDFIRE
8* 9* HOPEFUL 8* 9* LOADED **WHERE C1 EQ PILFER FUND:
```

Discussion Where the ASSIGN TREE command is used, the system checks the command for accuracy and then accomplishes a complete REMOVE TREE operation for the selected data sets, removing the information to allow the assignment of new information. The new information is contained in the data string portion of the command in loader string format. The first data set detailed in the loader string contains the new information for the specified repeating group given in the component identification. The subsequent or descendant data sets follow within the loader string and may contain as many data sets to as many levels as desired regardless of the original data tree that was removed.

Within the general format, the repeating group name may be ENTRY or CØ (zero) if an entire existing logical entry is going to be replaced by the ASSIGN TREE data string.

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see section 6.2.11.

6.2.8 The INSERT TREE Command

Purpose To add new data trees were data trees do not exist.

General Formats

(1) INSERT TREE <component identification> EQ <data string>

$\left[\begin{array}{l}\text{BEFORE}\\\text{AFTER}\end{array}\right]$ <update conditions>:

(2) INSERT TREE <component identification> <trace notation>[1] EQ <data string> WHERE <update conditions>:

Specific Examples

```
INSERT TREE ENTRY EQ 1* IOU FUND 2* JOHN DOE 8* 9* GLOSSY 12* 13*
BOEING CO. 12* 13* HILTON HOTELS 8* 9* BLACK 12* 13* AMERICAN AIRLINES
**BEFORE C1 EQ GOOD LIFE INSURANCE CO.:

INSERT TREE C25 EQ 26* BUY 27* 02/25/70 28* 5000 29* 52.25  **END
AFTER C27 EQ 06/30/69 AND C13 EQ HILTON HOTELS AND C9 EQ GLOSSY:
```

Discussion This command differs from all other UPDATE commands in that it creates a new data tree where no data tree ever existed. All other commands modify, in some way, existing data sets. Even the ASSIGN TREE command, which can create new data sets below the level of the specified repeating group must modify an existing data set where it attaches itself to the logical entry.

After inspection of the two formats just given for this command, it is apparent that several new concepts are introduced. The trace notation has always been optional in all of the UPDATE commands previously discussed. In the INSERT TREE command, a partial trace notation must be used when the command includes a WHERE

[1] A partial trace notation must be used in this command if a WHERE clause is used. See trace notation 6.2.11.

clause. The command name gives a direct clue to the next item of uniqueness, the ability to insert a data tree where the user desires. Besides the WHERE clause, two new words reflecting insertion location are introduced: BEFORE and AFTER. Either of these two terms may be used in place of the word WHERE to indicate a specific location to be satisfied after the update conditions have been met. When either of the words BEFORE or AFTER are used, a trace notation must not be used because of a direct contradiction of logic.

6.2.9 The Use of DITTO, SAME and PREVIOUS

There are three command words within the SYSTEM 2000 syntax which may be used to shorten the language required within an UPDATE command. Each of the three words are used similarly. These command words may only be used when multiple commands are submitted within the same job. When they are used, they stand for the related phrase in the preceding command.

DITTO This command word has the same meaning in the UPDATE module as when it is used in the RETRIEVAL module; it applies only to the portion of the request on the left hand side of the WHERE/BEFORE/AFTER clause. Any error in the previous request causes the job to halt; therefore, no action will be taken on subsequent DITTO requests in case of previous errors. Examples:

```
CHANGE C26 EQ SELL  **END WHERE C26 EQ DISPOSE:
DITTO WHERE C26 EQ S:
DITTO WHERE C26 EQ SL:
```

SAME Like the command word DITTO, SAME is identical to the SAME of the RETRIEVAL module. SAME and SAME modified apply only to the portion of the request following the WHERE/AFTER/BEFORE clause. An error detected in the previous request causes the job to halt and prevents subsequent action of an erroneous SAME condition.

SAME may be used across the RETRIEVAL and UPDATE modules. Examples:

```
UPDATE:
CHANGE C15 EQ OTC **WHERE C14 EQ EDS-U:
REMOVE C17 WHERE SAME:
RETRIEVAL:
PRINT C13, C15, C17 WHERE SAME:
```

PREVIOUS This word when used in an UPDATE command repeats the data string of the previous command. Any error in the previous command causes the job to halt and prevents subsequent action. Examples:

```
CHANGE TRANSACTIONS EQ 26* BUY 27* 03/19/70  28* 5000 29* 11.625
**END WHERE C13 EQ CONTINENTAL AIRLINES AND C9 EQ GROWTH:

ADD TRANSACTIONS EQ PREVIOUS WHERE C13 EQ CONTINENTAL AIRLINES
AND C9 EQ SPECIAL:
```

6.2.10 LIMIT Option

SYSTEM 2000 provides the user with the ability to establish a minimum and a maximum boundary for the number of data sets that may be selected for update action in any UPDATE command. This provides an effective safeguard against unwanted and unpredicted updates. The various commands appropriate to this capability are:

```
      (1)  LIMIT < integer number> :
 (or) (2)  LIMIT < integer number, integer number >:
(and) (3)  END LIMIT:
```

The integer numbers in the command refer to the number of allowable data sets to be selected by the command. If one integer number is given, then exactly that number of data sets must be selected. If two integer numbers are given, the first establishes the minimum number of data sets which can be selected. The second number establishes the maximum number of data sets that can be selected. The system count of selected data sets only includes the selected data sets which are horizontally related and does not include within the count any descendant data sets affected by the command. The LIMIT specification is good only during the SYSTEM 2000 job submission where the limits were set. All prior and subsequent jobs have a LIMIT default setting of zero, where zero means unlimited. During any one job, of course, the limit command may be used, changed several times, and then ended by the END LIMIT command prior to termination. The END LIMIT command causes the restoration of the zero setting, i.e., unlimited.

6.2.11 Trace Notation

The Concept of Normalizing as discussed in the RETRIEVAL module also applies within the UPDATE module to a great measure, particularly in the area of level adjustment, upward or downward, from the Qualified Data Sets. Within UPDATE, level adjustment is necessary because each selected data set that UPDATE operates on, must belong to the Specified Data Sets. After all necessary normalizing has been done for all Qualified Data Sets, the resulting list of Selected Data Sets is pruned by removing all duplications of Selected Data Sets. Thus, no UPDATE operation will ever affect a Selected Data Set more than once in a single request.

When adding or modifying new information to the data base, it is sometimes necessary to be able to point to a data set and say in effect, that is the one I want to change. This, of course, requires a rather specific knowledge of the affected data base, such that you know how and where to point. Pointing is done by identifying the exact data set at each level within a logical entry that forms the hierarchical linkage for the selected data sets. The activity just specified is called tracing and the data set identification is called trace notation.

Full Trace

Trace notation requires some knowledge of the data base structure. Trace notation selects a data set by its position as a node in the data base without regard to any condition of data values. Each integer number in the trace notation signifies the position of a data set within a particular subtree. Each integer number is separated from the next by the current separator; no imbedded blanks are allowed. The integer number immediately following the component identification refers to the level of the specified data set; the next integer refers to its parent at the next highest level; the next to the parent of the parent and so on. A WHERE clause is illegal when a full trace is used. In a full trace, an integer number must be supplied for each level starting with the level of the specified repeating group and ending with level Ø.

The integer numbers must be non-negative. A positive number, <n> denotes the nth position on a level (e.g. *14 means the 14th data set of the appropriate repeating group). The number Ø denotes the last position on a level.

Examples:

REMOVE TREE ENTRY*4:

> The fourth logical entry is removed from the data base.

ASSIGN PORTFOLIOS*2*Ø EQ 9* CLIMBER 11* J.D. Gilpen **END:

> The second PORTFOLIOS data set in the last logical entry is selected. Its contents are removed and replaced with the data string values.

<u>Combined WHERE Clause and Trace Notation</u>:

A partial trace extends upward to any desired level except level Ø. A trace extending all the way to level Ø is a full trace. A request with a partial trace always contains a WHERE clause. A full trace and a WHERE clause is meaningless and illegal.

In a request using a partial trace the WHERE clause is processed first as usual, and a list of qualifying data sets is produced. From the list of qualified data sets, <u>selected</u> data sets are chosen on the basis of the trace notation. Data set selection extends one level higher than the level of the specified data set, in order that the position of the specified repeating group within its parent repeating group can be determined when the partial trace is applied. These selected data sets are referred to as "selected ancestral data sets." Each "selected ancestral data set" is followed down through the trace notation to the specified position in the data tree. The data set occurring at the bottom of the trace becomes the selected data set to be affected by the UPDATE operation.

In general, the WHERE clause is used for broad selection of data sets, and the partial trace serves to isolate the nth data set below the generalized qualified data sets.

A partial trace included in an INSERT TREE ... WHERE ... request has a somewhat different meaning. Qualification and tracing does not change, but since insertion is specified, the partial trace means insert as the nth position.

Examples:

```
REMOVE C25*1 WHERE C1 EQ PILFER FUND:
```

The qualified data set is the level Ø data set for PILFER FUND. The selected ancestral data sets are all C12 repeating groups (STOCKS) for PILFER FUND. The trace extends to the first occurrence of C25 repeating group (TRANSACTION) and removes it.

```
ASSIGN C25*0*2 EQ 26* BUY 27* 02/25/70 28* 5000 29* 25.00  **END
WHERE C9 EQ RESERVE:
```

The qualified data set is the level 1 data set for the RESERVE portfolio. The selected ancestral data sets are all C8 (PORTFOLIOS) repeating groups. Since the WHERE clause qualified only one C8 repeating group (RESERVE portfolio), the selected ancestral data set is the RESERVE portfolio. Following the trace downward, the last transaction entered for the second occurrence of the STOCK repeating group is assigned new values.

6.3 UPDATE FILE CONTROL

The previous section has discussed the methods by which the user may modify the data within a SYSTEM 2000 data base. This section describes the SYSTEM 2000 capabilities that are additionally provided to record the data base modifications and to save the desired recordings in order to apply them to the data base at a later time.

6.3.1 General Information

When a data base is said to exist via the initial DEFINE and LOADER operations it occupies space on the disk, organized into eight tables. The version number of the definition as well as the data is number one. The user has several options at this point.

The user may decide to modify this working data base by making data base modifications via the previously discussed UPDATE commands. The data version number will be incremented by one each time an UPDATE job has been successfully completed. From this point the user may branch into the next option, if he desires.

The user will eventually want to save his working data base on tape. Whenever he takes this option, using the system-wide SAVE DATA BASE command, the data version saved on tape will equal the data version on disk because they are the same data base.

Up to this point, there has been no recording of the results of the update commands other than the actual data base modifications themselves. But once an archive data base tape has been made of the working data base, the user has the capability to modify, via UPDATE commands, the working data base independent of the archive tape. If no record were kept of these modifications then the two data bases could never be reconciled. The user can decide to continue in this operational manner by repetitively saving the new versions as they are produced. However, the user should carefully consider the additional options which are available.

Rather than keeping many archival tapes, the user can save a particular version of the data base and then maintain a record of modifications made to that base version so that the current data base or any intermediate version can always be reproduced by the archival recordings.

These recordings are kept on the Update File as directed by the user. The Update File is established by specifying the Update File Tape Visual Reel Number (uftvr#) when either a Load or Save Data Base command is given. This number, once specified, maintains the relationship between the archival data base saved on tape and the associated Update File.

Once this file has been established, the results of every successful update action are recorded on it. Each of these actions results in a recording. One or more recordings on the file constitute a Segment. A Segment is created as the result of actions taken between the call to the UPDATE module and any one of the following:

(1) Calling any other module
(2) A TERMINATE command is given
(3) End of job.

Each time a segment has been completed the data base version number is incremented by one which makes the archival data base one or more versions behind the data base on disk. The user can keep as many of these Segments as he wishes on the uftvr# and later apply as many Segments as necessary to recreate any desired version.

The user may wish to release his working data base from the disk. Prior to taking this action, he should consider saving some or all of the Segments on the Update File. Later when loading his archival data base, he may apply as many Segments as he wishes to his new working data base. If the user wishes to experiment with his working data base he may create a copy for his own use and even suspend the recording of update Segments.

Some of the commands referenced in the preceding discussion have been previously introduced in section 2.1.4, Data Base Control Commands, as system-wide commands. The remainder of the Update File commands are local to the UPDATE module and are introduced below.

6.3.2 KEEP ALL or KEEP <n> SEGMENTS Command

Purpose To copy update segments from the working update disk file to the Update File Tape.

Command Options (1) KEEP ALL:
(2) KEEP <n> SEGMENTS:
(3) KEEP <n> SEGMENT:
(4) KEEP <n>:

where n = positive integer number

Discussion If the user has saved an Archive data base, declared an Update File Tape Visual Reel Number, and wishes to keep update changes on the Update File Tape, this command allows the controlled recording of those update segments desired for permanent storage. The first segment recorded also records the following:

a. Data Base Name
b. Base Version Number
c. Date (b) was created
d. Time (b) was created

When the ALL command is used, all of the update segments which have been recorded on the disk file will be copied onto the tape. When n is used, n refers to the first <n> segments recorded on the disk file. The user should keep track of the number, <n>, of update segments which have been recorded on the Update Disk File if the user wishes to use the KEEP <n> SEGMENTS option.

If the user is in doubt as to the number of update segments which have been recorded on the disk file, he can use the KEEP ALL command. This action, when complete, creates an informative message output to the user, indicating the <n> segments which were kept. When the KEEP <n> SEGMENTS command is used, with <n> less than ALL, the final result is an automatic SUSPEND action.

6.3.3 SUSPEND Command

Purpose To erase or wipe out the Update Disk File and suspend further recording.

Command SUSPEND:

Discussion The suspension of update segment recording speeds up processing. This command causes the suspension of the recording of update segments on the Update Disk File and the erasing of any previously recorded update segments. This command would be used if the user wished to experiment with update commands but would not wish to record the results on the Update Disk File.

Automatic suspension occurs when any one of the following occurs:

(1) KEEP<n>SEGMENTS command is given, where<n>is less than ALL.

(2) If the user has never named the Update File Tape Visual Reel Number in either the Save or Load Data Base commands.

Suspension of the Update Disk File is lifted if the Update File Tape Visual Reel Number is named in either the Save or Load Data Base commands.

6.3.4 APPLY ALL or APPLY<n>SEGMENTS Command

Purpose To load and apply requested update segments from the tape to the accessed data base tables residing on disk.

Command Options (1) APPLY ALL:

(2) APPLY<n>SEGMENTS:

(3) APPLY<n>SEGMENT:

(4) APPLY<n>:

where <n> = positive integer number

Discussion These command options assume that update segments have previously been recorded and stored on the Update File Tape via the KEEP SEGMENTS command. These options allow the application of some or all of the kept segments to a compatible data base. Each segment which is applied to the data base will increment the data base version number by one.

The user may apply segments incrementally as long as no intervening changes are made to the data base such as individual update requests or a call to the LOADER module. If the data base is modified using commands other than the APPLY commands, then the working update disk file is suspended and no further segments can be applied. If the APPLY ALL: command is given, an informative message is issued telling the user how many segments were applied.

6.3.5 TERMINATE Command

Purpose To end an update session.

Command TERMINATE:

Discussion Any time the user is using the UPDATE module, the TERMINATE command may be issued. The result of this action is as follows:

(1) ends the current update session,
(2) creates an update segment on the Update Disk File, if the Update Disk File is not suspended,
(3) increments the data base version number, and
(4) leaves the user in the UPDATE module.

This command essentially allows the user to create an update segment without having to leave the UPDATE module. Selecting another module after giving some UPDATE commands or ending the job session will create an automatic TERMINATE.

Whenever an Update segment is created, an informative message is given to the user showing the current version number of the data base.

6.4 DEFAULT CONDITIONS IN THE UPDATE MODULE

(1) The limit on number of data sets selected for updating is unlimited unless a LIMIT command has been given.
(2) Padding and null options are effective as defined in the current definition for a data base and data base modification is performed accordingly.
(3) Any error in an UPDATE command terminates the job. This safeguards

unwanted destruction of the data base if update requests are dependent on previous update request results.

(4) Default mode is the suspended mode, i.e., no update file is produced unless an Update File Tape number has been specified.

(5) Terminate is automatic when user leaves the Update module or exits from the system; that is, a segment is created as though the TERMINATE command was given.

SYSTEM 2000™

DIAGNOSTIC MESSAGES

After collating the enclosed material, the version numbers* on each page should be as follows:

Pages	Version numbers
All Pages	1.1

*Version numbers appear in the document code which is in the upper right-hand corner of each page. For example, the version number embedded in RM-S2K-1.4 is the decimal number 1.4, indicating that this is the fourth revision of this particular page of the first edition.

SYSTEM 2000 DIAGNOSTIC MESSAGES

INTRODUCTION

SYSTEM 2000 contains a large set of user diagnostics. The diagnostics are generally of two types: error and informative messages. They are designed to keep the user apprised of system reaction to his actions. This document contains a full list of the SYSTEM 2000 user diagnostics alphabetically. The table of contents organizes the diagnostics by module of occurrence. If the diagnostic message does not appear sufficiently clear by itself, a descriptive paragraph is added. The understanding of each is greatly enhanced when they are presented in context with their actual occurrence. The modules which can generate each diagnostic message are also indicated.

If the message is not considered an error message, the word INFORMATIVE is shown. Each message is concluded with appropriate code letters to indicate the system treatment of additional commands which may have been submitted along with the action creating the message, and the effect upon the data base, if any. The code letters and their meaning are as follows:

INFORMATIVE = Informative message only. Never fatal to the rest of the job. Never destructive to the data base.

F = Fatal to the rest of the job. Actions requested following the action creating the message are not processed.

PF = Potentially fatal to rest of the job, depending upon the user directives to stop.

NF = Non-fatal to rest of the job.

D = Destructive to the data base.

PD = Potentially and probably destructive to the data base.

ND = Non-destructive to the data base.

The additional information represented by the code letters are presented in the belief that the user may need to know the consequences of the error conditions.

Each diagnostic message may or may not be preceded by a hyphen (-) or a string of hyphens. Diagnostic messages which begin with a variable, indicated here by the symbols <xxx>, are listed in the X section.

TABLE OF CONTENTS

SYSTEM-WIDE

DEFINE

DEFINE CONTINUED

TABLE OF CONTENTS

LOADER

LOADER CONTINUED

TABLE OF CONTENTS

RETRIEVAL

RETRIEVAL CONTINUED

TABLE OF CONTENTS

UPDATE

UPDATE CONTINUED

UPDATE CONTINUED

SYSTEM 2000

Diagnostic Messages

A

A COMPONENT IN THE FUNCTION DEFINITION IS NOT TYPE NUMBER OR DATE

DEFINE ND, NF - new definition
ND, F - old definition

A COMPONENT MAY NOT PRECEDE ITS PARENT

DEFINE F, ND

A DATA BASE IS CURRENTLY LOADED

When an existing data base has been loaded or named for this job, a new data base cannot be defined.

DEFINE F, ND

A FULL TRACE IS REQUIRED IN REQUESTS OMITTING THE WHERE CLAUSE

The update request was not processed.

UPDATE F, ND

A NEW DATA BASE NAME HAS ALREADY BEEN DECLARED

More than one name has been specified for a new data base definition.

DEFINE F, ND

ABOVE VALUE REJECTED DUE TO ERROR IN NEXT LABEL

In the loader input string, if a data value and the succeeding component number are not separated by a blank or if there is a question as to whether the component number indeed belongs to the value (incorrect use of a separator) both the preceding and conflicting data are rejected. This prevents erroneous data from entering the data base if no STOP command was given.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

ACC - <xxx> <yyy>

If errors occur in the loader input string and the user specified a full or partial display of accepted values, then for each logical entry having errors, each accepted data value is displayed line by line, single space. <xxx> is the element number followed by the separator followed by the accepted data value.

LOADER	ND

ACC - DUMMY DATA SET FOR RG<xxx>

If errors occur in the loader input string and the user specified a display of accepted values, then for each logical entry having errors non-valued data sets that were accepted are displayed.

LOADER	ND

AND IS UNSATISFIED

In the WHERE clause a logical AND operation produced no results. Request is processed as usual.

RETRIEVAL and UPDATE	INFORMATIVE ND, NF - retrieval ND, PF - update if LIMITS are effective

ARCHIVE UPDATE TAPE HAS INVALID LABEL

When using KEEP or APPLY commands, the archive UPDATE TAPE must correspond exactly regarding tape and data base identification.

UPDATE	F, ND

AT <xxx> IS NON-NUMERIC OR OUT-OF-RANGE

In the WHERE clause an AT phrase contained an erroneous numeric value. <xxx> is the numeric value. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

AT <xxx> IS UNSATISFIED

In the WHERE clause an AT phrase produced no results. <xxx> is the number associated with AT. This request is processed as usual.

RETRIEVAL and	INFORMATIVE
UPDATE	ND, NF - retrieval
	ND, PF - update if LIMITS are effective.

B

BEFORE AND AFTER CLAUSE CAN ONLY BE USED WITH INSERT CURRENTLY

The update request was not processed.

UPDATE	F, ND

BINARY ZEROES EXIST ON DATA FILE

The DATA file consists of display code characters only; a binary zero (00) is an illegal display code character. This error usually occurs because of machine generated loader input string.

LOADER	F, ND

C

C <xxx> <yyy> -- DATE OCCURS BEFORE 10/15/1582

In the WHERE clause the data value for a type DATE element occurs before the advent of the Gregarian calendar and cannot be converted to number of days elapsed. This request is not processed. <xxx> is the component number and <yyy> is the date specified.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

C <xxx> EXISTS -- NOT SATISFIED

In the WHERE clause use of EXISTS produced no results for that condition. <xxx> is the element number in the condition. The request is processed as usual.

RETRIEVAL and	INFORMATIVE
UPDATE	ND, NF - retrieval
	ND, PF - update if LIMITS are effective

C <xxx> <yyy> -- HAS PROHIBITED DAY CODE

In the WHERE clause the data value for a type DATE element contains a day value not equal to 01 through 31. <xxx> is the element number and <yyy> is the relational operator and the date data value. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

C <xxx> <yyy> -- HAS PROHIBITED MONTH CODE

In the WHERE clause the data value for a type DATE element contains a month value not equal to 01 through 12. <xxx> is the element number and <yyy> is the relational operator and the date data value. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

C <xxx> <yyy> -- INCORRECT MONTH/DAY CODE COMBINATION

In the WHERE clause the data value for a type DATE element contains a day value not compatible with the specified month or 29 days were specified with a year value that is not a leap year. This request is not processed. <xxx> is the element number and <yyy> is the relational operator and the date data value. This request is not processed.

RETRIEVAL and UPDATE	ND, NF - retrieval ND, F - update

C <xxx> <yyy> -- <yyy> REFERS TO RG INSTEAD OF AN ELEMENT

In the WHERE clause the component associated with relational operators in any specified condition must be an element that can have data values associated with it in the data base. <xxx> is the number for the erroneous component and <yyy> is a relational operator such as EQ, LE, GT, GE, NE, LT, SPANS, EXISTS or FAILS. This request is not processed.

RETRIEVAL and UPDATE	ND, NF - retrieval ND, F - update

C <xxx> <yyy> -- UNSATISFIED CONDITION

In the WHERE clause the condition displayed produced no results. <xxx> is the element number and <yyy> is the relational operator and the first ten characters of the value. The request is processed.

RETRIEVAL and UPDATE	INFORMATIVE ND, NF - retrieval ND, PF - update if LIMITS are effective

C <xxx> <yyy> -- VALUE HAS PROHIBITED CHARACTER LENGTH

In the WHERE clause the data value has none or too many characters for the type of element related to the data value. <xxx> is element number and <yyy> gives the relational operator and first ten characters of the value. This request is not processed.

RETRIEVAL and UPDATE	ND, NF - retrieval ND, F - update

C <xxx> <yyy> -- VALUE IS NON-NUMERIC

In the WHERE clause the data value in a condition contains a non-numeric character(s) and is related to an element defined to be a DECIMAL, INTEGER, or EXPONENTIAL NUMBER. <xxx> is the element number and <yyy> gives the relational operator and the first ten characters of the value. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

C <xxx> <yyy> VALUE IS OUT-OF-RANGE

In the WHERE clause the data value in a condition contains a number that is out-of-range for the Control Data 6000 series computers. <xxx> is the element number and <yyy> gives the relational operator and the first ten characters of the value. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

CHANGES IMPLY RESTRUCTURING OF THE DATA SETS AND VALUES

DEFINE	F, ND

270 CHARACTERS SCANNED WITHOUT FINDING A SEPARATOR -- DISCARD FIRST 265 CHARACTERS

An update request containing this error in the data string is not processed.

UPDATE	F, ND

COMMAND CURRENTLY NOT OPERATIONAL

Deletion, changing elements to RGs, etc., not allowed.

DEFINE	F, ND

COMMAND TOO LONG OR TOO COMPLICATED

Parentheses in a nested Boolean go deeper than 64 levels or more than 30 or 40 conditions are contained in the WHERE clause; request should be broken up into several requests using SAME AND or SAME OR to obtain results. This request is not processed.

RETRIEVAL and UPDATE	ND, NF - retrieval ND, F - update

COMPONENT NUMBER GIVEN DOES NOT EXIST

A command was given indicating a number change or padding/null option change to an undefined component number.

DEFINE	F, ND

COMPONENT NUMBER NOT FOUND

The component number in a DESCRIBE command is not contained in the definition.

RETRIEVAL	ND, NF

COMPONENT NUMBER UNIDENTIFIED

An undefined component number was encountered in the loader data input string.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

COMPONENT NUMBERS WILL EXCEED THE MAXIMUM OF 9999

A RENUMBER command has been given with a starting number or an increment too large that would result in component numbers exceeding the 9999 maximum.

DEFINE	F, ND

COMPONENT TYPE NOT A REPEATING GROUP

A null option change was specified for a component number defined as an element, function or string.

DEFINE	F, ND

CONTROL CARD ERROR

For an on-site or 200 terminal batch job, if the Scope Control Card that calls SYSTEM 2000 contains a syntax error, e.g., a keypunch error, the job terminates giving this message. For remote jobs, the control cards are generated by the system and this error should not occur.

SYSTEM-WIDE	F, ND

COPY CREATED ...
<xxx> DEFINITION <yyy>, VERSION <zzz>, <ddd> <ttt>

When a data base has been successfully copied (caused by a CREATE COPY command), an informative message is given where:

<xxx> = data base name
<yyy> = definition version number
<zzz> = data base version
<ddd> = date when copied
<ttt> = time when copied

SYSTEM-WIDE	INFORMATIVE ND, NF

D

DATA BASE DEFINITION DOES NOT EXIST FOR LOADER MODULE

LOADER F, ND

DATA BASE HAS BEEN MODIFIED VIA INDIVIDUAL UPDATE REQUESTS

The user cannot apply more segments once the data base has been modified by individual requests such as CHANGE, ADD, etc.

UPDATE F, ND

DATA BASE NAME ALREADY USED - <data base name>

Generated when NEW DATA BASE <data base name>: command specifies an existing data base.

DEFINE F, D

DATA BASE NAME NOT ON DBN TABLE

Use of the DATA BASE NAME IS command has given the system a misspelled data base name or has specified a data base that has not been loaded or created.

SYSTEM-WIDE F, ND

DATA BASE NAME NOT SPECIFIED YET

An informative message is given before the data base is named in the first job during a workday for all passwords. If the first command (after the password command) is not DATA BASE NAME IS <data base name>, then none of the modules can perform any serivce. Once the data base has been named, it is then automatically attached to subsequent jobs using that password. Each workday a data base must be named (or newly created), i.e., associated with the specified password.

SYSTEM-WIDE ND, NF

DATA BASE UNALTERED

No errors occurred, but an update request caused no action to be taken.

UPDATE	INFORMATIVE ND, NF

DATA BELOW NOT ACCEPTED FOR LOGICAL ENTRY <xxx>

If errors occur while scanning the loader input string, this heading appears before the list of errors for each logical entry having errors. <xxx> is the logical entry number.

LOADER	ND

DATA FILE EMPTY OR NOT REWOUND

The DATA file must be rewound, i.e., properly positioned at the beginning of a Section of loader input string, or the file will appear to be empty - or is empty.

LOADER	F, ND

DATA LABEL WITHOUT A DATA VALUE

In the loader data input string no data value was found after an element number; either the number was in error or the value was omitted. If an element has no value, the element number should be omitted.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

DATA SET <xxx> --- ELEMENT <yyy> BELONGS TO ANOTHER RG

In the data string of an update request an element number was miscoded or an RG was omitted. <yyy> is the element number. This request is not processed.

UPDATE F, ND

DATA SET <xxx> (RG <yyy>) HAS NO PARENT DATA SET

The hierarchy of RGs in the data string of an update request must be maintained. This request is not processed. <xxx> is the data set number and <yyy> is the RG number lacking a parent RG.

UPDATE F, ND

DATA SET <xxx>, NO VALUE FOR ELEMENT <yyy>

In data set number <xxx> in an update request, an element number appeared without an associated data value.

UPDATE F, ND

DITTO ILLEGAL DUE TO ERROR IN PREVIOUS REQUEST

1. In the RETRIEVAL module:

   a) The use of DITTO implies that a WHERE clause was given in the most previous request:

      PRINT C1:

      DITTO WHERE ....: (illegal use)

   b) If any error occurred to the left of the WHERE clause in a retrieval request, subsequent use of DITTO is illegal until an error-free PRINT clause, for instance, is encountered.

c) DITTO has no meaning across the RETRIEVAL and UPDATE modules, thus, the following sequence is illegal:

```
RETRIEVAL:
PRINT C1 WHERE C1 EXISTS:
UPDATE:
DITTO WHERE SAME:
```

2. In the UPDATE module:

a) If any errors occurred to the left of the WHERE clause in an update request, then use of DITTO is illegal on the next request.

b) DITTO has no meaning across the RETRIEVAL and UPDATE modules, thus, the following sequence is illegal:

```
UPDATE:
CHANGE C1 EQ .... WHERE ....:
RETRIEVAL:
DITTO WHERE SAME:
```

c) (Unlike RETRIEVAL, DITTO may be used legally in an update request when the previous update request had no WHERE clause.)

RETRIEVAL and UPDATE	ND, NF - job in RETRIEVAL ND, F - job in UPDATE

DUPLICATE COMPONENT NAMES

DEFINE	ND, NF - new definition ND, F - old definition

DUPLICATE COMPONENT NUMBERS

DEFINE	ND, NF - new definition ND, F - old definition

E

ELEMENT HAS A DATA VALUE IN THIS DATA SET

In the loader input string two values were assigned to one element in a single data set. Cause of error may be a missing RG number, an erroneous element number, a missing entry terminator at the end of a logical entry, or two actual values incorrectly assigned to the same element.

LOADER PF - depending on effective STOP command
ND - but hazardous depending on effective STOP command

ELEMENT NOT A MEMBER OF THIS RG

Element numbers are tested to see if each belongs to the last RG number in the loader input string (except level 0 elements). The cause of error may be a missing RG, an erroneous RG number, elements for one data set were scattered inadvertantly, or the element number was erroneous.

LOADER PF - depending on effective STOP command
ND - but hazardous depending on effective STOP command

END-OF-FILE AFTER READING <xxx> SEGMENTS ... NO SEGMENTS APPLIED

While trying to process an APPLY <n> SEGMENTS command, an end-of-file mark on the UPDATE FILE was encountered prematurely before finding <n> segments. <xxx> is a count of the number of segments that were read.

UPDATE F, ND

ENTIRE ENTRY DISPLAY

If any errors occur when scanning the loader input string, the user display option is given as a heading to the list of errors.

LOADER	ND

ERROR IN DATE OR NUMERIC VALUE

Numeric data value error or date error in the data string of any update request.

UPDATE	F, ND

EXTRANEOUS VALUE FOR ELEMENT <xxx> IN DATA SET <yyy>

In the data string of an update request each element may take on only one value per data set. <xxx> is the redundant element number and <yyy> is the data set number. This request is not processed.

UPDATE	F, ND

F

(There are no diagnostics under this category at the present time.)

G

GO TO RETRIEVAL FOR DESCRIBE

DESCRIBE requests are honored by the RETRIEVAL module only.

DEFINE ND, NF

H

(There are no diagnostics under this category at the present time.)

I

ILLEGAL AMOUNT (OVER 60 PERCENT) OF PADDING

DEFINE	ND, NF - new definition ND, F - old definition

ILLEGAL CHOICE OF SEPARATOR OR TOO MANY CHARS

The standard separator symbol is *. If the user changes the separator symbol, it may not be an alphanumeric character A - Z or 0 - 9; it may not be a ",", a ".", a ":", or a blank; it must be a single character.

SYSTEM-WIDE	F, ND

ILLEGAL DATE DATA VALUE

In the loader input string a date value contained a syntax error, month or day code was illegal, month/day combination was illegal, or date was before 10/15/1582.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

ILLEGAL KEYWORD AFTER A SPECIAL LABEL ERROR 9 REJECTIONS = <xxx>

In the loader data input string the ** must be followed by the current <entry terminator>, COMMENT, or SEPARATOR IS. Error 9 is known to LOADER and <xxx> is the total number of rejections encountered.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

ILLEGAL LIMITS -- ONLY NON-NEGATIVE INTEGERS ARE PERMITTED

UPDATE	F, ND

ILLEGAL NUMERIC DATA VALUE

In the loader data input string, a data value for a numeric type element contained embedded blanks, non-numerals, incorrect decimal point or omitted decimal point, or incorrect syntax of an exponential number or the element number is incorrect.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

ILLEGAL OPERATION - DIVISION BY ZERO

While calculating a user-defined function in a retrieval request, the data values caused division by zero.

RETRIEVAL	ND, NF

ILLEGAL OPERATION - NEGATIVE NUMBER TO A REAL POWER

While calculating a user defined function in a retrieval request, a negative data value raised to a fractional power such as square root $(-9^{.5})$ could not be calculated.

RETRIEVAL	ND, NF

ILLEGAL PASSWORD

An illegal password has been specified on a remote batch job or on a USER command under batch mode, also generated when INVALID PASSWORD IS <password>: command specifies a non-existent password.

SYSTEM-WIDE	F, ND

ILLEGAL TAPE NUMBER

The DATA, COMMAND, REPORT, and MESSAGE file names may not begin with "TAPE" and must not have more than 7 characters; they must begin with an alphabetic character and contain no special characters.

SYSTEM-WIDE	F, ND

ILLEGAL VALUE STRING IN REMOVE REQUEST

An update REMOVE command should not have a data value string.

UPDATE	F, ND

INCLUSION OF WHERE CLAUSE CONFLICTS WITH USE OF FULL TRACE

The update request was not processed.

UPDATE	F, ND

INCORRECT ARITHMETIC EXPRESSION

Arithmetic expression is not syntactically proper in a function definition.

DEFINE	ND, NF - new definition ND, F - old definition

INCORRECT DATA BASE NAME

When redefining an existing definition, the name of the old data base does not equal the data base currently loaded or in use.

DEFINE	F, ND

INCORRECT NUMBER OF OPERANDS FOR OPERATOR

The EQ, GE, GT, LT, LE, NE operators must have one operand. FAILS and EXISTS do not have an operand. SPANS takes two operands separated by a comma.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

INDETERMINATE VALUE OR COMPONENT NUMBER

Due to arbitrary syntax, perhaps use of unbalanced parentheses or incorrect separator symbol, a data value or component number is questionable.

DEFINE	NF - job for new definition or retrieval
	F - job for all else

J

(There are no diagnostics under this category at the present time.)

K

(There are no diagnostics under this category at the present time.)

L

LARGE RETRIEVAL, NEED ADDITIONAL FIELD LENGTH. <xxx> WORDS MORE TO RUN

In the WHERE clause the number of qualified data sets requires more field length to process entire results. <xxx> equals the number of qualified data sets that could not be included in the WHERE clause results. Request is processed as usual but those <xxx> data sets are not included in the output results. Rather than increase field length for remote batch jobs, the request perhaps should be splintered into smaller sections of WHERE clause conditional results.

RETRIEVAL and UPDATE	INFORMATIVE NF - retrieval F - update if LIMITS are effective PD - update if LIMITS are effective

LAST PREVIOUS REQUEST IN ERROR OR NO PREVIOUS REQUEST

Use of DITTO requires the appropriate clause in the last previous request.

UPDATE	F, ND

LEVEL 0 DISPLAY OPTION

If any errors occur when scanning the loader input string, the user display option is given as a heading to the list of errors.

LOADER	ND

LIMITS ARE <xxx> AND <yyy>

The system echos an informative message when it honors a LIMIT command.

UPDATE	INFORMATIVE ND, NF

LOADED ...

<xxx>, DEFINITION <yyy>, VERSION <zzz>, <ddd> <ttt>

When a data base has been successfully loaded from magnetic tape to disk, an informative message is given where:

<xxx> = data base name
<yyy> = definition version number
<zzz> = data base version
<ddd> = date when loaded
<ttt> = time when loaded

SYSTEM-WIDE	INFORMATIVE ND, NF

LOADER STOPPED AFTER <n> ERRORS

LOADER found <n> errors and stopped scanning the data input string as specified by the user; no data values were entered into the data base.

LOADER	ND, F

LOADER STOPPED AFTER <n> EXCLUDED VALUES

LOADER stopped after encountering 1000 excluded data values in the data input string; <n> is not a user option currently; no data values were entered into the data base.

LOADER	INFORMATIVE ND, F

LOADER WAS DIRECTED TO STOP AFTER SCANNING INPUT STRING

LOADER scanned the entire data input string for errors and stopped before entering the data values into the data base.

LOADER	ND, F

LOGICAL ENTRY <xxx>. <yyy> VALUES ACCEPTED

For each logical entry having errors in the loader data input string, regardless of the specified display option, an informative message gives the logical entry number <xxx> and the total number of data values accepted, <yyy>, for that logical entry.

LOADER ND

M

MAP COMMAND MUST BE USED FOR A NEW DEFINITION

A REMAP command can only be used when redefining an existing definition.

DEFINE	F, ND

MORE THAN 255 CHARACTERS IN A DATA VALUE

Data values cannot have more than 255 characters. While scanning the loader data input string, the current separator symbol was encountered too far beyond the 255th character of the data value.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

MORE THAN 127 COMPONENTS - MAXIMUM UNDER THIS VERSION

DEFINE	F, ND

MORE THAN 64 REPEATING GROUP LEVELS - MAXIMUM REACHED

DEFINE	F, ND

N

NEW SEPARATOR IS ILLEGAL OR NON-EXISTENT ERROR 10 REJECTIONS = <xxx>

An illegal separator symbol was specified within the loader input string. Error 10 is known to LOADER and <xxx> equals the total number of rejections at that point in the scanning process.

LOADER F, ND

NO CHANGES HAVE BEEN MADE TO THE DEFINITION

A REMAP command has been given and no redefining changes were specified.

DEFINE F, ND

NO DATA ACCEPTED FOR LOGICAL ENTRY <xxx>

If errors occur while scanning the loader input string, then for each logical entry having errors and no accepted data values, this informative message is given.

LOADER ND

NO DATA ACCEPTED FOR THIS SESSION

The entire loader data input string was rejected or excluded due to errors in the string.

LOADER F, ND

NO DATA BASE DECLARATION EXISTS

A MAP command has been issued and the new definition contains no component descriptions.

DEFINE F, ND

NO DATA BASE EXISTS FOR RETRIEVALS

RETRIEVAL	F, ND

NO DATA BASE LOADED FOR UPDATE REQUESTS

No data base was loaded or created or named before calling the UPDATE module or a definition exists without any data values. LOADER must be called at least once to create a minimum of one logical entry before using the UPDATE module.

UPDATE	F, ND

NO DATA BASE NAMED

A data base name does not exist for the DEFINE module to define or redefine. One of the following commands must be given:

NEW DATA BASE <data base name>:
OLD DATA BASE <data base name>:

DEFINE	F, ND

NO ENTRY TERMINATORS BEFORE EOF REJECTIONS = <xxx>

No entry terminators were encountered before an end-of-file indicator. <xxx> equals total number of rejections to that point in the scan of the loader input string.

LOADER	NF - but hazardous if EOF in error PD - depending on effective STOP command

NO PRECEDING PARENT RG DATA SET

In the loader data input string each RG number must be preceded by its parent RG back to the level 0 ancestral RG. The parent RG need not immediately precede siblings; this depends on the structure of the data tree being created.

LOADER PF - depending on effective STOP command
ND - but hazardous depending on effective STOP command

NO PRIOR VALUE STRING OR PREVIOUS VALUE HAD ERRORS

UPDATE F, ND

NO OUTPUT FOUND

No output was found for a retrieval request.

RETRIEVAL INFORMATIVE
ND, NF

NO RESTRUCTURING MODIFICATIONS WERE MADE

A REMAP command was given to an existing definition but the changes did not imply restructuring.

DEFINE F, ND

NO RESTRUCTURING NECESSARY ON A NEW DATA BASE

A REMAP command was given to finalize a new definition; the MAP command is sufficient.

DEFINE F, ND

NO USER SPECIFICATION

For on-site or 200 terminal jobs, the first SYSTEM 2000 command in every job must be the:

```
USER, <xxx>, <yyy>:
```

command where <xxx> is the user password and <yyy> is the user account number. The USER command gives legal passwords access to SYSTEM 2000; without the command the job terminates with issuance of the error message. For remote job submissions, the user must declare his password and account number once at login time to gain access to SYSTEM 2000; from login to logout time, SYSTEM 2000 is automatically available to the terminal that gave the legal password and this error will occur.

SYSTEM-WIDE	F, ND

NOT OPERATOR EXCLUDED ENTIRE DATA BASE

Request is processed as usual.

RETRIEVAL and	INFORMATIVE
UPDATE	ND, NF - retrieval
	ND, PF - update if LIMITS are effective

NOT OPERATOR QUALIFIED ENTIRE DATA BASE

Request is processed as usual.

RETRIEVAL and	INFORMATIVE
UPDATE	ND, NF - retrieval
	ND, PF - update if LIMITS are effective

NULLS APPLY ONLY TO REPEATING GROUPS

DEFINE	ND, NF - new definition
	ND, F - old definition

NUMBER OF SELECTED SETS IS ABOVE USER SELECTED UPPER LIMIT

The LIMIT command currently in effect stopped action from taking place for the last displayed update request.

UPDATE	F, ND

NUMBER OF SELECTED SETS IS BELOW USER SELECTED LOWER LIMIT

The LIMIT command currently in effect stopped action from taking place for the last displayed update request.

UPDATE	F, ND

O

OBJECT COMPONENT CANNOT BE AN ELEMENT THIS REQUEST TYPE

The component specified before EQ in an update request involving TREE processing must be a repeating group or ENTRY.

UPDATE	F, ND

OBJECT COMPONENT MUST BE IDENTICAL TO PREVIOUS OBJECT COMPONENT

When using PREVIOUS in an update request, the RG or element specified before EQ must be identical to that of previous request.

UPDATE	F, ND

ONLY ONE ENTRY TERMINATOR BEFORE EOF REJECTIONS = <xxx>

Two entry terminators before an end-of-file indicator signals the end of data for the loader input string. <xxx> is the total number of rejections to that point in the scanning process.

LOADER	NF - but hazardous PD - depending on effective STOP command

OR IS UNSATISFIED

In the WHERE clause a logical OR operation produced no results. Request processed as usual.

RETRIEVAL and UPDATE	INFORMATIVE ND, NF - retrieval ND, PF - update if LIMITS are effective

ORING OF A NOT (CONDITION) HAS SPECIFIED ENTIRE DATA BASE

In the WHERE clause a NOT condition combined with an OR operation qualified the entire data base. Request is processed as usual.

RETRIEVAL and	INFORMATIVE
UPDATE	ND, NF - retrieval
	ND, F - update if LIMITS are effective

P

PADDING INVALID FOR A REPEATING GROUP

DEFINE	ND, NF - new definition ND, F - old definition

PASSWORD <xxx> NOT AUTHORIZED TO USE <yyy>

The data base security check has found that password <xxx> is not authorized to use data base <yyy>.

SYSTEM-WIDE	F, ND

PASSWORD NOT VALID FOR DATA BASE

Generated when INVALID PASSWORD IS <password>: command specifies a password which is not valid (legal) for the accessed data base.

DEFINE	ND, NF

PRESERVED ...
<xxx>, DEFINITION <yyy>, VERSION <zzz>, <ddd> <ttt>

When the LOADER module successfully completes the creation of a new data base, the new data base is preserved on disk for future use and the new data base name is entered automatically into the data base name table.

<xxx> = data base name
<yyy> = definition version number
<zzz> = data base version number
<ddd> = date of creation
<ttt> = time of creation

SYSTEM-WIDE	INFORMATIVE ND, NF

Q

(There are no diagnostics under this category at the present time.)

R

-REJ- <separator> <separator> <yyy>

If a special label (i.e., **) is incorrect or is followed by an unrecognizable word in the loader input string, the non-data item is rejected and displayed. If the rejection was a ** COMMENT, no damage will occur.

LOADER
PF - depending on use and effective STOP command
ND - but hazardous depending on use and effective STOP command

-REJ- <xxx> <yyy>

If errors occur in the loader input string, each erroneous item is displayed with -REJ- followed by <xxx>, the component number and <yyy>, the data value. A specific error message is given on the line following the rejected item.

LOADER
PF - depending on effective STOP command
ND - but hazardous depending of effective STOP command

RELEASED ...
<xxx>, DEFINITION <yyy>, VERSION <zzz>, <ddd> <ttt>

When a RELEASE command has successfully released a data base, an informative message is given where:

<xxx> = data base name
<yyy> = definition version number
<zzz> = data base version
<ddd> = date of release
<ttt> = time of release

SYSTEM-WIDE
INFORMATIVE
ND, NF

RELOAD CURRENTLY INOPERATIVE

RETRIEVAL ND, NF

REPEATING GROUP DOES NOT EXIST

A component was described as being IN a component number defined to be an element, function or string.

DEFINE ND, NF - new definition
ND, F - old definition

REQUEST REJECTED

The last displayed update request was rejected because of errors.

UPDATE F, ND

REQUIRED VALUE STRING MISSING

A data value string or PREVIOUS must be supplied for all CHANGE, ADD, INSERT and ASSIGN requests.

UPDATE F, ND

RESULT OF WHERE CLAUSE IS A NOTTED LIST ... UPDATE CANNOT HANDLE IT AT PRESENT

UPDATE F, ND

RESULT OF WHERE CLAUSE IS WHOLE SHEBANG ... UPDATE CANNOT HANDLE IT AT PRESENT

UPDATE F, ND

S

SAME ILLEGAL DUE TO ERROR IN PREVIOUS REQUEST

In the RETRIEVAL and UPDATE modules:

An error occurred in the WHERE clause of the previous request.

RETRIEVAL and UPDATE	ND, F - job in UPDATE module ND, NF - job in RETRIEVAL module

SAVED ... <xxx>, DEFINITION <yyy>, VERSION <zzz>, <ddd> <ttt>

After a data base has been successfully saved on magnetic tape, an informative message is given where:

<xxx> = data base name
<yyy> = definition version number
<zzz> = data base version
<ddd> = date when saved
<ttt> = time when saved

SYSTEM-WIDE	INFORMATIVE ND, NF

SCANNER FOUND AN ERROR WHILE ASSUMING NO ERRORS, TRY AGAIN WITH ASSUME ERRORS

Certain types of errors are recognized even though the LOADER module may have been told to ASSUME NO ERRORS. However, most error conditions that may happen in the loader input string are not tested while ASSUME NO ERRORS is in effect. User should assure an error-free loader string by pre-editing before letting LOADER enter the values into the data base.

LOADER	F, ND

SECOND COMPONENT NUMBER APPEARS BEFORE THE FIRST COMPONENT NUMBER

In a DESCRIBE command

DESCRIBE C<xxx> THRU C<yyy>

<xxx> must appear before <yyy> in the definition regardless of the magnitude of the user component numbers. This request is not processed.

RETRIEVAL	ND, NF

SECOND <separator symbol> NOT FOUND AFTER 50 CHARACTERS

In a retrieval request a string name was not bounded by the current separator symbol. String names may have a maximum of 50 characters. This request is not processed.

RETRIEVAL	ND, NF

SEGMENT COUNT MUST BE POSITIVE INTEGER

When specifying explicitly the number of update segments to be kept or applied, the number must be a positive integer.

UPDATE	F, ND

SPANS A, B REQUIRES A LE B

When using SPANS in the WHERE clause, A must be less than or equal to B.

RETRIEVAL and UPDATE	ND, NF - retrieval ND, F - update

SPECIAL LABEL WITHOUT A VALUE, ERROR 8 REJECTIONS = <xxx>

If, for instance, ** is encountered in the loader string and is immediately followed by a component number, then an entry terminator or a COMMENT has been omitted, or the special label is erroneous. Error 8 is known to LOADER and rejections = <yyy> gives current total number of rejections.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

STRING DEFINITION GREATER THAN 700 CHARACTERS

DEFINE	F, ND

SYNTACTIC ERROR IN APPLY OR KEEP

UPDATE	F, ND

SYNTACTIC ERROR IN COMPONENT DESCRIPTION

Type of component misspelled, no type specified, no separator after component number when defining, or a SYSTEM 200 command word or symbol occurs in a component name.

DEFINE	ND, NF - new definition ND, F - old definition

SYNTACTIC ERROR IN DATA BASE NAME

DEFINE	F, ND

SYNTACTIC ERROR IN PADDING OPTION

DEFINE	ND, NF - new definition ND, F - old definition

SYNTACTIC ERROR IN TRACE

UPDATE	F, ND

SYNTAX ERROR IN COMMAND

The command has been recognized as a type of request that is legal at the time, but the syntax of the command is illegal; for instance,

```
INSERT TREC ....:
ASSUME NOT ERRORS:
UNLOAD X WHERE ....:
PRINT .... WHERE SAME C1 ....:
```

(SAME must be followed by AND or OR or :)

SYSTEM-WIDE	ND, NF - new definitions or retrievals ND, F - all else

SYNTAX ERROR IN COMPONENT NUMBER OR MISSING REQUIRED BLANK

In the data string of an update request, a syntax error occurred.

UPDATE	F, ND

SYNTAX ERROR IN FUNCTION DEFINITION

DEFINE	ND, NF - new definition ND, F - old definition

SYNTAX ERROR IN LABEL

In the loader data input string:

1. No numerals (component number) found before a single separator symbol. This may be because of an omission or because the separator symbol occurred within a data value.

2. Component number contains a non-numeric character (same possible causes as 1.)

3. Component number more than 4 numerals; ambiguity between a data value and a component number or a blank was omitted or the separator is being used incorrectly.

4. No blank was found before the component number.

5. No blank before an ** <entry terminator> or ** COMMENT.

6. No recognizable word after **.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

```
SYSTEM 2000, VERSION <xxx>:
DATA BASE NAME IS <       >
DEFINITION VERSION NUMBER:   <xxx>
DATA BASE VERSION NUMBER:    <xxx>
```

These four lines appear in the heading with the display of full DESCRIBE, DESCRIBE FUNCTIONS and DESCRIBE STRINGS commands.

RETRIEVAL	INFORMATIVE ND, NF

SYSTEM ERROR CODE NO. xxx

A system error message is the result of a malfunction in the system and does not involve the user's commands. System errors are coded with numbers that are meaningful to programmers working on SYSTEM 2000. All system errors are fatal to the rest of the job. Most of the system errors destroy or partially damage the data base as it exists on the disk permanent files. A RELEASE command can be given to release the damaged data base and then the data base may be restored by a LOAD command.

The following system errors will not destroy any data base. All numbers not listed below are destructive.

NON-DESTRUCTIVE SYSTEM ERROR CODES

1
5
7
9
10
11
12
14
15
16
17
18
19
20
21
22
23
24
104
105
106
212
213
214
215
216
217
218
219
303
304
305
307 (if KEEP was used)

SYSTEM-WIDE F
D - except for list shown above

T

THE DATA BASE HAS NOT BEEN MAPPED OR REMAPPED

If the DEFINE module has been called upon to define a new data base or to modify an existing definition and the user attempts to call another module, such as RETRIEVAL, without giving a MAP or a REMAP command, the job terminates with this message. Inadvertantly forgetting to finalize a new definition or changes to a definition causes the new or changed components to be unavailable for use.

DEFINE	F - job always ND

TOO MANY BRANCHES SPECIFIED IN TRACE

The update request was not processed.

UPDATE	F, ND

TOO MANY SETS DEFINED IN VALUE STRING

Only one data set can exist in the data string of an update request unless the request specifies a TREE operation.

UPDATE	F, ND

TOO MANY VALUES IN VALUE STRING

If an element is specified before EQ in an update request, only one value can be given in the data string.

UPDATE	F, ND

TRACE CANNOT BE USED IN CONJUNCTION WITH BEFORE OR AFTER CLAUSE

The update request was not processed.

UPDATE F, ND

TRACE REQUIRED WITH INSERT ... WHERE

UPDATE F, ND

U

UNBALANCED PARENTHESES

Parentheses in a function definition must be balanced.

DEFINE	ND, NF - new definition ND, F - old definition

UNDEFINED COMPONENT/FUNCTION <xxx> USED IN COMMAND

Component given in a retrieval request does not match any of those contained in the definition. This request will not be processed.

RETRIEVAL	ND, NF

UNDEFINED COMPONENT NAME <xxx>

A component name given in an update request does not match any of the component names in the definition. This request will not be processed.

UPDATE	F, ND

UNDEFINED COMPONENT NUMBER <xxx>

A component number given in an update request does not match any component number in the definition. This request will not be processed.

UPDATE	F, ND

UNDEFINED OPERAND USED IN FUNCTION DEFINITION

The definition of a function contains a component number that does not appear in the definition.

DEFINE ND, NF - new definition
ND, F - old definition

UNDEFINED STRING USED IN COMMAND

A string name given in a retrieval request does not exist in the current definition. This request is not processed.

RETRIEVAL ND, NF

UNEXPECTED END-OF-FILE ON ARCHIVE TAPE

While processing KEEP or APPLY commands, an end-of-file mark was encountered before the expected number of segments were processed.

UPDATE F, ND

UNRECOGNIZABLE COMMAND

An illegal first word(s) in a request causes an unrecognizable command. The erroneous first word may have been misspelled or may have been illegal for the task module currently in service, such as giving a PRINT request to the UPDATE module.

SYSTEM-WIDE ND, NF - new definition and retrieval
ND, F - all else

UNRECOGNIZED OBJECT COMPONENT

The component name or number specified for update action (before EQ or component in REMOVE commands) does not exist in the definition.

UPDATE F, ND

UPDATE FILE AUGMENTED

The system returns an informative message indicating that another segment has been added to the UPDATE FILE each time a TERMINATE or an automatic terminate has taken place provided that the UPDATE FILE has not been SUSPENDED.

UPDATE INFORMATIVE
ND, NF

UPDATE FILE HAS BEEN SUSPENDED

A KEEP or APPLY command has been given for a suspended UPDATE FILE or the UPDATE FILE has never been declared.

UPDATE F, ND

UPDATE FILE HAS ONLY <xxx> SEGMENTS

If a KEEP <n> SEGMENTS command is given and <n> is greater than the total number of segments that have been created, an error exists and none of the segments are kept. <xxx> is the number of segments that have been created on the UPDATE FILE.

UPDATE F, ND

UPDATING COMPLETE ... CURRENT VERSION IS <xxx> <yyy> <zzz>

When a TERMINATE (or an automatic terminate) or an APPLY action has been completed, the system returns an informative message indicating:

1. The current version number, <xxx>
2. The current date, <yyy>
3. The current time, <zzz>

UPDATE INFORMATIVE
ND, NF

UPPER LIMIT LOWER THAN LOWER LIMIT

Parameters in the LIMIT command must be given in ascending order or be equal.

UPDATE F, ND

USER SHOULD REMAP CHANGES TO OLD DATA BASE

The MAP command finalizes a new definition; the REMAP command should be used to modify an existing definition.

DEFINE F, ND

V

VALUE EXCEEDS 255 CHARACTERS

A data value was too long in the data string of an update request. This could be caused by an incorrect separator symbol.

UPDATE	F, ND

VALUE GIVEN AFTER A DATA SET LABEL

In the loader data input string a repeating group number was followed by a data value not another component number. Either the RG number is in error or an element number has been omitted.

LOADER	PF - depending on effective STOP command ND - but hazardous depending on effective STOP command

VALUE GIVEN AFTER RG IDENTIFIER

Error in data string of an update request.

UPDATE	F, ND

W

WHEN USING C<xxx> HAS C<yyy>, - C<xxx> MUST BE A SENIOR RG TO C<yyy>

In the WHERE clause the repeating group specified with HAS must be a parent or ancestor RG to the element given in the condition. (ENTRY is an implied ancestral RG to all elements in the definition.) <xxx> is the repeating group number for HAS and <yyy> is the element number in the condition. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

WHEN USING C<xxx> HAS C<yyy>, C<xxx> MUST BE AN RG

In the WHERE clause HAS takes a repeating group component or ENTRY operand. <xxx> is the HAS operand number and <yyy> is the element number associated with the HAS condition. This request is not processed.

RETRIEVAL and	ND, NF - retrieval
UPDATE	ND, F - update

WHERE CLAUSE QUALIFIED ENTIRE DATA BASE

Request is processed as usual.

RETRIEVAL and	INFORMATIVE
UPDATE	ND, NF - retrieval
	ND, F - update if LIMITS are effective

WHERE CLAUSE STATUS MUST BE CONSISTENT WITH THAT OF PREVIOUS REQUEST

When using DITTO:

1. Previous request must have WHERE clause if DITTO uses WHERE.

2. Previous request must have AFTER or BEFORE clause to use DITTO with AFTER or BEFORE.

UPDATE	F, ND

X

<xxx> DEFINITION VERSION <yyy> DATA VERSION <zzz> <ttt> <ddd>

LOADER informative message displays data base name <xxx> and both version numbers <yyy> and <zzz> after loading is complete; message is given whether or not checkpoint reports were specified. <ttt> is time of day and <ddd> is the date.

LOADER	INFORMATIVE ND, NF

<xxx> ERROR <yyy> REJECTIONS = <zzz>

If errors occur while scanning the loader input string, the appropriate error message, <xxx>, is displayed below the rejected item along with an error number, <yyy>, known to LOADER, and a consecutive total number of rejected values up to that point in the scanning process.

LOADER	ND, PF - depending on effective STOP command

<xxx> EXCLUDED VALUES IN LOGICAL ENTRY <yyy>

Informative message given if errors occurred for a logical entry in the loader data input string; <xxx> is the total number of excluded values in logical entry <yyy>.

LOADER	ND

<xxx> HAS ALREADY BEEN LOADED

If a data base has already been loaded onto the disk, an informative message is given where <xxx> is the data base name.

SYSTEM-WIDE	INFORMATIVE ND, NF

<xxx> IS AN UNDEFINED COMPONENT NUMBER

Error is in the data string of an update request.

UPDATE	F, ND

<xxx> <yyy> -- REFERS TO RG INSTEAD OF AN ELEMENT

In a retrieval request AVG, SUM, MAX, MIN have no meaning when used with a repeating group. <xxx> is the arithmetic function and <yyy> is the component number.

RETRIEVAL	ND, NF

<xxx> REJECTED VALUES IN LOGICAL ENTRY <yyy>

Informative message given if errors occurred for a logical entry in the loader data input string; <xxx> is the total number of rejected values in logical entry number <yyy>.

LOADER	ND

<xxx> -- REQUIRES NUMERIC DATA VALUES FROM C <yyy>

In a retrieval request AVG, SUM, and SIGMA only have meaning for elements with numeric type data values.

RETRIEVAL	ND, NF

<xxx> <yyy> -- RESULTS WERE OUT-OF-RANGE

Where <xxx> is AVG, SUM, COUNT, or SIGMA and <yyy> is a component number.

RETRIEVAL	INFORMATIVE ND, NF

<xxx> SEGMENTS DISCARDED ... UPDATE FILE SUSPENDED

When a KEEP <n> SEGMENTS command is given and <n> is a number less than all of the segments, the user is informed of the number of segments, <xxx>, that were discarded. The UPDATE FILE is suspended and no further segments can be created.

UPDATE INFORMATIVE
ND, NF

<xxx> SEGMENTS WILL BE APPLIED

An informative message is returned to the user when an APPLY ALL command is given to the system.

UPDATE INFORMATIVE
ND, NF

<xxx> SEGMENTS WILL BE SAVED

When KEEP ALL command is given, an informative message is returned to the user showing how many segments, <xxx>, will be saved on the UPDATE FILE.

UPDATE INFORMATIVE
ND, NF

<xxx> SELECTED DATA SETS

<xxx> is the number of data sets selected for updating for a request.

UPDATE INFORMATIVE
ND, NF

<xxx> <yyy> -- UNSATISFIED - NO DATA VALUES FOR C<yyy>

Where <xxx> is AVG, SUM, SIGMA, MAX, or MIN and <yyy> is an element number.

RETRIEVAL	INFORMATIVE ND, NF

<xxx> VALUES ACCEPTED BUT NOT DISPLAYED FOR LOGICAL ENTRY <yyy>

If the user specified a display of errors only while scanning the loader input string, then if any errors occur, this informative message is given to indicate the number of values, <xxx>, that were accepted but not displayed for each logical entry, <yyy>, that contained errors.

LOADER	ND

Y

(There are no diagnostics under this category at the present time.)

Z

(There are no diagnostics under this category at the present time.)

COMMENT SHEET

MANUAL TITLE SYSTEM 2000 PRELIMINARY USER INFORMATION MANUAL

PUBLICATION NO. D0028087002 REVISION July 1970

FROM:

NAME:

BUSINESS ADDRESS:

COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references.

CUT ALONG LINE

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.
FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
4550 WEST 77TH STREET
MINNEAPOLIS, MINNESOTA 55435

CUT ALONG LINE

ATTN: DATA SERVICES MARKETING STAFF
MARKETING SERVICES DEPARTMENT

FOLD

FOLD

REVISION DISTRIBUTION

From time to time
MRI will publish
revisions of certain
pages of this document
and less frequently,
new editions of the entire
document.
If you wish to receive
revisions of pages
and notification
of when a new
edition is about
to be published,
please complete
the card, detach,
and mail to MRI.

DOCS. ______________________
____
____

NAME______________________

TITLE______________________

ORGANIZATION______________________

ADDRESS______________________

CITY______________________

STATE______________________ZIP__________

PLACE
STAMP
HERE

DIRECTOR OF COMMUNICATIONS
MRI
2209 Hancock Drive
Austin, Texas 78756

CORPORATE HEADQUARTERS
8100 34TH AVENUE SOUTH
MINNEAPOLIS, MINNESOTA 55440

SALES OFFICES AND SERVICE CENTERS
IN MAJOR CITIES
THROUGHOUT THE WORLD

CONTROL DATA
CORPORATION