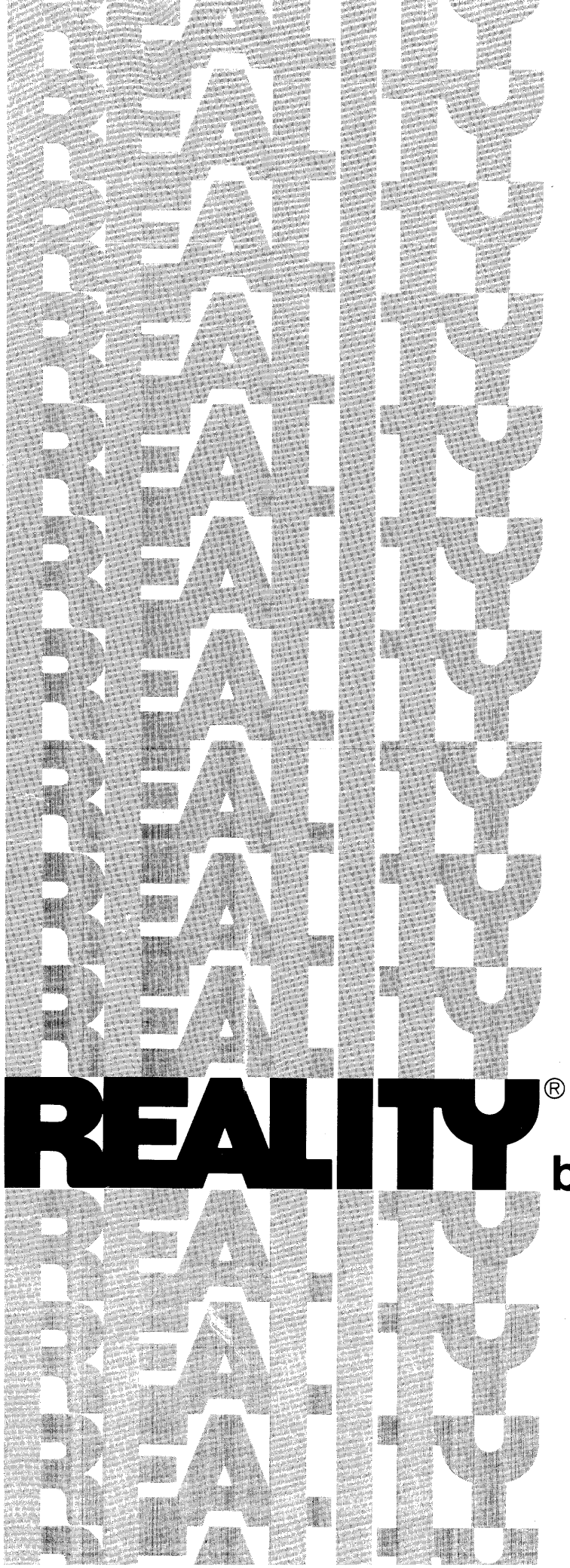

by Microdata

Introduction to Reality

REALITY®
(3.0 SERIES)

Introduction to Reality

771050

Price: $5.00

Microdata Corporation
17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764

CONTENTS

Section		Topic
1	Introduction	
	How to Use the Reality Manuals	1.1
	What is Reality?	1.2
2	The Reality Computer System	
	An Overview	2.1
	The Reality Software Processors	2.2
	The Reality File Hierarchy	2.3
	The Reality File Structures	2.4
	The Reality Dictionaries	2.5
3	The Terminal Control Language (TCL)	
	Logging On and Off the System	3.1
	Verbs and Processors	3.2
4	The Data Base Managment and Utility Processors	
	Data Base Management	4.1
	Utilities	4.2
5	The ENGLISH Processors	
	An Overview	5.1
	An ENGLISH Primer	5.2
	The ENGLISH Verbs	5.3
6	The DATA/BASIC Processor	
	An Overview	6.1
	DATA/BASIC Language Definition	6.2
	Creating, Compiling, and Executing DATA/BASIC Programs	6.3
7	The EDITOR Processor	
	An Overview	7.1
	EDITOR Language Definition	7.2
8	The PROC Processor	
	An Overview	8.1
	PROC Language Definition	8.2
9	The Reality CPU and Instruction Set	
	An Overview	9.1

1 INTRODUCTION

1.1 HOW TO USE THE REALITY MANUALS

The Reality manuals are written in modular format with each pair of facing pages presenting a single topic.

The approach taken in this and other Reality manuals differs substantially from the typical reference manual format. The left-hand page of each topic is devoted to text, while the right-hand page presents figures referred to by the text. At the head of each text page are a pair of titles, the first one naming the section and the second one naming the topic. Immediately below these titles is a brief summary of the material covered in the topic.

The advantage of this format will become readily apparent to the reader as he begins to use this manual. First of all, the figures referred to in the text are always conveniently right in front of the reader at the point where the fererence is made. Secondly, the reader knows that when he turns the page, he is done with one idea and ready to encounter a new one.

Additional documentation for the Reality system includes the following:

- The Reality Programmer's Reference Manual
- The Reality EDITOR Reference Manual
- The Reality ENGLISH Reference Manual
- The Reality DATA/BASIC Reference Manual
- The Reality PROC and BATCH Reference Manual
- The Reality Assembly Language Reference Manual

In presenting examples throughout this manual, certain conventions apply; these conventions are defined in Figure A.

CONVENTION	MEANING
TEXT	*Shaded text represents the user's input.*
TEXT	*Standard text represents computer output printed by the system.*
TEXT	*Italicized text is used for comments and notes which help explain or describe the example.*
(CR)	*This symbol represents a carriage return.*
∅	*This symbol represents a space (blank).*

Figure A. Conventions Used in Examples Throughout This Manual

1.2 WHAT IS REALITY?

Reality is a generalized, data base management computer system. It is a complete system that provides multiple users with the capability to instantly update and/or retrieve information stored in the on-line data files. Users communicate with the system through local or remote terminals to access files that may be private, common, or security-controlled. Each terminal user's vocabulary can be individually tailored to specific application jargon.

The Reality Computer System includes the powerful, yet simple to use ENGLISH[T.M.] inquiry language and the DATA/BASIC and PROC and BATCH high-level languages, file maintenance tools, an EDITOR, complete programming development facilities, and a host of other user amenities. Reality runs in an on-line, multi-user environment with all system resources and data files being efficiently managed by a microprogrammed Virtual Memory Operating System.

Reality is built on field proven Microdata computers and peripherals, utilizing microprograms to provide users with unrivaled performance and reliability in the medium-sized computer market.

Reality is uniquely different when measured from any angle: system capability, multi-user performance, file management languages, ease of programming, data structure, and architectural features. The high performance and fast response of Reality are possible only through the use of the high-speed microprocessor which greatly reduces system overhead and program execution time. The unique microprogrammed firmware contains the:

- Virtual memory manager
- Multi-user operating system
- Special data management instructions
- Input/output processors

The unique System Software includes:

ENGLISH, DATA/BASIC, PROC, TCL, and BATCH languages
Selectable/automatic report formatting
Dynamic file/memory management
Selectable levels of file/data security
Optional RPG II and BISYNC languages

The unique file structure provides:

- Variable length files/records/fields
- Multi-values (and subvalues) in a field
- Efficient storage utilization
- Fast accessibility to data items
- Selectable degrees of data security
- File size limited only by size of disc
- Record size up to 32K bytes

In summary, the Reality Computer System encompasses the following extraordinary features:

- True data base management
- Complete small business computer capabilities
- Microprogrammed virtual memory operating system
- Up to 32 users
- On-line file update/retrieval
- ENGLISH retrieval language
- Variable file/record/field lengths
- Dynamic file/memory management
- Automatic report formatting
- Total data/system security
- Fast terminal response
- Line printer spooling
- Special data management processors
- High-speed generalized sort
- Small computer price
- Big computer performance

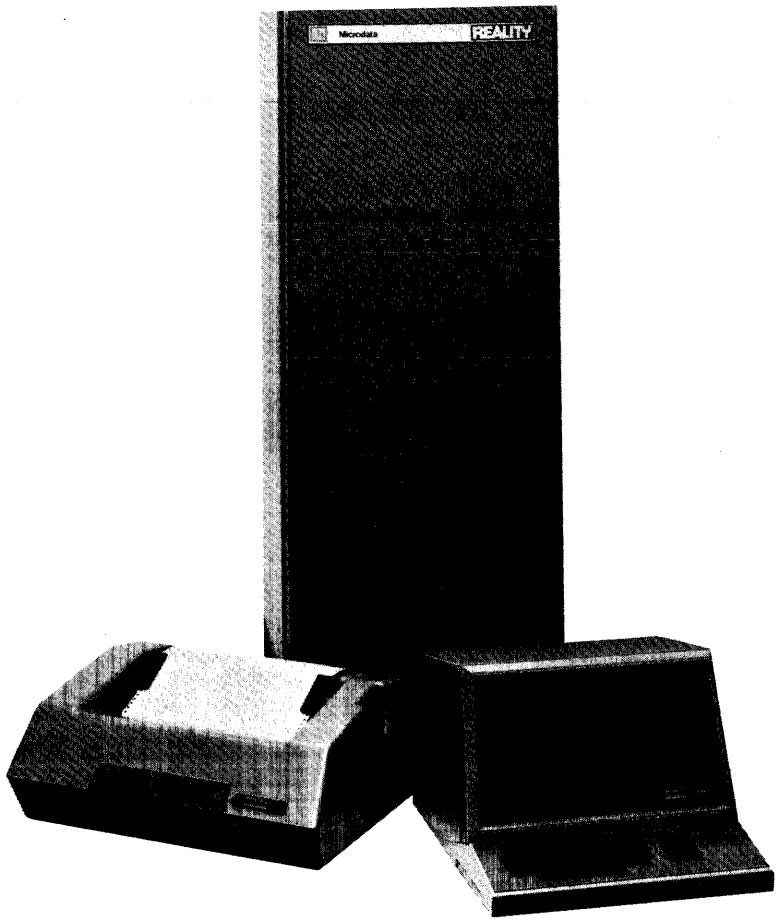

Figure A. Typical Microdata Reality System

2 THE REALITY COMPUTER SYSTEM

2.1 AN OVERVIEW

Reality is a completely new system of computer hardware, software, and firmware, specifically oriented to provide a vehicle for the implementation of cost-effective data base management. Data base management systems implemented in Reality afford two major benefits: 1) providing accurate and timely information to form the basis for significantly improving the decision-making process, and 2) substantially reducing the clerical and administrative effort associated with the collection, the storage, and dissemination of the information pertaining to an organization.

Reality is a completely new computer system combining both proprietary hardware and proprietary software to create an effective tool for on-line data base management. Through the use of microprogramming, Microdata has implemented a truly revolutionary on-line transaction processing system. Three major components of the system have been implemented directly in high-speed microprogrammed firmware:

- The virtual memory operating system
- The software level architecture
- The terminal input/output routines

The virtual memory operating system which has long been used in larger computer systems has been impractical for minicomputers due to the large amount of overhead needed for the operating system itself. In Reality, the virtual memory operating system has been directly implemented in firmware (i.e., highspeed read-only memory), which executes many times faster than would a comparable system normally implemented in software. Thus, the overhead time is reduced to negligible levels.

With the operating system directly implemented in read-only memory, only a small amount of main memory (core) is needed to run Reality. Slightly over 4,000 bytes of core are allocated for the operating system monitor. Everything else (system software, user software and data) is transferred automatically into main memory from the disc drive by the virtual memory operating system only when required.

Everything in the Reality Computer System is organized into 512-byte pages (frames) which are stored on the disc. As a frame is needed for processing, the operating system automatically determines if it is already in core memory. If it is not, the frame is automatically transferred from the disc unit (virtual memory) to core. Frames are written back onto the disc on a "least-recently-used" basis. This concept is illustrated in Figure A. The virtual memory feature of Reality allows the user to have access to a programming area not constrained by core memory, but as large as the entire available disc storage on the system.

The second feature implemented directly in firmware by Reality is the software level architecture of the machine itself. Through microprogramming, Microdata has implemented a machine architecture expressly designed and optimized for data base management. The architecture of Reality includes very powerful instructions expressly designed for character moves, searches, compares, and all supporting operations germane to managing variable length fields and records.

The third major feature implemented in microcode is the handling of Input/Output (I/O) communications with the on-line terminals. In any minicomputer on-line application, one of the main problems is that of managing the I/O from on-line interactive terminals. As these terminals increase in number, the load on the CPU becomes overwhelming and consequently the response to the terminals degrades dramatically. Microdata in Reality, has implemented the I/O processing of the on-line terminals in high-speed microcode. This means that other program execution need not be interrupted to handle characters coming or going to each and every terminal. The firmware handles all these transactions and only interrupts the software at the completion of a block. As a result, a very large number of terminals may be connected to the Microdata Reality System before any significant degradation in response time is detected.

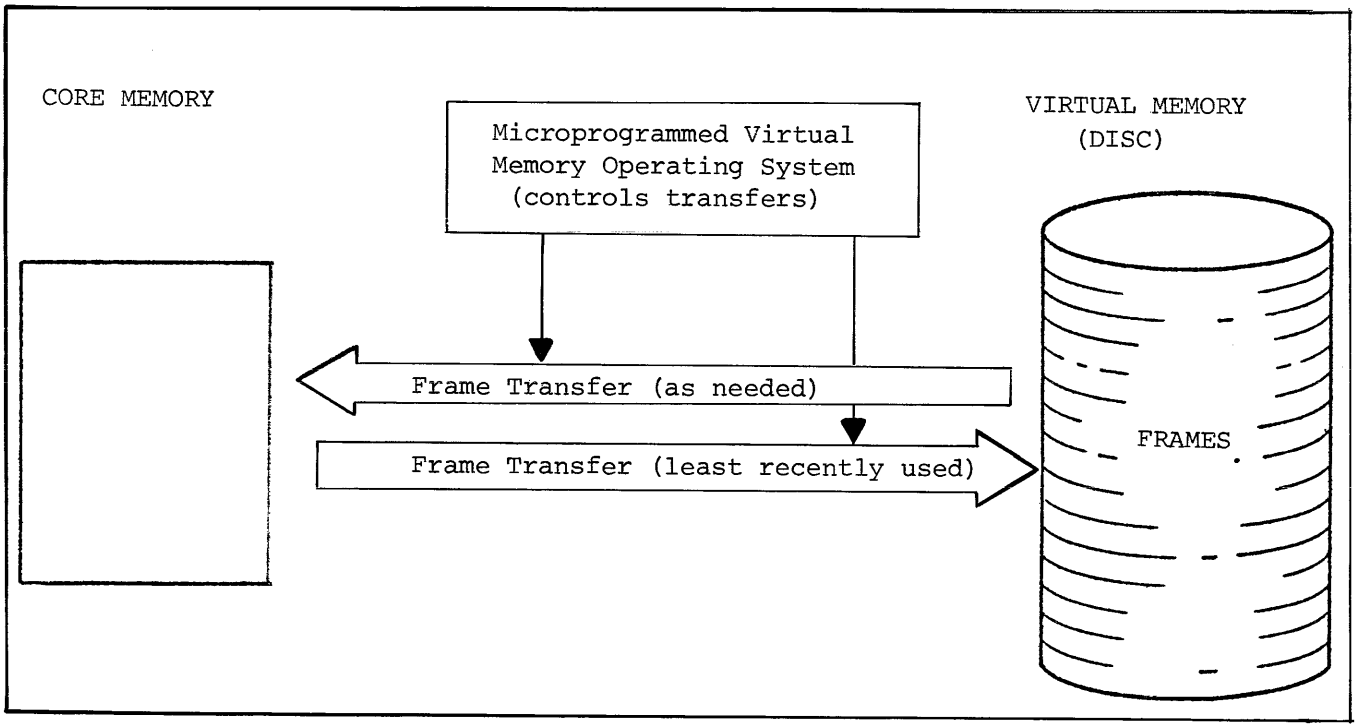

Figure A. Reality Frame Transfers

2.2 THE REALITY SOFTWARE PROCESSORS

The processors available on the Reality Computer System comprise the most extensive data base management software available on any minicomputer. An overview of some of the processors available to all terminal users is presented in this topic.

The ENGLISH Processor

ENGLISH is a generalized information management and data retrieval language. A typical ENGLISH inquiry consists of a relatively free-form sentence containing appropriate verbs, file names, data selection criteria, and control modifiers. ENGLISH is a dictionary-driven language. The vocabulary used in composing an ENGLISH input sentence is contained in several dictionaries. Each user's vocabulary, however, can be individually tailored to his particular application jargon. ENGLISH encompasses the following extended features:

- Freedom of word order and syntax for user inputs
- Automatic or user-specified output formatting
- Sorting capabilities plus generation of statistical information
- Relational and logical operations
- Verbs such as: LIST, SORT, SELECT, COUNT, STATISTICS, etc

ENGLISH is further described in the section titled THE ENGLISH PROCESSORS.

The DATA/BASIC Processor

BASIC (Beginners All-Purpose Symbolic Instruction Code) is a simple yet versatile programming language suitable for expressing a wide range of processing capabilities. BASIC is a language especially easy for the beginning programmer to master. DATA/BASIC is an extended version of BASIC which includes the following features:

- Flexibility in selecting meaningful variable names
- Complex and multi-line statements
- String handling with unlimited, varying length strings
- Integrated with Data Base file access and update capabilities

DATA/BASIC is described further in the section titled THE DATA/BASIC PROCESSOR.

The PROC Processor

The PROC processor allows the user to prestore a complex sequence of operations which can then be evoked by a single word command. Any sequence of operations which can be executed from the terminal can also be prestored via the PROC processor. The PROC processor encompasses the following features.

- Argument passing
- Interactive terminal prompting
- Conditional and unconditional branching
- Pattern matching
- Free-field and fixed-field character moving

The PROC processor is described further in the section titled THE PROC PROCESSOR.

The EDITOR Processor

The EDITOR permits on-line interactive modification of any item in the data base. The EDITOR may be used to create and/or modify DATA/BASIC programs, PROC's, assembly source, data files, and file dictionaries. The EDITOR uses the current line concept; that is, at any given time there is a current line that can be listed, altered, deleted, etc. The EDITOR includes the following features:

- Absolute and relative current line positioning.
- Merging of lines from terminal or from other file items.
- Character string locate and replace.
- Input/Output formatting.

The EDITOR is described further in the section titled THE EDITOR PROCESSOR.

The Data Base Management Processors

The data base management processors provide the capabilities for generating, managing, and manipulating files (or portions of files) within the Reality system. The data base management processors include the CREATE-FILE processor, the CLEAR-FILE processor, the DELETE-FILE processor, and the COPY processor. These processors are further described in the section titled THE DATA BASE MANAGEMENT AND UTILITY PROCESSORS.

The Utility Processors

Numerous utility processors are also included which provide an extensive complement of utility capabilities for the system. These processors are discussed in the section titled THE DATA BASE MANAGEMENT AND UTILITY PROCESSORS.

Software Processor Usage

These and any other software processors may be used by any or all terminals simultaneously. Processing is invoked thru appropriate verbs contained in each terminal user's Master Dictionary. User accessability to these capabilities may be limited by controlling the verb selection available in specific user's Master Dictionaries.

2.3 THE REALITY FILE HIERARCHY

The REALITY files are organized in a hierarchial structure, with files at one level pointing to multiple files at a lower level. Four distinct file levels exist: System Dictionary, User Master Dictionary, File Level Dictionary, and Data File.

The hierarchial Reality file structure is illustrated in Figure A. The term file as used in the context of the Reality system refers to a mechanism for maintaining a set of like items logically together. The data in a file is normally accessed via the dictionary associated with it. Since the dictionary itself is also a file, it contains items (records) just as a data file does. The items in a dictionary serve to define lower level dictionaries or data files.

The Reality system can contain any number of files. Files can contain any number of records, and can automatically grow to any size. Records are variable length, and can contain any number of fields and characters up to a maximum of 32,267 bytes.

System Dictionary (SYSTEM)

The highest level dictionary is called the System Dictionary (SYSTEM). A Reality system contains only one System Dictionary. It contains all legitimate user Logon names, passwords, security codes, and system privileges. This dictionary contains a pointer to each user's Master Dictionary.

User Master Dictionaries (M/DICT's)

The Master Dictionaries (M/DICT's) comprise the next dictionary level. Each user's account may have a unique M/DICT associated with it; the M/DICT defines all user vocabulary (verbs, PROC's, etc.) and accessible file names, and contains attributes describing the structure of the information in lower level dictionaries. The file name pointers can reference any file or dictionary in the system.

File Level Dictionaries

The File Level Dictionaries describe the structure of the data in the associated data files.

Data Files

The Data files contain the actual data stored in variable record/field length format. In addition to the normal record/field data structure, a field (called an attribute) can contain multiple values, and a value (in turn) can consist of multiple sub-values. Thus, data may be stored in a three-dimensional variable length format.

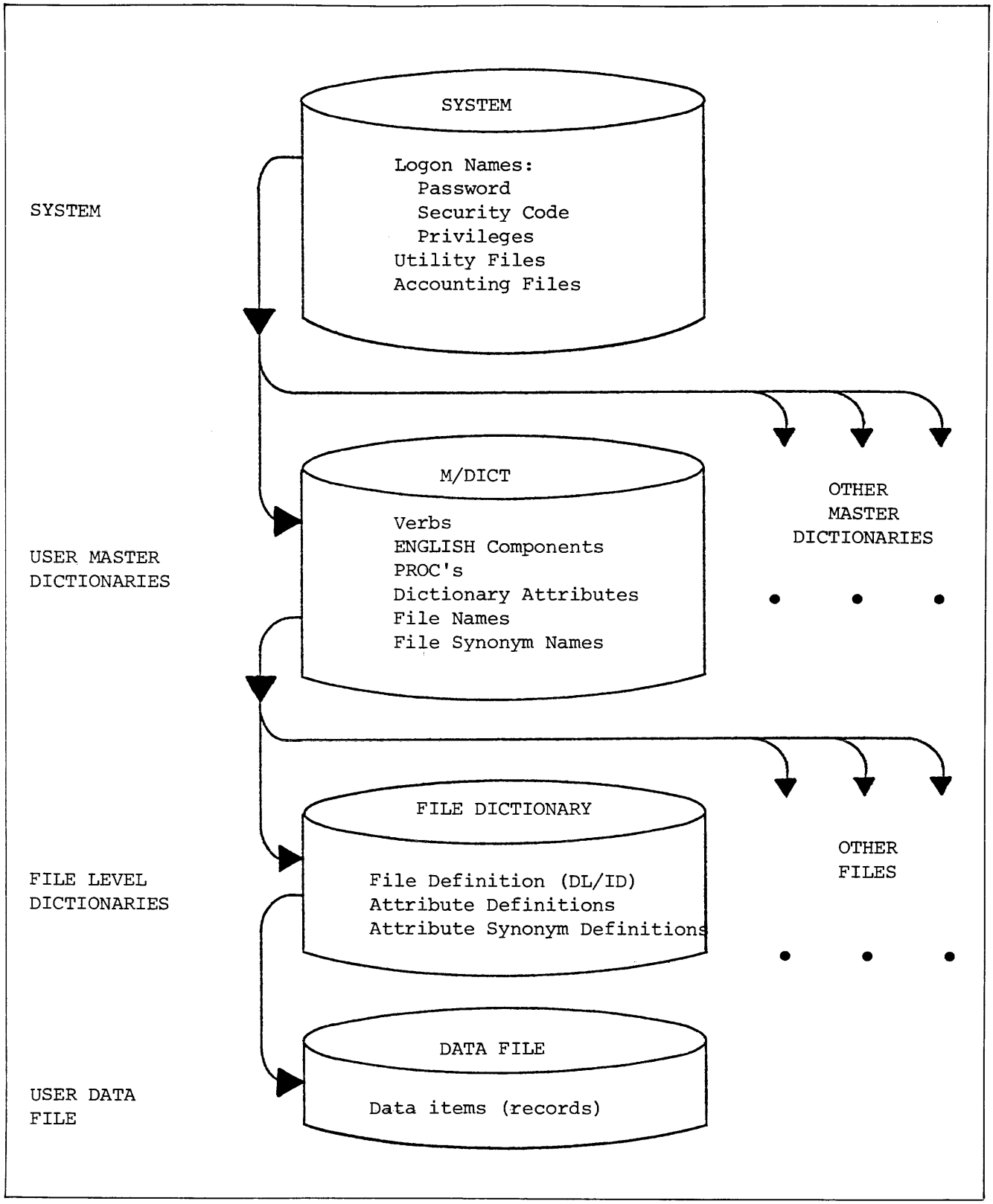

Figure A. The Reality File Hierarchy

2.4 THE REALITY FILE STRUCTURE

> The Reality file access system is designed to very efficiently access any specific item (record) or all items in a file.

A Reality file is a mechanism for maintaining a set of like items logically together so that they can be accessed for both retrieval and update purposes. A file is referenced by a file-name.

A record is called an item. Items are individually variable in length. The maximum size of any item is 32,267 bytes. There is no limit to the number of files in a Reality system, or to the number of items in a file. Each item is associated with an item-id. An item-id is a unique item identifier (key) by which all of the data in the item are identified or referenced.

A computational hashing technique is automatically used by the system; this technique operates on the item-id (using several variables unique to the file) to produce the virtual memory address where the item is stored. This permits direct access to any item regardless of the file size.

An item consists of one or more variable length attributes (also known as fields) separated by attribute mark characters. An attribute, in turn, may consist of any number of variable length values separated by value mark characters. Finally, a value may consist of any number of variable length secondary values (also known as sub-values) separated by sub-value mark characters.

Utility processors like COPY and the EDITOR deal at the file - item - attribute level. They make no logical distinction in definition between various attributes in an item. ENGLISH processors, however, add an additional dimension through the use of the dictionary. The dictionary defines the nature of the information stored for each of the attributes. It permits access by name (e.g., DATA, PRICE, QUANTITY-ON-HAND) and specifies internal and external data formats.

The Reality file structure is summarized in Figure A.

- *REALITY SYSTEM* CONTAINS ON-LINE:
- ANY NUMBER OF *FILES*, WHICH CONTAIN:
- ANY NUMBER OF *ITEMS (RECORDS)*, WHICH CONTAIN:
- MULTIPLE *ATTRIBUTES (FIELDS)*, WHICH MAY CONTAIN:
- MULTIPLE *VALUES*, WHICH MAY CONTAIN:
- MULTIPLE *SUB-VALUES*.

- ALL *FILES*, *ITEMS*, *ATTRIBUTES*, *VALUES*, AND *SUB-VALUES* ARE VARIABLE IN LENGTH.

- EACH *ITEM* MUST BE LESS THAN 32,268 CHARACTERS LONG

Figure A. Reality File Structure Summary

2.5 THE REALITY DICTIONARIES

A dictionary defines and describes data within its associated file. Dictionaries exist at several levels within the Reality system.

As introduced in the topic titled THE REALITY FILE HIERARCHY, the following dictionary levels exist within the Reality system:

- System Dictionary (one per Reality system).
- User Master Dictionary (one per user-account).
- File Level Dictionary (one per data file).

A dictionary defines the nature of the data stored in its associated file. It contains such information as:

- The user-assigned name of the file (or attribute).
- Retrieval and update security codes.
- Conversion specifications which are used to perform table look-ups, masking functions, etc.
- Correlative specifications which are used to describe inter-file and intra-file data relationships.
- Type (alphabetic or numeric) and justification (left or right) for output purposes.
- Maximum and minimum lengths for stored value.
- Pattern editing masks.

Since the dictionary itself is also a file, it contains items just as a data file does. The items in a dictionary serve as the actual definitions for lower level dictionaries or data files. Four types of items are used in dictionaries:

- file definition items
- file synonym definition items
- attribute definition items
- attribute synonym definition items

The file definition items and the file synonym definition items are used to define files. The attribute definition items and the attribute synonym definition items are used to define attributes within data file items. Each dictionary item consists of attributes (just as file items do).

The Reality dictionary concept is graphically illustrated in Figure A. For a detailed discussion of dictionaries and the items they contain, refer to the Reality Programmer's Reference Manual.

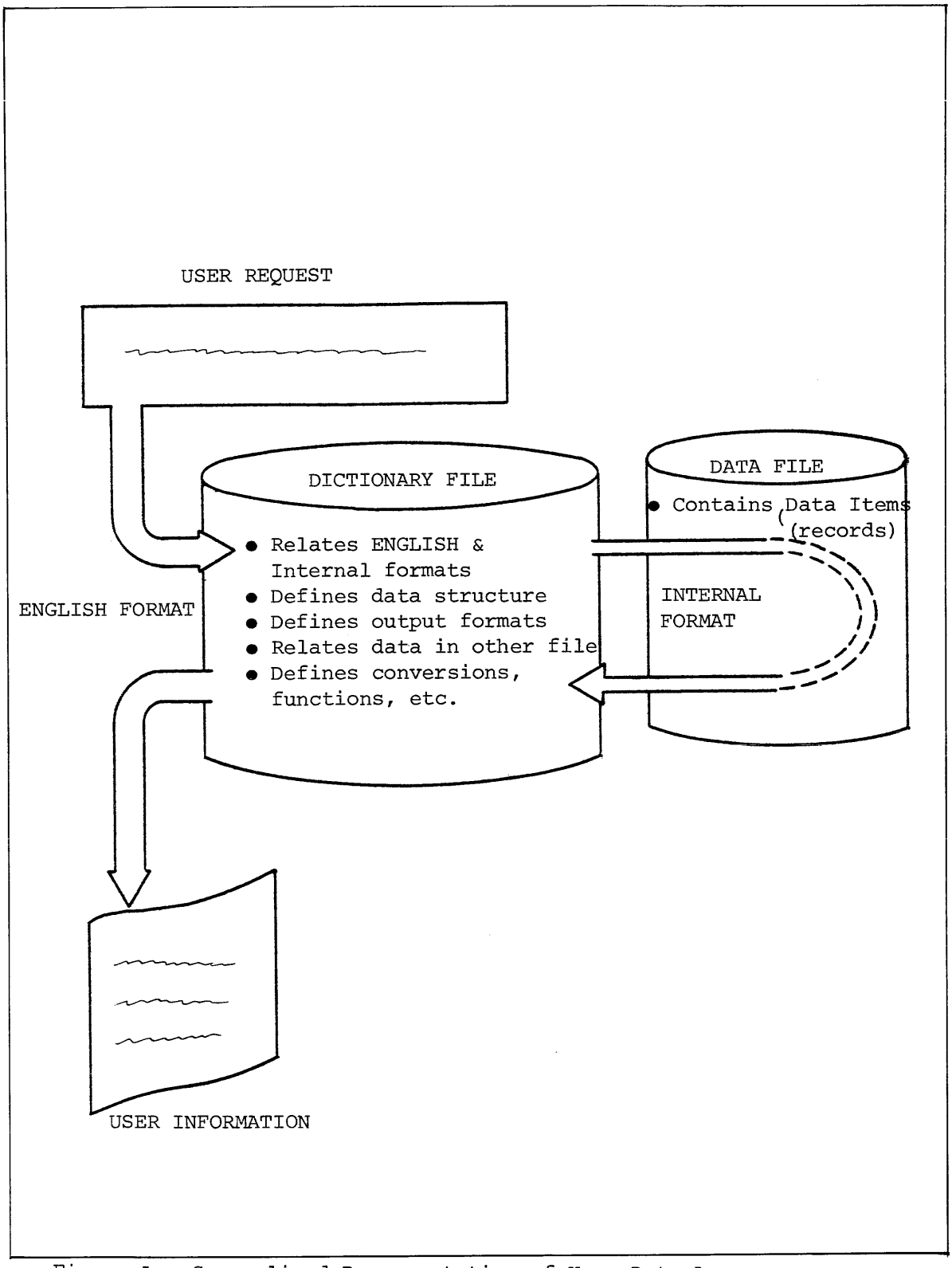

Figure A. Generalized Representation of User Data Access through Reality Dictionary Structure

3 THE TERMINAL CONTROL LANGUAGE (TCL)

3.1 LOGGING ON AND OFF THE SYSTEM

The Logon processor provides a facility for initiating a user's session by identifying valid users and their associated passwords. The Logoff processor is used to terminate the session. These processors accumulate accounting statistics for billing purposes and also associate the user with his privileges and security codes.

Logging On to the System

The user may log on to the Reality System when the following message is displayed:

```
LOGON PLEASE:
```

The user then enters the name (identification) established for him in the system. If a password has also been established, he may follow his identification with a comma, and then the password. If the password is not entered as a response to the LOGON PLEASE message, the system will display the message:

```
PASSWORD:
```

Reality validates the user's identification against the entries in the System Dictionary. If the user has successfully logged on to the system (i.e., both the identification and the password have been accepted), the following message is displayed:

```
***    WELCOME  TO  MICRODATA R E A L I T Y   ***
***    time       RELEASE x.y          date   ***
:
```

where "time" is the current time, "date" is the current date, and "x.y" is the current Reality software release level. A colon (:) is the Terminal Control Language (TCL) prompt character, which indicates that the user may now enter any valid command. Figure A illustrates a sample logon interaction.

Logging OFF of the System

Logoff is achieved by entering the word OFF. A message indicating the connect time (i.e., number of minutes that the user was logged on) and the appropriate charge units will be displayed. The system then displays the LOGON PLEASE message and waits for the next user session to be initiated. The general form of the logoff message is as follows:

```
***    CONNECT TIME = n MINS.;   CHARGE UNITS = m    ***
***    LOGGED OFF AT   time              ON   date    ***

LOGON PLEASE:
```

where "n" is the number of minutes of connect time, "m" is the number of charge units, "time" is the current time, and "date" is the current date. The charge-units represent usage of the CPU; it is normally in tenths of a CPU second. Figure B illustrates a sample lofoff interaction.

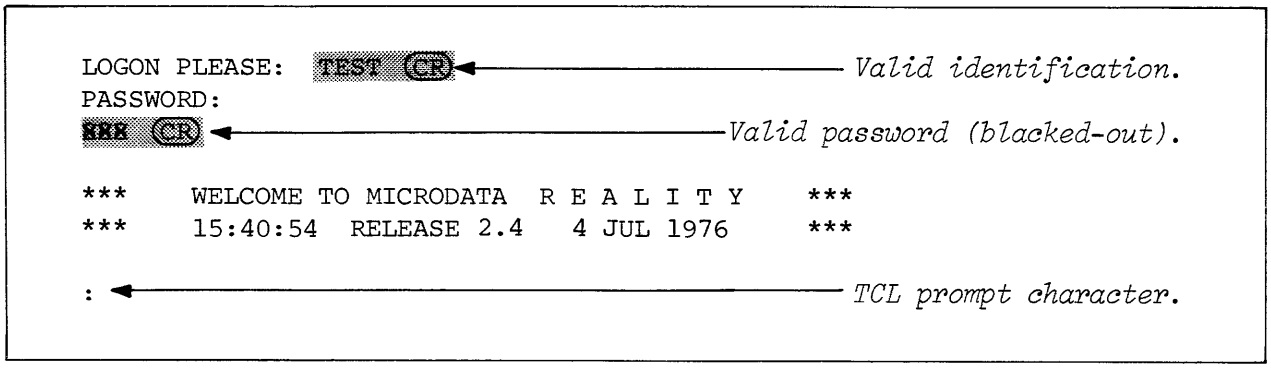

Figure A. Sample Logon Interaction

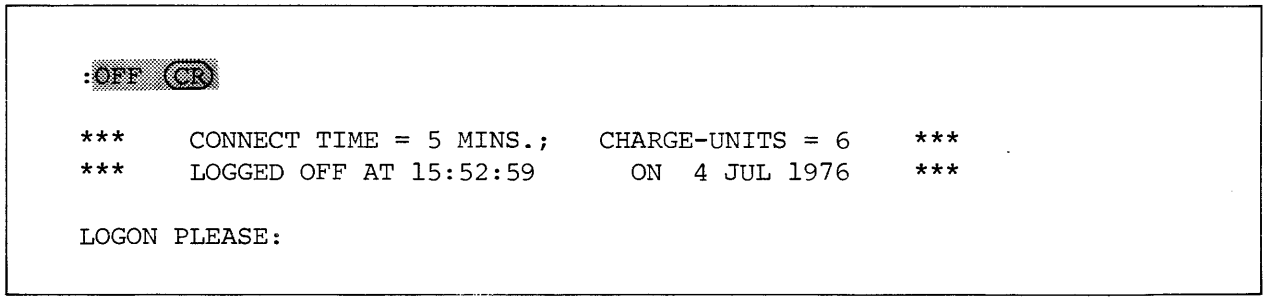

Figure B. Sample Logoff Interaction

3.2 VERBS AND PROCESSORS

The Terminal Control Language (TCL) is the primary interface between the terminal user and the various Reality processors.

Most processors are evoked directly from the Terminal Control Language by a single input statment, and return to TCL after completion of processing. TCL prompts the user by displaying a colon (:). This is referred to as the "TCL prompt character". Input statements are constructed by typing a character at a time from the terminal until the carriage return or line feed key is depressed, at which time the entire line is processed by TCL.

The first word of an input statement must be a valid Reality "verb." (The statement must not contain any other verbs.) Selected verbs are listed in Figure A.

One of the powerful features of Reality is the ability to customize the vocabulary for each user. Since verbs reside in the individual user's Master Dictionary (M/DICT), the vocabulary may be added to or deleted from without affecting the other users. In addition, an unlimited number of synonyms may be created for each verb.

Reality operates in the full-duplex mode of communication with each user's terminal. Full-duplex means that data may be transmitted in both directions simultaneously between the terminal and the computer. Additionally, Reality operates in what is known as an "Echo-Plex" environment. This means that each data character input by the terminal is sent to the computer and echoed back to the terminal before being displayed. The user is thus assured that if the data character displayed on the terminal is correct, the data character stored in the computer is correct.

For a complete discussion of the Terminal Control Language, the user should refer to the Reality Programmer's Reference Manual.

VERB	DESCRIPTION
ASSEM (or AS)	*Assembles source code.*
BASIC	*Compiles a DATA/BASIC program.*
CATALOG	*Catalogs a DATA/BASIC program.*
CLEAR-FILE	*Removes all items from a file or dictionary.*
COPY	*Copies data/dictionary files and items.*
COUNT	*Counts number of items which meet specified conditions.*
CREATE-FILE	*Creates a new file.*
DELETE-FILE	*Deletes an entire file.*
EDIT (or ED)	*Evokes the EDITOR processor.*
GROUP	*Provides file usage statistics on groups.*
ISTAT	*Generates a file hashing histogram for a file.*
ITEM	*Provides usage statistics on file items.*
LIST	*Generates a formatted output of selected items and attributes.*
RUN	*Executes a DATA/BASIC program.*
SELECT	*Selects items for use by subsequent processor.*
SORT	*Generates a sorted and formatted output of selected items and attributes.*
SSELECT	*Selects and sorts items for use by subsequent processor.*
STAT	*Counts, averages, and sums a specified attribute.*
TERM	*Sets terminal characteristics.*
TIME	*Prints time and date.*
WHAT	*Displays current system parameters.*
WHO	*Prints the line number and account number to which the terminal is logged on.*

Figure A. Typical Reality Verbs

4 THE DATA BASE MANAGEMENT AND UTILITY PROCESSORS

4.1 DATA BASE MANAGEMENT

The data base management processors provide the capabilities for generating, managing, and manipulating files (or portions of files) within the Reality system. The data base management processors include the CREATE-FILE processor, the CLEAR-FILE processor, the DELETE-FILE qrocessor, and the COPY processor.

The CREATE-FILE Processor

The CREATE-FILE processor is used to generate new dictionaries and/or data files. The processor creates file dictionary entries in the user's Master Dictionary (M/DICT), and can also be used to reserve disc space for the data portion of the new file. The user need only specify the name of the file and values for the desired "modulo" and "separation." The "module" and "separation" parameters are selected to balance storage efficiency and accessing speed, based on the number of items in the file, the average item size, etc. The required file space is allocated from the available file space pool. Files may automatically grow beyond their initial size by attaching additional "overflow" space from the available file space pool.

The CLEAR-FILE Processor

The CLEAR-FILE processor clears the data from a file. "Overflow" space that may be linked to the primary file space will be released to the available file space pool. Either the data section or the dictionary section of a file may be cleared.

The DELETE-FILE Processor

The DELETE-FILE processor allows for the deletion of a file. All allocated file space is returned to the available file space pool. Either the data section or the dictionary section (or both) of the file may be deleted.

The COPY Processor

The COPY processor allows the user to copy an entire file (or selected items from the file) to the terminal, to the line printer, to the magnetic tape unit, to another file (either in the same account or in some other user-account), or to the same file under a different name (item-id).

Examples

As a general introduction, Figure A presents a number of examples illustrating the use of the file management processors. For further information regarding these processors, refer to the Reality Programmer's Reference Manual.

EXAMPLE	EXPLANATION
:CREATE-FILE (DICT TEST 5,1) CR	*Creates a file dictionary for the TEST file, with a modulo of 5 and a separation of 1.*
:CREATE-FILE (DATA TEST 7,2) CR	*Reserves disc space for the data area of the TEST file, with a modulo of 7 and a separation of 2.*
:CREATE-FILE (FNA 3,1, 11,2) CR	*Creates a file dictionary for the FNA file, with a modulo of 3 and a separation of 1. Also reserves disc space for the data area of the FNA file, with a modulo of 11 and a separation of 2.*
:CLEAR-FILE (DATA XYZ) CR	*Clears data section of XYZ file.*
:DELETE-FILE (DICT INV) CR	*Deletes dictionary section of INV file.*
:DELETE-FILE (FAB) CR	*Deletes data and dictionary sections of FAB file.*
:COPY TEST I1 I2 I3 CR TO:X1 X2 X3 CR	*Copies data items I1, I2, and I3 back into the same file (TEST) but gives them item-id's of X1, X2, and X3.*
:COPY DICT SAMPLE * CR TO:(DICT FLAVORS) CR	*Copies all dictionary items from file SAMPLE to the dictionary of file FLAVORS.*
:COPY TEST * (P) CR	*Copies all items in the TEST file to the line printer.*
:COPY TEST * CR TO: CR	*Copies all items in the TEST file to the user's terminal.*

Figure A. Sample Usage of File Management Processors

4.2 UTILITIES

The Reality Utility processors provide an extensive complement of utility capabilities for the system.

The Reality Computer System includes a very large number of utility processors. These processors provide such capabilities as:

- Magnetic tape unit functions
- Mathematical functions
- Line printer spooling control
- File save/restore functions
- File statistics
- Creation of user-accounts
- Setting of terminal charactersitics
- Block printing
- Virtual memory dumping
- Inter-user message communications
- Bootstrapping and cold-start
- Systems accounting

A few examples of utility processor usage is shown in Figure A. For further information regarding the Reality utility processors, the reader is referenced to the Reality Programmer's Reference Manual.

EXAMPLE	EXPLANATION
:T-ATT (CR)	*Attaches magnetic tape unit to terminal issuing command.*
:T-FWD 10 (CR)	*Spaces magnetic tape forward 10 records.*
:T-READ (4-6) (CR)	*Bypasses next 3 magnetic tape records; dumps 4th 5th, and 6th records at terminal, and positions tape at beginning of 7th record.*
:T-DUMP DICT TEST-FILE (CR)	*Dumps to the magnetic tape all items in the dictionary of the TEST-FILE file.*
:T-REW (CR)	*Rewinds magnetic tape unit to BOT.*
:ADDD 5 1 (CR) 6	*Adds decimal 5 to decimal 1 (result is decimal 6).*
:MULX FFF EEF (CR) EEE111	*Multiplies hex FFF to hex EEF (result is hex EEE111).*
:SP-STATUS (CR)	*Displays current spooler status.*
:TERM 79,23,1,3,1,21 (CR)	*Sets specific terminal characteristics.*
:BLOCK-PRINT AB12 (CR)	*Produced block-print of characters AB12 on line printer.*
:MESSAGE ROD HELLO THERE (CR)	*Transmits message "HELLO THERE" to user ROD.*

Figure A. Sample Usage of Utility Processors

5 THE ENGLISH PROCESSORS

5.1 AN OVERVIEW

> ENGLISH is a user-oriented data retrieval language used for accessing files within the Reality Computer System.

ENGLISH is a generalized information management and data retrieval language. A typical ENGLISH inquiry consists of a relatively free-form sentence containing appropriate verbs, file names, data selection criteria, and control modifiers. Each user's vocabulary can be individually tailored to his particular application jargon.

ENGLISH is a dictionary-driven language to the extent that the vocabulary used in composing an ENGLISH sentence is contained in several dictionaries. Verbs and file names are located in each user's Master Dictionary (M/DICT). User-files consist of a data section and a dictionary section; the dictionary section contains a structural definition of the data section. ENGLISH references the dictionary section for data attribute descriptions. These descriptions specify attribute fields, functional calculations, inter-file retrieval operations, display format, and more.

Not only does ENGLISH provide an ability to selectively or conditionally retrieve information, it also provides an automatic report generation capability. Output reports (which normally appear on the terminal but optionally may be transmitted to the line printer) are automatically formatted for the user by the Reality system. The output may be sorted into any sequence defined by the user, and attributes may be totaled based on user-specified control breaks.

ENGLISH encompases the following extended features:

- Relatively free-form input of word order and syntax
- Automatic or user-specified output report formats
- Generalized data selection using logical and arithmetic relationships
- Sorting capability on variable number of descending or ascending sort-keys
- Generation of statistical information concerning files
- Selection and sorting of items for use by subsequent processors
- Support of 11 digit signed arithmetic

Figure A through C illustrates some typical ENGLISH inquiries.

```
:LIST ACCOUNT WITH BILL-RATE = "30" NAME ADDRESS BILL-RATE (CR)

PAGE 1                                          11:08:37   16 JAN 1976

ACCOUNT... NAME.....................  ADDRESS.............  BILL-RATE

11115      D R MASTERS                 100 AVACADO                30
11085      A B SEGUR                   101 BAY STREET             30
11040      E G MCCARTHY                113 BEGONIA                30
11050      J R MARSHECK                125 BEGONIA                30
11020      J T O'BRIEN                 124 ANCHOR PL              30
11095      J B STEINER                 124 AVACADO                30
11110      D L WEISBROD                106 AVACADO                30
11015      L K HARMAN                  118 ANCHOR PL              30
11105      C C GREEN                   112 AVACADO                30
11090      J W JENKINS                 130 AVACADO                30
23030      L J DEVOS                   201 CARNATION              30

11 ITEMS LISTED
```

Figure A. Sample ENGLISH Inquiry Using LIST Verb

```
:STAT TEST-FILE DEPOSIT WITH NO CURR-BAL (CR)

STATISTICS OF DEPOSIT  :
TOTAL = 39.00 AVERAGE = 7.800  COUNT = 5
```

Figure B. Sample ENGLISH Inquiry Using STAT Verb

```
:SORT ACCOUNT > '35070' NAME DEPOSIT BY DEPOSIT (CR)

PAGE 1                                          14:15:47   16 JAN 1976

ACCOUNT... NAME.....................  DEPOSIT

35090      D U WILDE                      3.17
35100      R W FORSTROM                   8.00
35110      H E KAPLOWITZ                 10.00
35080      G A BUCKLES                   10.50
35095      A W FEVERSTEIN                10.75
35105      S J FRYCKI                    10.80
35075      J L CUNNINGHAM                10.90
35085      J F SITAR                     12.00

8 ITEMS LISTED.
```

Figure C. Sample ENGLISH Inquiry Using SORT Verb

5.2 AN ENGLISH PRIMER

The user forms ENGLISH sentences which specify the desired data retrieval functions. The ENGLISH retrieval language is limited natural English; formats for sentences are simple yet very general. The ENGLISH processors, together with the use of dictionaries, permit inputs to be stated directly in the technical terminology natural to each application area.

ENGLISH accepts any number of variable length words and permits a general freedom of word order and syntax. An ENGLISH sentence is entered at the TCL level, i.e., when the system prompts with a colon (:). The sentence then directs the appropriate ENGLISH processor to perform the specified data retrieval function. The general form of the ENGLISH sentence contains several grammatical structures which can be represented as shown in Figure A.

The *verb* must be the first word in the ENGLISH sentence, while the other words may generally be in any order. ENGLISH verbs are action-oriented words which evoke specific ENGLISH processors. The *file-name* specification permits the access of either the data section or the dictionary section of a file. A verb and a file-name are required; all other elements are optional. Thus, the minimum ENGLISH sentence consists of a verb followed by a file-name.

The *attribute* list specifies those attributes desired for output. The attribute list may be explicity stated using attribute names found in the file dictionary. If none are specified in sentence, the implicit attribute synonym list in the file dictionary will be used to specify the displayed fields.

The *selection criteria* determine which items in the file will be operated upon. If nothing is specified, then all items will be used. One or more direct references may be made by specifying the item-id in single quotes. A conditional retrieval may be specified by using a WITH clause. All items in the file will be interrogated, but only those meeting the specified criteria will be accepted. The WITH clause may be a simple or complex combination of attribute names, relational operators (=, >, LT, AFTER, etc.), logical operators (AND, OR), and explicit data values ("100", "12/2/76", "RESISTOR", etc.).

The *miscellaneous connectives* may be used to modify the effect of the verb, or to alter the display format.

Figure A illustrates a number of sample ENGLISH sentences.

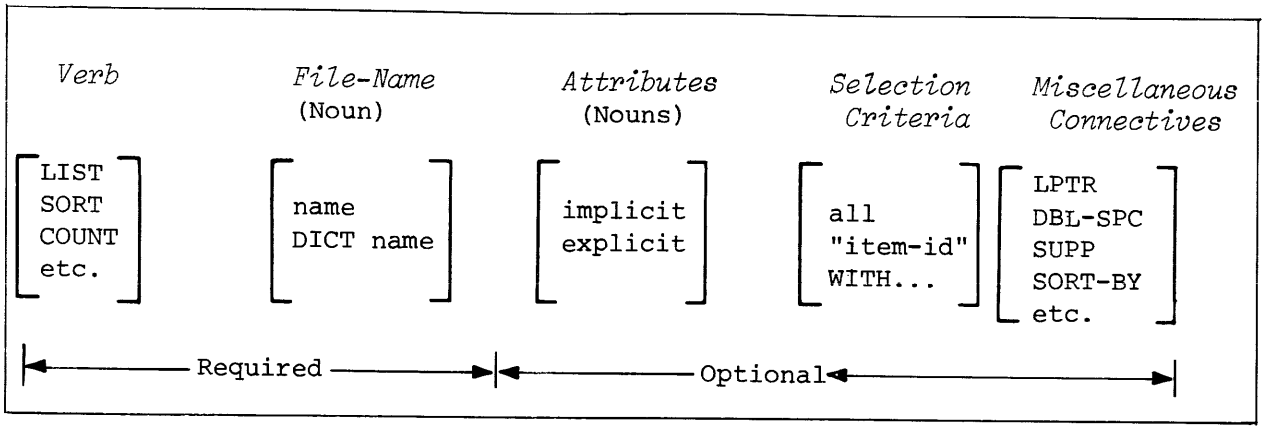

Figure A. Generalized Grammatical Structure of an ENGLISH Sentence

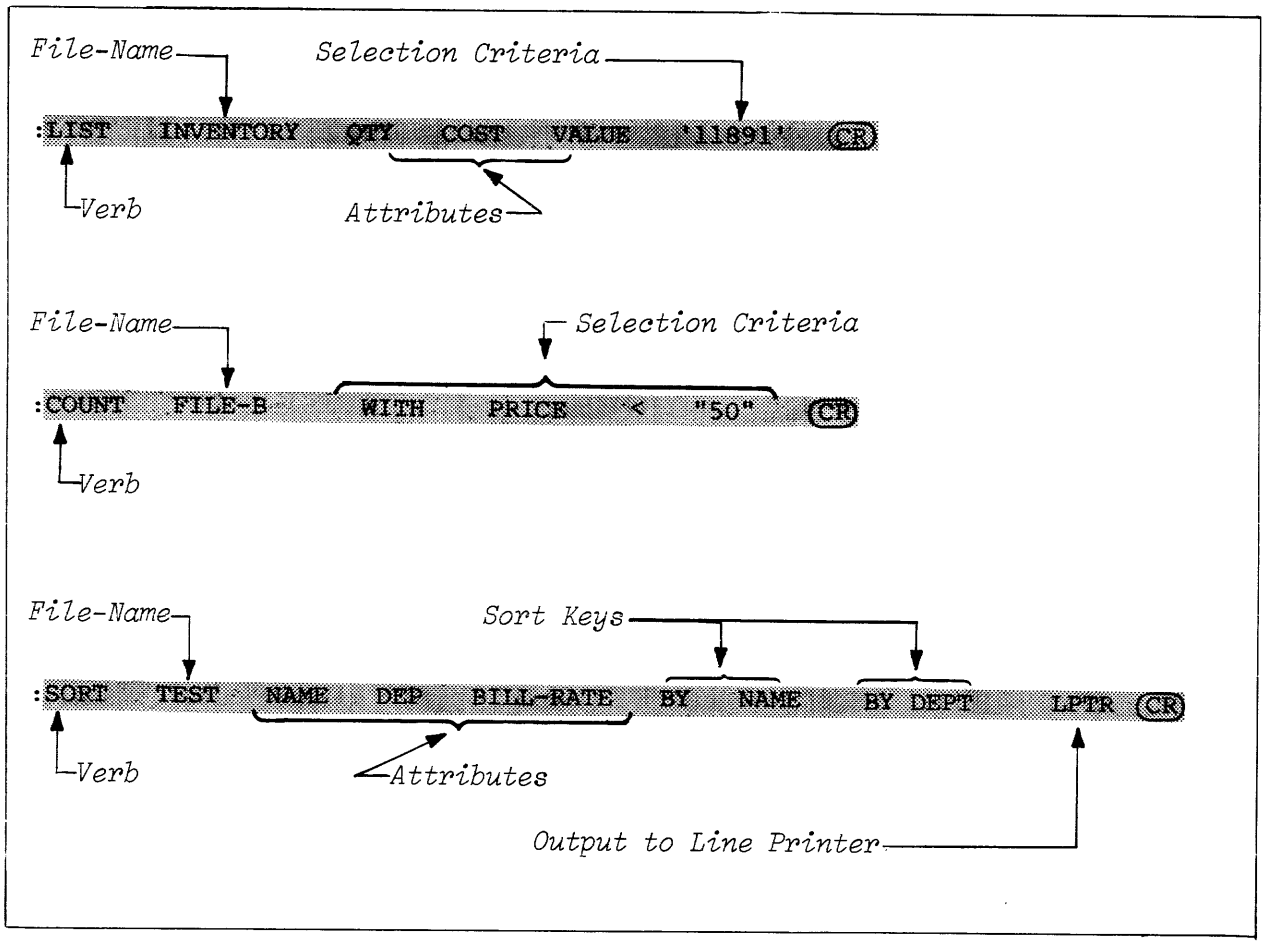

Figure B. Sample ENGLISH Sentences

5.3 THE ENGLISH VERBS

Each ENGLISH sentence must begin with one (and only one) ENGLISH verb. ENGLISH verbs are action-oriented words which evoke specific ENGLISH processors. Some of the major ENGLISH verbs are briefly discussed below.

LIST and SORT

The LIST and SORT verbs are used to generate formatted output. LIST simply lists the selected output, while SORT orders the output in some specified sorted order. Generated output will be formatted into a columnar output if possible, taking into account the maximum defined size of the specified attributes and their associated names, along with the width of the terminal page. If more attributes have been specified than will fit across the page, a non-columnar output will be generated with the attribute names down the side and the associated attribute values to the right. LIST and SORT will automatically format multivalued attributes and sub-values. They provide sub-totalling via the BREAK-ON and TOTAL modifiers, as well as other format controls. Sample use of the LIST verb with noncolumnar output is shown in Figure A. SORT can handle any number of ascending or descending sort keys.

COUNT

The COUNT verb counts the number of items meeting the conditions specified. The output generated by this verb is simply the number of items counted. Figure B illustrates the use of the COUNT verb.

SUM and STAT

The SUM and STAT verbs provide a facility for summing one specified attribute. The STAT verb additionally provides a count and average for the specified attribute. The output generated by these verbs are the derived statistics. Figure C illustrates the use of the SUM verb.

SELECT and SSELECT

The SELECT verb provides a facility to select a set of items. These selected items are then available one at a time to certain Reality processors. The output from the SELECT verb is a message signaling the number of items extracted or selected. The SSELECT verb combines the SORT capability with the SELECT capability.

T-DUMP, I-DUMP, ISTAT, HASH-TEST, and CHECK-SUM

The T-DUMP and I-DUMP verbs allow the user to selectively dump his dictionaries and data files to the magnetic tape or to the terminal, respectively. The ISTAT and HASH-TEST verbs provide file hashing histograms. The CHECK-SUM verb is used to determine if data in a file have been changed.

```
:LIST ACCOUNT '23080' '23090' NAME ADDRESS START-DATE CURR-BALNC (CR)

PAGE 1                                                  11:19:58    16 JAN 1976

ACCOUNT  :  23080
NAME   J W YOUNG
ADDRESS  207 COVE STREET
START-DATE  27 MAR 1970
CURR-BALNC  $    89.32

ACCOUNT  :  23090
NAME  W J HIRSCHFIELD
ADDRESS  230 BEGONIA
START-DATE  01 JAN 1968
CURR-BALNC  $    20.45

2 ITEMS LISTED
```

Figure A. Sample ENGLISH Inquiry Using LIST Verb (Non-Columnar Output)

```
:COUNT ACCOUNT GE '11115' WITH CURR-BALNC AND WITH BILL-RATE "30" (CR)

2 ITEMS COUNTED.

:COUNT ACCOUNT WITH NO SEWER-ASMT (CR)

57 ITEMS COUNTED

:COUNT TEST (CR)

10 ITEMS COUNTED.

:COUNT DICT INVENTORY WITH D/CODE "A" (CR)

55 ITEMS COUNTED.
```

Figure B. Sample ENGLISH Inquiries Using COUNT Verb

```
:SUM ACCOUNT CURR-BALNC (CR)
TOTAL OF CURR-BALNC IS:  $2,405,118.10
:SUM ACCOUNT CURR-BALNC WITH CURR-BALNC > "100000" (CR)
TOTAL OF CURR-BALNC IS  : $1,836,287.99
```

Figure C. Sample ENGLISH Inquiries Using SUM Verb

6 THE DATA/BASIC PROCESSOR

6.1 AN OVERVIEW

> The DATA/BASIC Language is an extended version of Dartmouth BASIC, specifically designed for data base management processing on Reality.

BASIC (Beginners All-Purpose Symbolic Instruction Code) is a simple yet versatile programming language suitable for expressing a wide range of problems. Developed at Dartmouth College in 1963, BASIC is a language especially easy for the beginning programmer to master. DATA/BASIC is an extended version of BASIC with the following features:

- Optional statement labels (statement numbers)
- Statement labels of any length
- Alphanumeric variable names of any length
- Multiple statements on one line
- Complex IF statements
- Multiline IF statements
- Formatting and terminal cursor control
- String handling with unlimited, varying length strings
- One and two dimensional arrays
- Magnetic tape input and output
- Floating point arithmetic with up to 11 digit precision
- ENGLISH data conversion capabilities
- Reality file access and update capabilities
- Pattern matching
- Dynamic file arrays
- External subroutines

Sample DATA/BASIC programs are presented in Figures A and B. The program in Figure A lists (prints) the numbers from 1 to 10. The program in Figure B queries an inventory file as further described by the program's comment statements, i.e., program statements which begin with an asterisk (*) are considered comment (remark) statements.

```
      I=1
    5 PRINT I
      IF I=10 THEN STOP
      I=I+1
      GOTO 5
      END
```

Figure A. Sample DATA/BASIC Program which Prints the Numbers from 1 to 10

```
*************************************************************************
*                                                                       *
*     THIS PROGRAM QUERIES AN INVENTORY FILE.                           *
*     IT READS THE DICTIONARY OF FILE 'INV' TO GET THE ATTRIBUTE        *
*     NUMBERS OF 'DESC' (DESCRIPTION) AND 'QOH' (QUANTITY-ON-HAND).     *
*     THE PROGRAM THEN PROMPTS THE USER FOR A PART-NUMBER WHICH         *
*     IS THE ITEM-ID OF AN ITEM IN 'INV' AND USES THE ATTRIBUTE         *
*     NUMBERS TO READ AND DISPLAY THE PART DESCRIPTION AND              *
*     QUANTITY ON HAND.  THE PROGRAM LOOPS UNTIL A NULL PART            *
*     NUMBER IS ENTERED.                                                *
*************************************************************************
*
*     GET ATTRIBUTE DEFINITIONS FROM DICTIONARY OF INVENTORY FILE
      OPEN 'DICT','INV' ELSE PRINT 'CANNOT OPEN "DICT INV"'; STOP
      READV DESC.AMC FROM 'DESC',2 ELSE PRINT 'CANT READ "DESC" ATTR'; STOP
      READV QOH.AMC  FROM 'QOH',2  ELSE PRINT 'CANT READ "QOH"  ATTR'; STOP
*     OPEN DATA PORTION OF INVENTORY FILE
      OPEN '', 'INV' ELSE PRINT 'CANNOT OPEN "INV"'; STOP
*     PROMPT FOR PART NUMBER
100   PRINT
      PRINT 'PART-NUMBER ':
      INPUT PN
      IF PN = '' THEN PRINT '--DONE--'; STOP
      READV DESC FROM PN,DESC.AMC ELSE PRINT 'CANT FIND THAT PART'; GOTO 100
      READV QOH  FROM PN,DESC.AMC ELSE PRINT QOH=0
*     PRINT DESCRIPTION AND QUANTITY-ON-HAND
      PRINT 'DESCRIPTION - ': DESC
      PRINT 'QTY-ON-HAND - ': QOH
      PRINT
      GOTO 100
  END
```

Figure B. Sample DATA/BASIC Program which Queries an Inventory File

6.2 DATA/BASIC LANGUAGE DEFINITION

A DATA/BASIC program is comprised of DATA/BASIC statements. DATA/BASIC statements may contain variables, constants, expressions, and DATA/BASIC Intrinsic Functions.

A DATA/BASIC program consists of a sequence of DATA/BASIC statements terminated by an END statement. More than one statement may appear on the same program line, separated by semicolons. Any DATA/BASIC statement may begin with an optional statement label. A statement label is used so that the statement may be referenced from other parts of the program.

DATA/BASIC statements may contain arithmetic, relational, and logical expressions. These expressions are formed by combining specific operators with variables, constants, or DATA/BASIC Intrinsic Functions. The value of a variable may change dynamically throughout the execution of the program. A constant, as its name implies, has the same value throughout the execution of the program. An Intrinsic Function performs a predefined operation upon the parameter(s) supplied.

The DATA/BASIC Intrinsic Functions are listed in Figure A. Figure B lists the DATA/BASIC statements.

The reader should note that a DATA/BASIC program, when stored, constitutes a file item, and is referenced by its item-id (i.e., the name it is given when it is created via the EDITOR). An individual line within the DATA/BASIC program constitutes an attribute.

FUNCTION	BRIEF DESCRIPTION
ABS	*Returns an absolute value.*
ASCII	*Converts string from EBCDIC to ASCII.*
CHAR	*Converts numeric value to ASCII character.*
COL1	*Returns column position preceding FIELD-selected sub-string.*
COL2	*Returns column position following FIELD-selected sub-string.*
DATE	*Returns current internal date.*
DELETE	*Deletes attribute, value, or sub-value from dynamic array.*
EBCDIC	*Converts string from ASCII to EBCDIC.*
EXTRACT	*Returns attribute, value, or sub-value from dynamic array.*
FIELD	*Returns a delimited sub-string.*
ICONV	*Provides for Reality input conversion.*
INDEX	*Returns column position of sub-string.*
INSERT	*Inserts attribute, value, or sub-value into dyanmic array.*
INT	*Return an integer value.*
LEN	*Returns length of string.*
NOT	*Returns logical inverse.*
NUM	*Tests for numeric value.*
OCONV	*Provides for Reality output conversion*
REPLACE	*Replaces attribute, value, or sub-value in dynamic array.*
RND	*Generates random number.*
SPACE	*Generates string containing blanks.*
STR	*Generates specified string.*
TIME	*Returns internal time of day.*
TIMEDATE	*Returns external time and date.*
@	*Controls terminal cursor.*

Figure A. Summary of DATA/BASIC Intrinsic Functions

STATEMENT	BRIEF DESCRIPTION
CHAIN	*Passes control to another DATA/BASIC program.*
CLEAR	*Initializes all variables to zero.*
CLEARFILE	*Clears data section of specified file.*
DELETE	*Deletes specified file item.*
DIM	*Reserves storage for arrays.*
END	*Designates the physical end of the program.*
FOR	*Specifies the beginning of a program loop.*
GOSUB	*Transfers control to a subroutine.*
GOTO	*Transfers control to another statement.*
HEADING	*Prints a page heading.*
IF	*Provides for conditional execution of specified statements.*
INPUT	*Inputs data from the terminal.*
LOCK	*Sets an execution lock.*
LOOP	*Provides for structured program loops.*
MAT	*Assigns value to each element of an array.*
NEXT	*Specifies the ending of a program loop.*
NULL	*Specifies a non-operation.*
OPEN	*Selects a file for subsequent I/O.*
PAGE	*Pages output device and prints heading.*
PRINT	*Causes specified data to be printed.*
PRINTER	*Controls selection of printer or terminal for program output.*
PROMPT	*Selects a prompt character for the terminal.*
READ	*Reads a file item.*
READNEXT	*Reads next item-id.*
READT	*Reads next magnetic tape record.*
READV	*Reads an attribute.*
REM	*Specifies a remark (command) statement.*
RETURN	*Returns control from a subroutine.*
REWIND	*Rewinds magnetic tape.*
RQM	*Terminates programs current time quantum.*
STOP	*Designates a logical end of the program.*
STORAGE	*Controls dynamic storage allocation.*
UNLOCK	*Resets an execution lock.*
WEOF	*Writes an EOF on magnetic tape.*
WRITE	*Updates a file item.*
WRITET	*Writes a magnetic tape record.*
WRITEV	*Updates an attribute value.*
=	*Assigns value to variable.*

Figure B. Summary of DATA/BASIC Statements

6.3 CREATING, COMPILING, AND EXECUTING DATA/BASIC PROGRAMS

The DATA/BASIC program is created via the EDITOR, is compiled by issuing the BASIC verb, and is executed by issuing the RUN verb.

DATA/BASIC programs are created via the Reality EDITOR. To enter the EDITOR, the user issues the EDIT verb. The general command format is:

```
EDIT file-name item-id
```

The EDITOR will be entered, and the user may begin entering his DATA/BASIC program. The program will have the name specified by "item-id".

Once the DATA/BASIC program has been created, it may be compiled by issuing the BASIC verb. The general command format is:

```
BASIC file-name item-id
```

The "file-name" and "item-id" specify the DATA/BASIC program to be compiled. If the program is incorrectly formed, compilation errors will result. Error messages are printed as the program is compiled.

RUN is the verb issued to execute a compiled DATA/BASIC program. This command locates the assembly code version of the compiled DATA/BASIC program, which is then loaded and executed. The general command format is:

```
RUN  file-name item-id
```

The "file name" and "item-id" specify the compiled DATA/BASIC program to be executed. If run-time errors occur, appropriate warning and/or fatal error messages will be printed. Fatal run-time errors will cause the program to abort.

A DATA/BASIC program may be cataloged by issuing the CATALOG verb. The general command format is:

```
CATALOG file-name item-id
```

The cataloged program can then be executed by simply entering the program name (item-id) as a verb.

Figure A illustrates the creation, compilation, and execution of the following DATA/BASIC program:

```
PRINT "THIS IS"
PRINT "A TEST"
END
```

The program is stored in the file named PROGRAMS and in the item named TESTING.

For further information regarding the DATA/BASIC processor, refer to the DATA/BASIC Programming Manual.

```
:EDIT PROGRAMS TESTING (CR)
NEW ITEM
TOP

.I (CR)
001 PRINT "THIS IS" (CR)
002 PRINT "A TEST" (CR)
003 END (CR)
004 (CR)
TOP
.FI (CR)
'TESTING' FILED.

:BASIC PROGRAMS TESTING (CR)
TESTING
***
LINE 003  B0  COMPILATION COMPLETED

:RUN PROGRAMS TESTING (CR)
THIS IS
A TEST

:
```

Figure A. Creation, Compilation, and Execution of Sample DATA/BASIC Program

7 THE EDITOR PROCESSOR

7.1 AN OVERVIEW

The EDITOR is a Reality processor which permits on-line interactive modification of any item in the data base.

The Reality EDITOR may be used to create and/or modify DATA/BASIC programs, PROC's, assembly source, data files, and file dictionaries. The EDITOR uses the current line concept; that is, at any given time there is a current line (i.e., attribute) that can be listed, altered, deleted, etc. The Reality EDITOR includes the following features:

- Two variable length temporary buffers
- Absolute and relative current line positioning
- Line number prompting on input
- Merging of lines from the same or other items
- Character string locate and replace
- Conditional and unconditional line deletion
- Input/Output formatting
- Prestoring of commands

Figure A illustrates a sample EDITOR session.

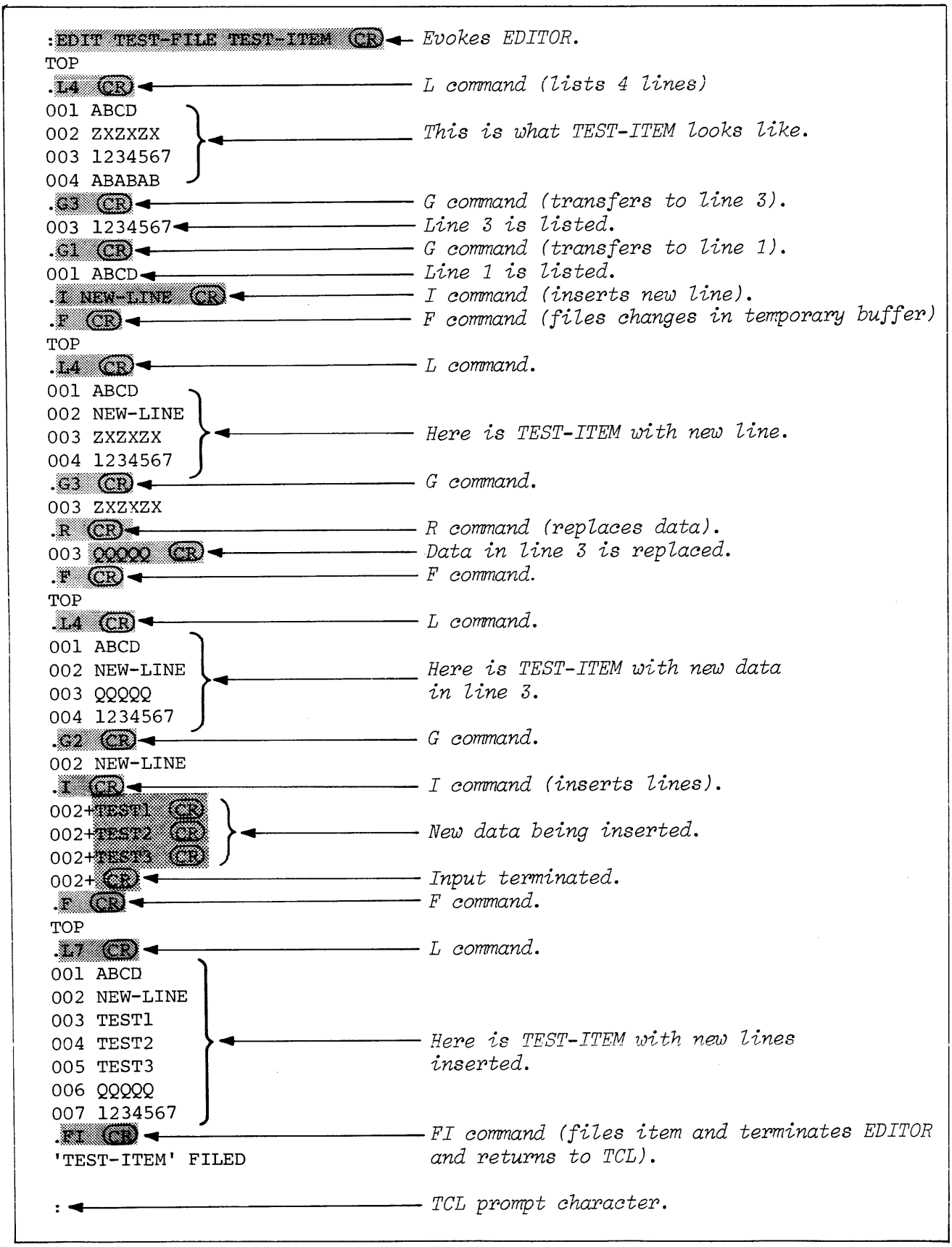

Figure A. Sample EDITOR Session

7.2 EDITOR LANGUAGE DEFINITION

The EDIT verb is used to evoke the EDITOR. EDITOR commands are then issued to update the item on a "line-at-a-time" basis.

The EDITOR is entered by issuing the EDIT verb. The general command format is as follows:

```
EDIT file-name item-id
```

The item specified by "file-name" and "item-id" will be edited. If the specified item does not already exist on file, a new item will be created.

The EDITOR uses two variable length temporary buffers to create or update an item. When the EDITOR is entered, the item to be edited is copied into one buffer. Each line (i.e., attribute) of the item is associated with a line number; a "current line pointer" points to the current line of the item. EDITOR operations are performed on one line at a time (the current line) in an ascending line number sequence. As an EDITOR operation is performed on a line, the modified line and all previous lines are copies to the second buffer.

EDITOR commands are one or two letter mnemonics. Command parameters follow the command mnemonic. The EDITOR commands are summarized in Figure A.

For further information regarding the Reality EDITOR, the reader should refer to the EDITOR Operator's Guide.

COMMAND	BRIEF DESCRIPTION
A	*Executes last Locate (L) command again.*
B	*Moves current line pointer to bottom of item.*
DE	*Used to delete lines from the item.*
EX	*Exits from the EDITOR.*
FD	*Deletes item from file.*
FI	*Files item and returns to TCL.*
FS	*Saves item in file and returns to EDITOR.*
F	*Files updates with previously existing item.*
G	*Directs current line pointer to go to specified line.*
I	*Used to input new lines.*
L	*Used to list a specified number of lines; or used to locate a specified string.*
ME	*Used to merge lines from another item.*
N	*Skips current line pointer over next N lines.*
P	*Used to prestore an EDITOR command.*
R	*Used to replace a number of lines.*
S	*Suppresses printing of line numbers.*
TB	*Used to set tabs.*
T	*Moves current line pointer to top line.*
U	*Moves current line pointer up.*
X	*Deletes effect of last update command.*
Z	*Sets print column limits.*
?	*Interrogates the position of the current line pointer.*

Figure A. Summary of EDITOR Commands

8 THE PROC PROCESSOR

8.1 AN OVERVIEW

An integral part of the Reality Computer System is the ability to define stored procedures called PROC's.

The PROC processor allows the user to prestore a complex sequence of TCL operations (and associated processor operations) which can then be evoked by a single command. Any sequence of operations which can be executed by the Terminal Control Language (TCL) can also be prestored via the PROC processor. This prestored sequence of operations (called a PROC) is executed interpretively by the PROC processor and therefore requires no compilation phase.

The PROC processor encompasses the following features:

- Four variable length I/O buffers
- Argument passing
- Interactive terminal prompting
- Extended I/O and buffer control commands
- Conditional and unconditional branching
- Relational character testing
- Pattern matching
- Free-field and fixed-field character moving
- Optional command labels
- User-defined subroutine linkage
- Inter-Proc linkage

Figure A shows a sample EDITOR operation which changes attribute 3 of item 11115 of file ACCOUNT to the value ABC. Figure B shows a PROC named CHANGE which will perform the exact same operation. (Note that the PROC has been written in such a manner that it will update any specified attribute in any specified item in any specified file). If, for example, the user wishes to perform the same operation shown in Figure A, then the PROC named CHANGE must be evoked as shown in Figure C.

```
:EDIT ACCOUNT 11115 (CR)
TOP
.G3 (CR)
003 100 AVACADO
.R (CR)
003 ABC (CR)
.FI (CR)
'11115' FILED.
```

Figure A. Sample EDITOR Operation

item 'CHANGE' in M/DICT

```
001 PQ
002 HEDIT
003 A2
004 A3
005 STON
006 HG
007 A4
008 H<
009 HR<
010 A5
011 H<
012 HFI<
013 P
```

Figure B. Generalized PROC Stored As Item 'CHANGE' Which Will Perform Identical Operation

```
:CHANGE ACCOUNT 11115 3 ABC (CR)
```

Figure C. Sample Execution of the PROC 'CHANGE'

8.2 PROC LANGUAGE DEFINITION

A PROC provides a means to prestore a highly complex sequence of operations which can then be evoked from the terminal by a single command.

The usage of the PROC processor is quite similar to the use of a Job Control Language (JCL) in some large-scale computer systems. The PROC language in Reality, however, is more powerful since it has conditional capabilities, and can be used to interactively prompt the terminal user. Additionally, a PROC can test and verify input data as they are entered from the terminal keyboard.

A PROC is stored as an item in a dictionary or data file. The first attribute value (first line) of a PROC is always the code PQ. This specifies to the system that what follows is to be executed by the PROC processor. All subsequent attribute values contain PROC statements that serve to generate TCL commands or insert parameters into a buffer for interactive processors (such as the EDITOR). PROC statements consist of an optional numeric label, a one- or two-character command, and optional command arguments. Some PROC commands are listed in Figure A.

PROC's operate on four input/output buffers: the primary input buffer, the secondary input buffer, the primary output buffer, and the secondary output buffer (called the stack). Essentially, the function of a PROC is to move data from either input buffer to either output buffer, thus forming the desired TCL and processor commands. At any given time, one of the input buffers is specified as the "currently active" input buffer, while one of the output buffers is specified as the "currently active" output buffer. Buffers are selected as "currently active" via certain PROC commands. Thus, when moving data between the buffers, the source of the transfer will be the currently active input buffer, while the destination of the transfer will be the currently active output buffer.

The primary input buffer contains the PROC name and any optional arguments, exactly as they were entered when the PROC was evoked. The primary output buffer is used to build the command which will ultimately be submitted at the TCL level for processing.

The secondary input buffer contains data subsequently input by the user in response to an IN command. Usually the data in this buffer will be tested for correctness and then moved to the secondary output buffer (the stack). When all desired data have been moved to the secondary output buffer, control will be passed to the primary output buffer via a P or PP command. The command which resides in the primary output buffer will be executed at the TCL level and the data in the secondary output buffer (if any) will be used to feed processors such as ENGLISH or EDITOR. When the process is completed, control returns to the PROC at which time new data may be moved to the output buffers.

Once a PROC is evoked, it remains in control until it is exited. When the PROC temporarily relinquishes control to a processor such as the EDITOR or a user-supplied subroutine, it functionally remains in control since an exit

from the called processor returns control to the PROC. TCL only regains control when the PROC is exited explicitly, or when all of the lines in the PROC have been exhausted.

For further information regarding the PROC processor, refer to the PROC and BATCH Programming Manual.

COMMAND	BRIEF DESCRIPTION
A	*Moves data argument from input to output buffers.*
B	*Backs up input pointer.*
BO	*Backs up output pointer.*
C	*Specifies comment.*
D	*Display either input buffer to terminal.*
F	*Moves input pointer forward.*
GO	*Unconditionally transfers control.*
H	*Moves text string to either output buffer.*
IF	*Conditionally executes specified command.*
IH	*Moves text string to either input buffer.*
IN	*Inputs from terminal to secondary input buffer.*
IP	*Inputs from terminal to either input buffer.*
IT	*Inputs from tape to primary input buffer.*
O	*Outputs text string to terminal.*
P	*Causes execution of PROC.*
PP	*Displays content of output buffers and executes PROC.*
RI	*Clears (resets) input buffers.*
RO	*Clears (resets) output buffers.*
S	*Sets position of input pointer and optionally selects primary input buffer.*
ST ON	*Selects secondary output buffer (stack on).*
ST OFF	*Selects primary output buffer (stack off).*
U	*Exits to user-defined subroutine.*
X	*Exits back to TCL level.*
+	*Adds decimal number to a parameter in input buffer.*
-	*Subtracts decimal number from a parameter in input buffer.*
()	*Transfers control to another PROC.*

Figure A. Some PROC Commands

9 THE REALITY CPU AND INSTRUCTION SET

9.1 AN OVERVIEW

Although the user need never be concerned with the architecture and instruction set of the Reality computer, the following section is provided for those readers who would like some information on Reality's unique structure.

The Reality CPU

The Reality Central Processing Unit (CPU) incorporates an architecture comparable to a medium scale computer. The Reality instruction set has been specifically designed for character moves, searches, compares, and all supporting operations pertinent to managing variable length fields and records.

The Reality CPU, although physically small in size and priced in the mini-computer category, has the architecture of a medium scale computer. Its main memory is core, and is expandable from 16,384 bytes to 131,072 bytes. Its full cycle operation is 1 microsecond per byte.

The virtual memory is disc which is oriented into 512-byte frames, expandable from 4,871 frames (2.5 million bytes) to 292,000 frames (300 million bytes). The CPU is capable of handling a large number of asynchronous processes, each associated with an input/output device. The Reality CPU will support up to 32 terminals (or asynchronous processes).

The CPU has 16 addressing registers and one extended accumulator for each terminal. A variable return stack accommodating up to 11 recursive subroutine calls for each terminal is also provided. By indirect addressing through any one of the 16 registers, any byte in the virtual memory can be accessed. Relative addressing is also possible using an off-set displacement plus one of the 16 registers to any bit, byte, word (16 bits), double word (32 bits), or triple word (48 bits) in the entire virtual memory.

The microprogrammed firmware contains the nucleus of Virtual Memory Operating System, the Input/Output processors, and the software instruction emulator. Complete 16-bit microinstructions are executed every 200 nanoseconds (i.e., 5 thousand instructions per second). This very fast speed ensures that the complex overhead functions take very little time away from user processing. This means fast response time and very high system throughput.

The Reality Instruction Set

The Reality Computer System has an extensive instruction set. The main features include:

- Bit, Byte, word, double-word, and triple-word operations

- Memory to memory operation using relative addressing on bytes, words, double-words, and triple-words
- Bit operations permitting the setting, resetting, and branching on condition of a specific bit
- Branch instructions which permit the comparison of two relative memory operands and branching as a result of the compare
- Addressing register operations for incrementing, decrementing, saving, and restoring addressing registers
- Byte string operations for the moving of arbitrarily long byte strings from one place to another
- Byte string search instructions
- Buffered terminal Input/Output instructions
- All data and program address references are handled by the firmware virtual memory operating system
- Operations for the conversion of binary numbers to printable ASCII characters and vice versa
- Arithmetic instructions for loading, storing, adding, subtracting, multiplying, and dividing the extended accumulator and a memory operand
- Control instructions for branching, subroutine calls, and program linkage

For further details regarding the Reality instruction set, refer to the Reality Assembly Language Programming Manual.

Microdata Corporation
17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764