REALITY® by Microdata

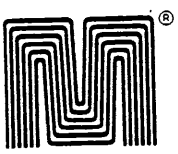
Introduction to Reality

REALITY®

Introduction to Reality

831076

Price: 5.00



Microdata Corporation

17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/250-1000 · TWX: 910-595-1764

WARNING: This equipment generates, uses, and can radiate, radio frequency energy, and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation, it has not been tested for compliance with the limits for Class A computing devices, persuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

FOREWORD

This book was produced by one of our REALITY® computer systems using RUNOFF , Microdata's word processing program. Printer output pages were used as printing masters.

RUNOFF™ includes elements of the publishing style, e.g., page size, justified right and left margins, headings, italics, pagination, centering, tabular illustrations, etc. The contents and index are automatically generated each time the document is printed, reflecting any changes made. Line art and photos have been added in spaces, or "windows", left for this purpose by RUNOFF™.

Because the complete book is stored on disc, it is easy to update the document. The stored text is corrected using the EDITOR, and a new printout is made of only the changed pages. The corrected pages are then substituted for the old pages in the printing masters.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 REALITY® Computer System

The REALITY® computer system is a generalized system for data base management. It is a complete system providing multiple users with the capability to instantly update and/or retrieve information stored in on-line data files. Users communicate via local or remote terminals with computer files that may be private, common, or security-controlled. Each user's vocabulary can be individually tailored to meet specific applications.

REALITY®systems are built of field-proven Microdata computers and peripherals, using microprograms to provide you with unrivaled performance and reliability in the medium-sized computer market.

The REALITY computer system includes the powerful, yet simple to use, ENGLISH® retrieval language, DATA/BASIC™ and PROC high level languages, the EDITOR processor, complete program development facilities and file maintenance tools, and a host of other user amenities. REALITY systems run in an on-line, multiuser environment with all system resources and data files efficiently managed by a microprogrammed virtual-memory operating system.

REALITY has advantages from every angle: system capability, multiuser performance, file management languages, ease of programming, data structure, and architectural features. REALITY's high performance and fast response time are made possible by extensive use of high-speed microprocessors which greatly reduce program execution time and system overhead. The entire REALITY computer system is unique -- one of a kind.

Microprogrammed firmware contains:

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

System software includes:

- ENGLISH, DATA/BASIC, PROC, EDITOR and ASSEMBLY languages
- Selectable/automatic report formatting
- Dynamic file/memory management
- RUNOFF™ text processing
- New SCREENPRO™ language -- an easy way to set up terminal displays
- Optional BISYNC communications

The file structure provides:

- Variable length files/records/fields
- Multivalues (and subvalues) in a field
- Efficient storage utilization
- Fast access to data items
- Selectable degrees of data security
- File size limited only by size of disc
- Item size up to 32,267 bytes

1.2 The Flexible Family of Computer Systems

The expandable REALITY family of high-performance data base management processors ranges from an economical system for first-time users with limited data processing requirements and/or experience, to the high capacity systems used by some of the largest companies in the United States.

Besides superb performance, the entire REALITY computer line offers unmatched growth advantages. As your company grows, you can add REALITY equipment to meet its increased data processing needs without the costly replacement and conversion charges usually associated with updating computer facilities. All REALITY systems are both hardware- and software- compatible. Start with REALITY. Grow with REALITY.

A typical basic system has:

- Central processing unit (CPU and cabinet)
- Mass storage disc drive
- Magnetic tape drive
- PRISM®II cathode ray tube (CRT) data terminal (up to 32)
- System printer

All REALITY systems operate in Microdata's easy-to-use ENGLISH retrieval language, as well as the more advanced DATA/BASIC and PROC, and are fully compatible with other REALITY data processing systems.

Microdata has designed a high performance REALITY system for the small to medium-sized company just entering computerized data base management. This system is a low-cost, efficient way to start. The computer and all peripherals are totally compatible with Microdata's complete REALITY line. This system has a special extended performance feature for future expansion.

At the top of the REALITY line is Microdata's most advanced microprocessing technology. Greater load capacity. Still faster data processing. More applications. All without overloading the central processing unit or degrading the speed of terminal response. The advanced system's exceptional power and adaptability provides up to 48 separate users with fingertip access to voluminous business, technical, and scientific applications that use data base management techniques.

- Complete small business computer capabilities
- Microprogrammed virtual memory operating system
- Up to 48 users and 514 million characters of file storage
- On-line file update/retrieval
- ENGLISH retrieval language
- Fast terminal response
- Multiple printer spooling
- Optional communications capabilities
- Special data management processors
- High-speed sort capability
- Small computer price
- Big computer performance
- Top to bottom computer/peripheral compatibility within the REALITY family

Figure A. REALITY System Advantages

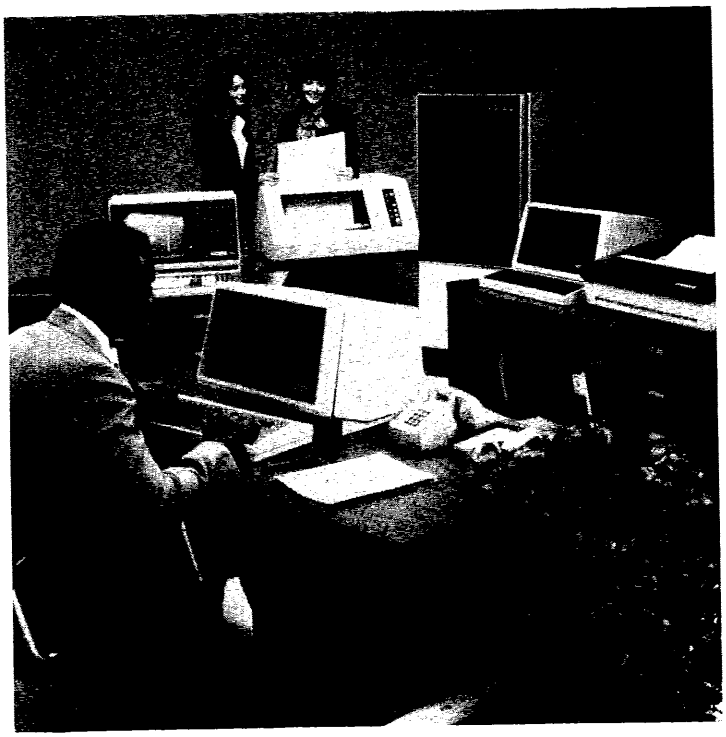


Figure B. Typical Microdata REALITY System

1.3 RÉALITY Software

Processors available on the REALITY computer system comprise the most extensive data base management software available on any minicomputer. Overviews of the processors and their typical uses follow.

ENGLISH Language

ENGLISH is a generalized data retrieval/report generator language. A typical ENGLISH inquiry consists of a relatively free-form sentence containing verbs, file-names, data selection criteria, and control modifiers. An easy-to-use, dictionary-based language that uses simplified prose statements, ENGLISH permits you to produce original reports rapidly and efficiently.

ENGLISH applications are limitless because of the ease with which output can be accessed from user files. Since nonprogrammers can master the process quickly, ENGLISH is a valuable information management tool for many people in an organization, from sales personnel to top-level executives. Its major uses are report generation and inquiry/response applications. ENGLISH also is a convenient method of producing output after file updates with DATA/BASIC or PROC software, as well as for printing one-of-a-kind reports without writing a program.

SCREENPRO

The SCREENPRO processor was developed to minimize the software gap between the establishment of data files and the creation of reports. No longer must you develop your own method of creating and processing screens to display text, inputs, validations and updates.

Because SCREENPRO requires fewer program statements, it greatly simplifies program maintenance while increasing operator and programmer efficiency. Data throughput is accelerated. A screen can be designed, displayed, tested and changed without affecting the program.

DATA/BASIC

BASIC (Beginners All-purpose Symbolic Instruction Code) is a simple, yet versatile, programming language suitable for expressing solutions to a wide range of problems. DATA/BASIC, an extension of Dartmouth BASIC, is especially easy for the beginning programmer to learn.

DATA/BASIC is the primary method of updating user files on a REALITY system. Because of its flexibility, DATA/BASIC is used for a variety of business applications including accounts payable/receivable, general ledger, inventory control, payroll, sales forecasting/analysis, order processing, invoicing, claims processing, data entry, and other projects.

With the addition of SCREENPRO, DATA/BASIC programs are even easier to write -- and run faster -- since screen handling and data validation can be removed from the program.

PROC

The PROC processor enables you to prestore a complex sequence of operations which can then be evoked by a single word command. Any sequence of operations that can be executed from the terminal can be prestored in a PROC. Although PROC is similar to the Job Control Language (JCL) used in larger computer systems, it is less cryptic and has far greater capabilities including interactive (optionally formatted) terminal prompting, input validation, printer formatting, and file input/output.

PROCs are typically used to create special user-defined functions by combining execution of DATA/BASIC programs, ENGLISH data retrieval operations, and PROC argument passing.

TERMINAL INDEPENDENT PROCESS HANDLER

The Terminal Independent Process Handler (TIPH) initiates a process on a port without an associated terminal, thus freeing the terminal for user interaction. Any terminal output (such as error messages, logon/off messages) will be placed in a spooler hold file. Although terminal I/O is not allowed, you may "stack" input in the command stream sent to the TIPH processor. This allows execution of a program which requires operator input and such input is known in advance.

EDITOR

The EDITOR permits on-line interactive modification of any item in the data base. Primarily, the EDITOR is used to create and/or modify DATA/BASIC or PROC programs. The EDITOR enters and updates text processed by RUNOFF. Particularly useful in word processing is the EDITOR's global search and replace capabilities. Performing one-of-a-kind modifications to items in user files is another EDITOR function.

RUNOFF

RUNOFF is a text processing facility offering many special features. RUNOFF processes text entered and modified with the EDITOR. RUNOFF numbers pages automatically and can print text headings and footnotes.

Another RUNOFF feature is chapter and section numbering. New chapters and/or sections may be added to a document, and the subsequent updated publication, with changes and additions, will be completely renumbered automatically. RUNOFF assembles and prints a table of contents covering all subjects, including corrected/updated copy. RUNOFF also assembles a publication index, based on specified words and phrases. RUNOFF supplies index page numbers. If new pages are added, the index is automatically updated.

RUNOFF also performs tabulations, centering, selective left/right justifications, underlining, and boldface printing.

This and all REALITY user manuals were produced by RUNOFF on a REALITY computer system.

1.4 How to Use the REALITY Manuals

This manual is 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. Here, each pair of pages discusses an individual topic. Generally the left-hand page is devoted to text, while the right-hand page presents figures referred to by the text. A pair of titles, the first naming the chapter and the second naming the topic, are at the top of each text page. Immediately below these titles is a brief summary of the material covered in the topic.

The advantage of this format will become readily apparent as you begin to use this manual. First of all, the figures referred to in the text are always conveniently in front of you at the point where the reference is made. Secondly, there is a psychological advantage knowing that when a topic is completed and the page is turned, you are done with one idea and are ready for another.

Documentation for the REALITY system includes the following manuals:

- Introduction to REALITY
- Programmer's Reference Manual
- EDITOR Programming Manual
- ENGLISH Programming Manual
- DATA/BASIC Programming Manual
- PROC Programming Manual
- SCREENPRO Programming Manual
- ASSEMBLY Language Programming Manual
- BISYNC Programming Manual
- 5750 Terminal Operator's Guide
- 2870PLUS Communications Executive Manual

In presenting general command formats and examples throughout this and other REALITY manuals, certain conventions apply. Conventions used in presenting general command formats are listed in Figure A, while conventions used in the examples are listed in Figure B.

Marginal change bars will be used in future manuals and supplements for the convenience of present REALITY users and will indicate significant additions or changes from prior REALITY publications.

Convention	Meaning
UPPER CASE	*Characters printed in upper case are required and must appear exactly as shown.*
lower case	*Characters or words printed in lower case are parameters to be supplied by you (i.e., file-name, item-id, data, etc.).*
{ }	*Braces surrounding a word and/or parameter indicates that the word and/or parameter is optional and may be included or omitted at your option.*
{ }...	*If an ellipses (i.e., three dots) follows the terminating bracket, then the enclosed word or parameter may be omitted or repeated an arbitrary number of times.*
item-list*	*An asterisk following an item-list indicates that the list of item-ids may be omitted if supplied by a previous SELECT, SSELECT, GET-LIST, or FORM-LIST statement.*

Figure A. Conventions Used in General Command Formats

Convention	Meaning
TEXT	*Shaded text represents your input.*
TEXT	*Standard text represents 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.*
<lf>	*This symbol represents a line feed.*
<c>	*This symbol specifies that the following character is a control character generated by pressing the <CTRL> key while typing the character. Also press the <SHIFT> key if the character appears on the upper half of a key top.*
_	*This is the ASCII underline character represented as a backarrow (←) on some terminals.*

Figure B. Conventions Used in Examples

2 REALITY COMPUTER SYSTEM

2.1 Overview

REALITY is a complete system of computer hardware, software and firmware specifically designed to implement cost-effective data base management. REALITY data base management systems afford two major benefits: (1) accurate and timely information access to significantly improve decision-making processes, and (2) substantially reduced clerical and administration effort to collect, store, and disseminate organizational information.

REALITY is an award-winning computer system that combines proprietary hardware and software to create an effective tool for on-line data base management. Through microprogramming, Microdata has developed a truly revolutionary on-line transaction processing system. Three major features of the high-speed microprogrammed firmware are:

- Virtual memory operating system
- Software level architecture
- Terminal input/output routines

The virtual memory operating system, long used in larger computer systems, has been impractical for minicomputers due to the extensive overhead time needed for the operating system itself. In REALITY systems, the virtual memory operating system is in firmware (high speed read-only memory) and executes many times faster than would a comparable system implemented in software. Thus, overhead time is significantly reduced.

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

The REALITY computer system is organized into 512-byte frames stored on the disc. As a frame is needed for processing, the operating system determines if it is already in main memory. If it is not, the frame is transferred from the disc unit (virtual memory) to main memory -- all automatically. Frames are written back into the disc on a "least-recently-used" basis (Figure A). The virtual memory feature of REALITY allows you to have access to a programming area not constrained by main memory, but as large as the entire available disc storage on the system.

The second feature implemented directly in REALITY firmware is the software level architecture of the machine itself, which has been expressly designed and optimized for data base management. The architecture of REALITY includes high speed instructions for character moves, searches, compares, and all supporting operations relating to the management of variable length fields and records.

The third major microcode feature is the processing of input/output (I/O) communications with on-line terminals. In most minicomputers, one of the main problems is managing the I/O from interactive terminals. As the number of terminals increases in most systems, the load on the CPU becomes overwhelming, greatly reducing response time at the terminals. However, REALITY systems efficiently manage I/O processing of on-line terminals through high-speed microprogramming. This means that data processing can proceed without interruption at each and every terminal. The firmware handles all these transactions and only interrupts the software at the completion of a block. Thus, a very large number of terminals may be connected to the Microdata REALITY system without any significant degradation in response time.

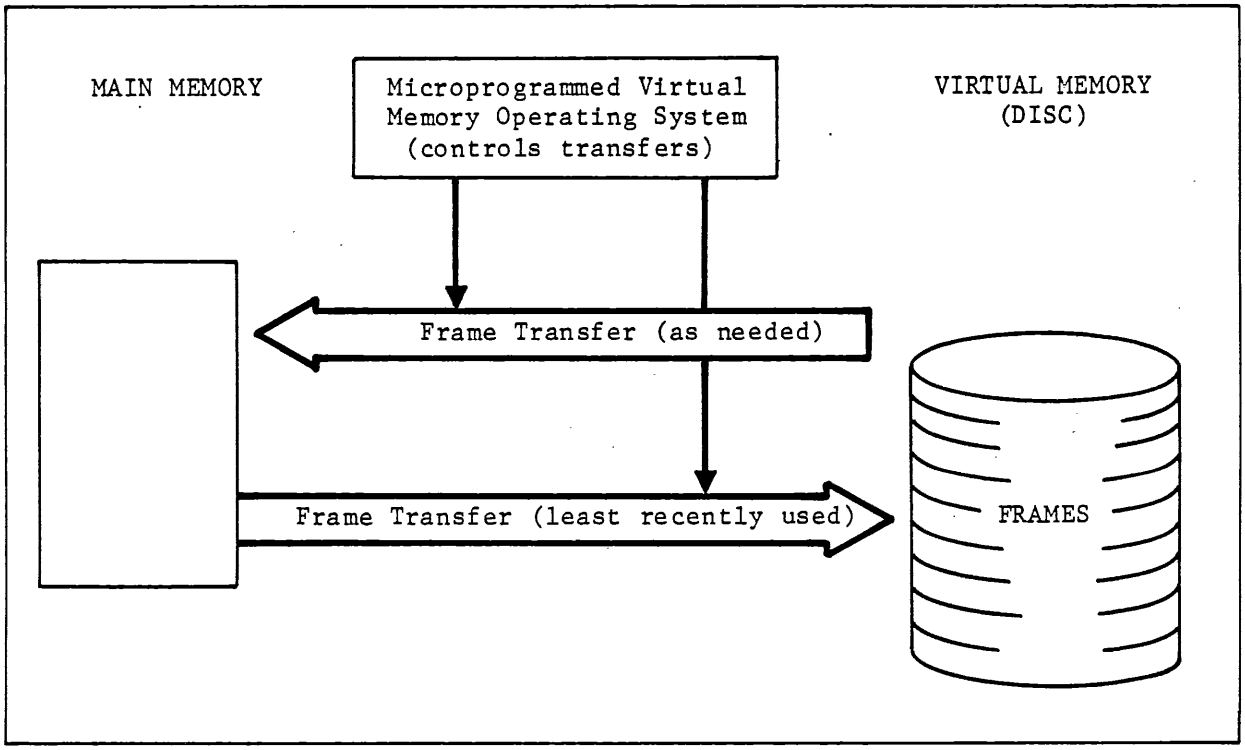


Figure A. REALITY Frame Transfers

2.2 Software Processors

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 that may be used by any or all terminals follows. Each processor presented here is described further in later chapters.

ENGLISH

ENGLISH is a generalized information management and data retrieval language. It is a dictionary-driven language with the input vocabulary contained in several dictionaries. ENGLISH offers these features:

- Freedom of word order and syntax for user inputs
- Automatic user-specified output formatting
- Sorting capabilities plus generation of statistical information
- Relational and logical operations
- English language verbs such as LIST, SORT, SELECT, COUNT, STAT, etc.

DATA/BASIC

BASIC (Beginners All-purpose Symbolic Instruction Code) is a simple yet versatile programming language suitable for solving a wide range of processing needs. Microdata's DATA/BASIC includes the following enhancements:

- Flexibility in selecting meaningful variable names
- Complex and multiline statements
- Unlimited, variable length string handling capability
- Integration with the data base file access facilities and update capabilities

SCREENPRO

SCREENPRO is a system software tool which enables the building and processing of terminal display screens. SCREENPRO'S features include:

- Increased programmer and operator efficiency and data throughput
- Simplification of program maintenance and modification
- Independence from the DATA/BASIC program
- A screen 'painter' to facilitate design and construction of screens

TIPH

The TIPH processor initiates a process on a port without an associated terminal. Features of the TIPH processor include the following:

- Increased system efficiency by freeing terminal for operator interaction
- All REALITY resources are available to TIPH process except terminal I/O
- Programs requiring operator interaction may be executed by stacking input in command stream
- All terminal output is passed to a spooler hold file
- New TIPH verbs and PROCs allow operator to control TIPH operations

PROC

The PROC processor allows you to prestore a complex sequence of operations which can then be evoked by a single word command. The PROC processor features:

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

EDITOR

The EDITOR permits on-line interactive modification of any item in the data base. The EDITOR uses the current line concept; that is, at any given time the current line can be listed, altered, deleted, etc. EDITOR features include:

- Absolute and relative current line positioning
- Merging of lines from within an item or from other file items
- Character string locate and replace
- Character, word, line or multiple line deletion, insertion, and replacement
- Input/output formatting

RUNOFF

The RUNOFF processor has extensive text processing capabilities. RUNOFF uses text prepared with the EDITOR and automatically formats the information. Commands stored within the text instruct RUNOFF to perform special functions which include:

- Automatic page numbering
- Automatic chapter and section numbering to five levels
- Automatic generation of table of contents and index
- Left and right flush tabbing (for statistical tables)
- Selectable right margin justification

Data Base Management Processors

The data base management processors generate, manage and manipulate files (or portions of files) within the REALITY system. Data base management processors include the CREATE-FILE, CLEAR-FILE, DELETE-FILE, and COPY processors.

2.3 File Hierarchy

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

This hierarchial 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. 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. 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. Within the System Dictionary are all legitimate user Logon names, passwords, security codes, and system privileges. This dictionary contains a pointer to each user's Master Dictionary.

User Master Dictionary (M/DICT)

Master Dictionaries (M/DICT) 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, PROCs, 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.

Dictionary Level File

Dictionary Level Files describe the structure of the data in associated data files. They are used by ENGLISH to define the type and format of data for output. These definitions may also be used by SCREENPRO to define the allowable format of data during input.

Data Files

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 subvalues. Thus, data may be stored in a three-dimensional variable length format.

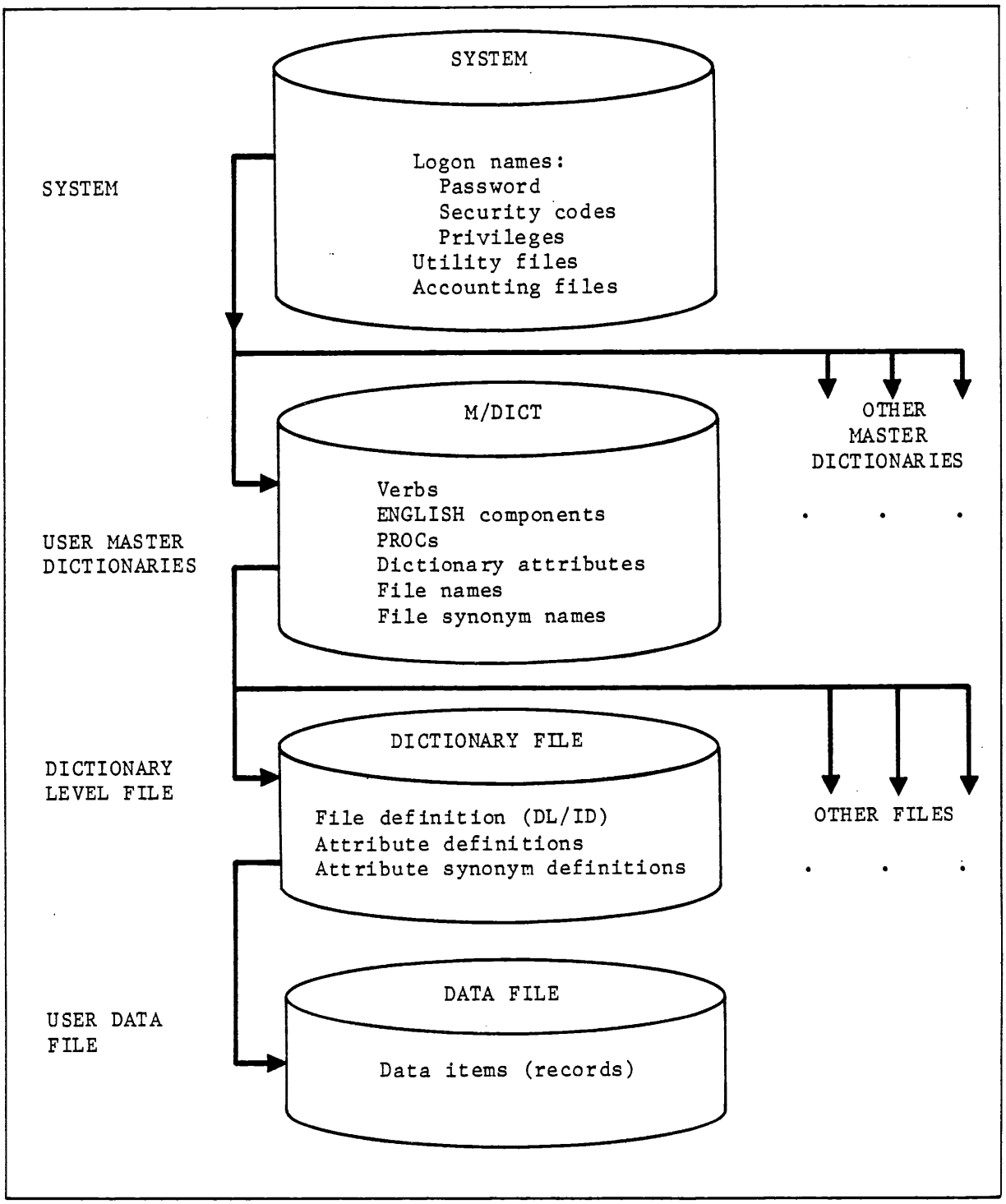


Figure A. Reality File Hierarchy

2.4 File Structure

The REALITY file access system is designed to 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 they can be accessed for both retrieval and update. A file is referenced by a file-name.

A record is called an item. Items may vary in length; the user need never identify its size. 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 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. This also frees you from having to know where things are on disc, since all information is accessed by name.

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 subvalues (also known as secondary values) separated by subvalue mark characters.

Utility processors such as COPY and the EDITOR operate at the file-item-attribute level. They make no logical distinction in definition between various attributes in an item. ENGLISH, SCREENPRO, and DATA/BASIC 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.

- The 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 subvalues.

- All files, items, attributes, values, and subvalues are variable in length and can contain any (or no) characters.
- Each item must be less than 32,267 characters long.

Figure A. REALITY File Structure Summary

2.5 Dictionaries

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

As shown in the file hierarchy, the following dictionary levels exist within the REALITY system:

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

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

- User-assigned name of the field (or attribute)
- Retrieval and update security codes
- Conversion specifications used to perform table look-ups, masking functions, etc.
- Correlative specifications used to describe interfile and intrafile data relationships
- Justification (left or right) for output purposes
- Maximum column width for printing the values
- Column headings
- Input editing and acceptance criteria

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. Dictionaries contain four types of items:

- File definition items
- File synonym definition items
- Attribute definition items
- Attribute synonym definition items

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

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

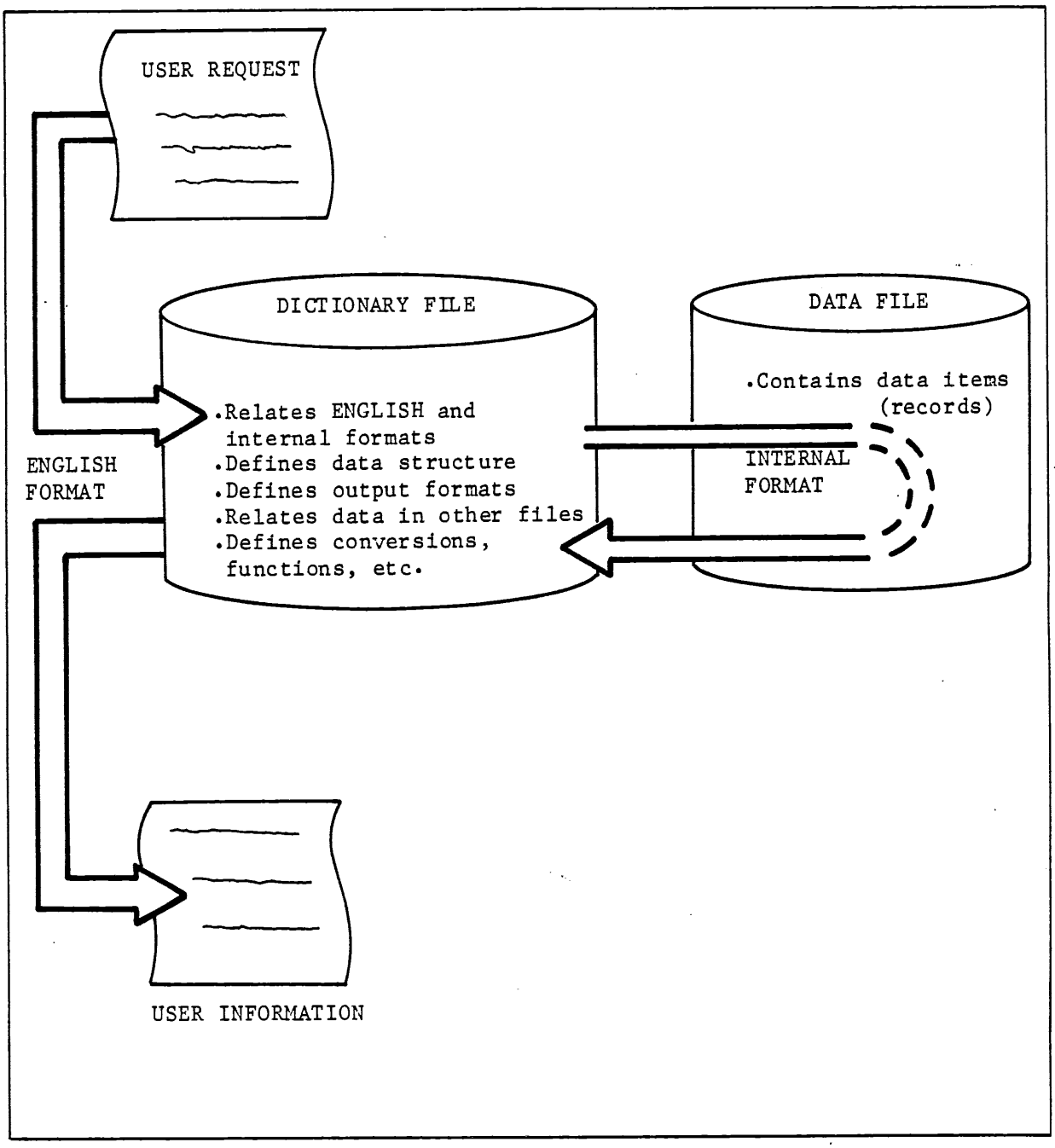


Figure A. Representation of User Data Access
Through Reality Dictionary Structure

3 TERMINAL CONTROL LANGUAGE (TCL)

3.1 Logging On and Off the System

The logon processor initiates each session by identifying valid users and their associated passwords. The logoff processor is used to end the session. These processors accumulate accounting statistics for billing purposes and also associate the users with their privileges and security codes.

Logging On to the System

You may log on to the REALITY system when the following or similar message is displayed:

```
LOGON PLEASE:
```

You then enter the name (identification) established for you in the system. If a password has also been established, you may follow your 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:        (password is not displayed on terminal)
```

REALITY validates your identification against the entries in the System Dictionary. If you have 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 you may now enter any valid command. Figure A illustrates a sample logon interaction.

Logging Off the System

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

```
*********************************************************
***       CONNECT TIME AT  time  =  n  MINUTES        ***
***       CPU MS. =  m          DISC READS = d        ***
***           NUMBER OF ACTIVATIONS =  a              ***
*********************************************************
***       LOGGED OFF AT  time          ON  date       ***
```

where "time" is the current time, "n" is the number of minutes of connect time, "m" is the number of CPU milliseconds, "d" is the number of disc reads initiated by you, "a" is the number of times your terminal communicated with the CPU, and "date" is the current date. All of this information is stored in a log used for system accounting.

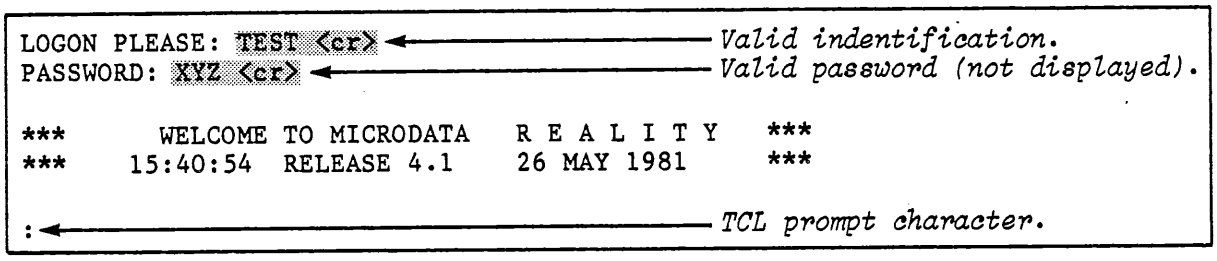


Figure A. Sample Logon Interaction

```
:OFF <cr>

************************************************************
***          CONNECT TIME AT 22:30:53 = 10 MINUTES       ***
***    CPU MS. = 5183            DISC READS = 123         ***
***           NUMBER OF ACTIVATIONS = 597                 ***
************************************************************
***     LOGGED OFF AT 22:31:03      ON 26 MAY 1981       ***
```

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 TCL by a single input statement, and return control to TCL when processing is complete. TCL prompts you 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 linefeed key is pressed, 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 may not contain any other verbs. Selected verbs are listed in Figure A.

REALITY has the ability to customize your system vocabulary. Since verbs reside in each Master Dictionary (M/DICT), your vocabulary may be changed without affecting other users. In addition, an unlimited number of synonyms may be created for each verb. This flexibility allows your capabilities to be either expanded or limited, by adding or deleting verb entries from your Master Dictionary.

REALITY operates in the full-duplex mode of communication with each terminal. Full-duplex means that data may be simultaneously transmitted in both directions 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. Thus, you are assured that the data character displayed on the terminal is the same data character stored in the computer.

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

VERB	DESCRIPTION
ACCOUNT-RESTORE	*Adds a new account (from tape) to an existing system.*
BASIC	*Compiles a DATA/BASIC program.*
BLOCK-PRINT	*Prints expanded text on the system printer.*
CATALOG	*Catalogs a DATA/BASIC program.*
CHARGES	*Displays current computer usage since logon.*
CHARGE-TO	*Charges the current usage to a specific account.*
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.*
ED	*Evokes the EDITOR processor.*
EDIT	*Evokes the EDITOR processor.*
ENTER	*Evokes the Screen Builder function of SCREENPRO.*
ENTER-DICT	*Evokes the Dictionary Builder function of SCREENPRO.*
FORM-LIST	*Creates item-list from item-ids stored in an item.*
GET-LIST	*Retrieves an item-list saved by a previous SAVE-LIST.*
GROUP	*Provides file usage statistics on groups.*
ISTAT	*Generates a file hashing histogram for a file.*
ITEM	*Displays the FID to which the item hashes.*
LIST	*Generates a formatted output of selected items.*
LIST-LABEL	*Formats selected items and values into a mailing list.*
LOGTO	*Allows the user to log from one account to another.*
MESSAGE	*Provides for intrasystem communications.*
OFF	*Evokes LOGOFF processor, ending the current session.*
RUN	*Executes a DATA/BASIC program.*
RUNOFF	*Evokes the RUNOFF text processor.*
SAVE-LIST	*Saves an item-list created by a SELECT or SSELECT.*
SEL-RESTORE	*Restores files or items from a file-save tape.*
SELECT	*Selects items for use by a subsequent processor.*
SET-DATE	*Sets the system date.*
SET-TIME	*Sets the system time.*
SORT	*Performs a sorted LIST.*
SORT-LABEL	*Performs a sorted LIST-LABEL.*
SP-ASSIGN	*Assigns options to a print job and/or job to a form queue.*
SP-EDIT	*Examines a closed print job.*
SP-STATUS	*Displays the current spooler status.*
SSELECT	*Performs a sorted SELECT.*
STAT	*Counts, averages, and sums a specified attribute.*
T-DUMP	*Dumps specific items and files to tape.*
T-LOAD	*Loads items and files from tape.*
T-READ	*Dumps contents of tape to the output device.*
TERM	*Sets terminal characteristics.*
TIME	*Displays 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 DATA BASE MANAGEMENT AND UTILITY PROCESSORS

4.1 Data Base Management

Data base management processors generate, manage, and manipulate files (or portions of files) within the REALITY system. These processors include CREATE-FILE, CLEAR-FILE, DELETE-FILE, and COPY processors.

CREATE-FILE Processor

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

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.

DELETE-FILE Processor

The DELETE-FILE processor deletes 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.

COPY Processor

The COPY processor copies an entire file (or selected items from the file) to the terminal, printer, magnetic tape unit, 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

Figure A presents a number of examples illustrating the use of the file management processors. For further information, 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 the data section of file XYZ.*
:DELETE-FILE (DICT INV) <cr>	*Deletes dictionary section of INV file.*
:DELETE-FILE (FAB) <cr>	*Deletes the 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 printer.*
:COPY TEST * <cr> TO:<cr>	*Copies all items in the TEST file to the terminal.*

Figure A. Sample Usage of File Management Processors

4.2 Utilities

The REALITY Utility processors give the system extensive utility capabilities.

The REALITY computer system includes a large number of utility processors which provide such capabilities as:

- Magnetic tape unit controls
- Mathematical functions
- Multiple printer spooling control
- Formatted file save/restore functions
- Binary save/restore functions
- File statistics
- Creation of user-accounts
- Setting of terminal characteristics
- Block (enlarged) printing
- Virtual memory dumping
- Interuser message facilities
- Bootstrapping and coldstart
- System accounting
- System status and usage information

A few examples of utility processor usage are shown in Figure A. For further information, refer to the REALITY Programmer's Reference Manual.

EXAMPLE	EXPLANATION
:T-ATT <cr>	*Attaches the magnetic tape unit to the terminal issuing command.*
:T-FWD 10 <cr>	*Moves magnetic tape forward 10 records.*
:T-READ (4-6) <cr>	*Bypasses next 3 magnetic tape records; dumps the 4th, 5th, and 6th records to the terminal, and positions the tape at the beginning of the 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 the magnetic tape unit to the Beginning Of Tape mark.*
: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>	*Produces block-print of characters AB12 on line printer.*
:MESSAGE ROD HELLO THERE <cr>	*Transmits message "HELLO THERE" to user ROD.*
:FILE-SAVE <cr>	*Produces a logical save of the system files onto tape.*
:BINARY-SAVE <cr>	*Produces a byte-for-byte disc transfer onto tape.*
:LIST-FILE-STATS <cr>	*Prints a file statistics report using the statistics currently in the STAT-FILE.*

Figure A. Sample Usage of Utility Processors

4.3 REALITY Spooler

The REALITY computer system is a multiuser system which permits you to perform processing operations with complete independence from other users. However, output devices can only support one user at a time. The REALITY spooler is sophisticated system software which resolves such conflicts by providing simultaneous output to a maximum of four system printers, one tape drive, and as many ports as each system configuration will allow.

The spooler allows multiple users to share REALITY's system peripherals. Output reports are "spooled" to the disc unit which frees the terminal to initiate some other processing task. When the designated peripheral device and the "despooling" processor become available, the report is automatically "despooled" from the disc unit to either a magnetic tape, a system printer, or a terminal port.

To fully understand the features and advantages of the REALITY spooler, the terms "print job" and "form queue" must be defined.

Print Jobs

Individual reports to be spooled to an output device are called "print jobs". A print job may be any item created by the EDITOR, a PROC, an ENGLISH statement, or a DATA/BASIC program. Before a print job can be output, it must be assigned to a form queue.

Form Queue

A form queue is a "list" of print jobs waiting to be spooled. Print jobs are assigned to a form queue via the SP-ASSIGN verb. Once created, the form queue may be assigned to an output device (a system printer, a tape unit, or a terminal port).

Figure A lists the major spooler features and Figure B describes the advantages of the 4.1 spooler.

FEATURES

- Spooler supports simultaneous output to a maximum of four system printers, one magnetic tape unit and as many ports as the system configuration will allow.
- You may restart a print job on another device if the printer malfunctions.
- You may create as many form queues as needed.
- You may move one or more print jobs from one form queue to another.
- You may specify number of copies to be printed.
- You may examine a closed print job prior to printing.
- SP-STATUS display shows current status of all form queues.
- SP-JOBS display shows current status of all print jobs, open and closed.
- SP-STATUS display alerts you when spooler has aborted.
- System may be restarted without loss of closed hold files.

Figure A. Features of REALITY Spooler

ADVANTAGES

- You control spooling activities by creating form queues, assigning print jobs to selected queues and outputting those jobs on the desired output device.
- SP-STATUS and SP-LISTQ displays inform you of current spooler status.
- You may "prioritize" print jobs within a form queue.
- Edit feature allows you to selectively print individual print jobs.
- Print jobs may be listed on the terminal screen prior to printing.
- Multiple output devices are now supported to evenly distribute work load.
- Remote printers need no operator intervention.

Figure B. Advantages of REALITY Spooler

5 ENGLISH LANGUAGE PROCESSORS

5.1 Overview

ENGLISH is a user-oriented data retrieval language used to access 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. System vocabulary can be individually tailored to your 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, interfile retrieval operations, display format, and more.

ENGLISH selectively retrieves information and generates reports automatically. Output reports (which normally appear on the terminal but may optionally be transmitted to the printer) are automatically formatted for you by the REALITY system. You may sort the output into any defined sequence, and total attributes by using control breaks.

ENGLISH features include:

- 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 a 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

Figures A through C illustrate typical ENGLISH inquiries.

```
:LIST ACCOUNT WITH BILL-RATE = ".30" NAME ADDRESS BILL-RATE <cr>

PAGE 1                                                  11:08:37  12 FEB 1980

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

11115      D R MASTERS          100 AVOCADO              0.30
11085      A B SEGUR            101 BAY STREET           0.30
11040      E G MCCARTHY         113 BEGONIA              0.30
11050      J R MARSHECK         125 BEGONIA              0.30
11020      J T O'BRIEN          124 ANCHOR PL            0.30
11095      J B STEINER          124 AVOCADO              0.30
11110      D L WEISBROD         106 AVOCADO              0.30
11015      L K HARMAN           118 ANCHOR PL            0.30
11105      C C GREEN            112 AVOCADO              0.30
11090      J W JENKINS          130 AVOCADO              0.30
23030      L J DEVOS            201 CARNATION            0.30

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                                                  11:15:47  12 FEB 1980

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 ENGLISH Language Primer

You may form ENGLISH sentences which specify desired data retrieval functions. The ENGLISH retrieval language is modified natural English; formats for sentences are simple, yet very general. The ENGLISH processors, together with the use of dictionaries, permit inputs to be stated in the terminology natural to each application.

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 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 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 explicitly stated using attribute names found in the file dictionary. If none are specified in the sentence, the implicit attribute synonym list in the file dictionary will be used to specify displayed fields.

Selection-criteria determine which items in the file will be used. If nothing is specified, then all items will be used. One or more direct references may be made by specifiying the item-id in single quotes. A conditional retrieval may be specified by using a WITH or IF clause. All items in the file will be examined, 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.).

Miscellaneous connectives may be used to modify the effect of the verb or to alter the display format.

Figure B illustrates ENGLISH sentences.

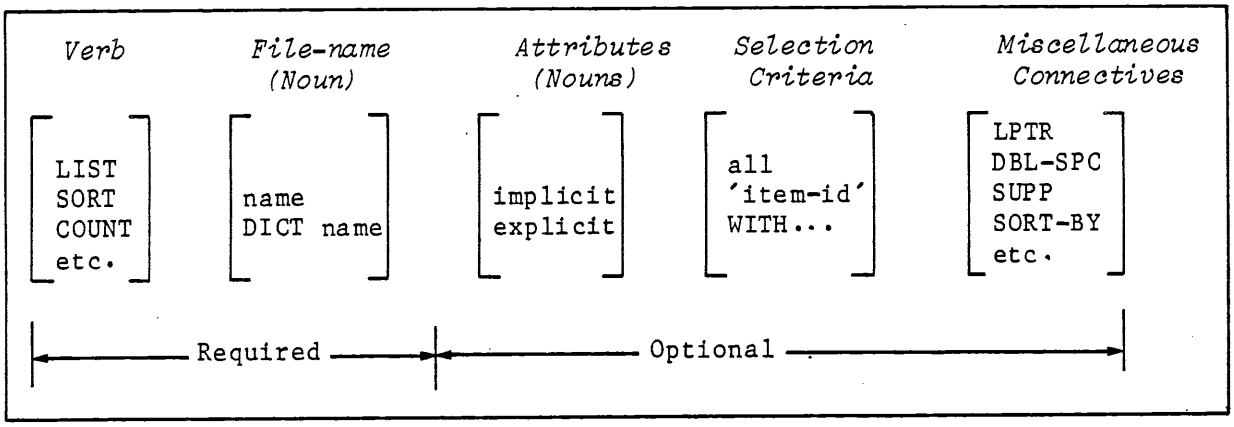


Figure A. Generalized Grammatical Structure of an ENGLISH Sentence

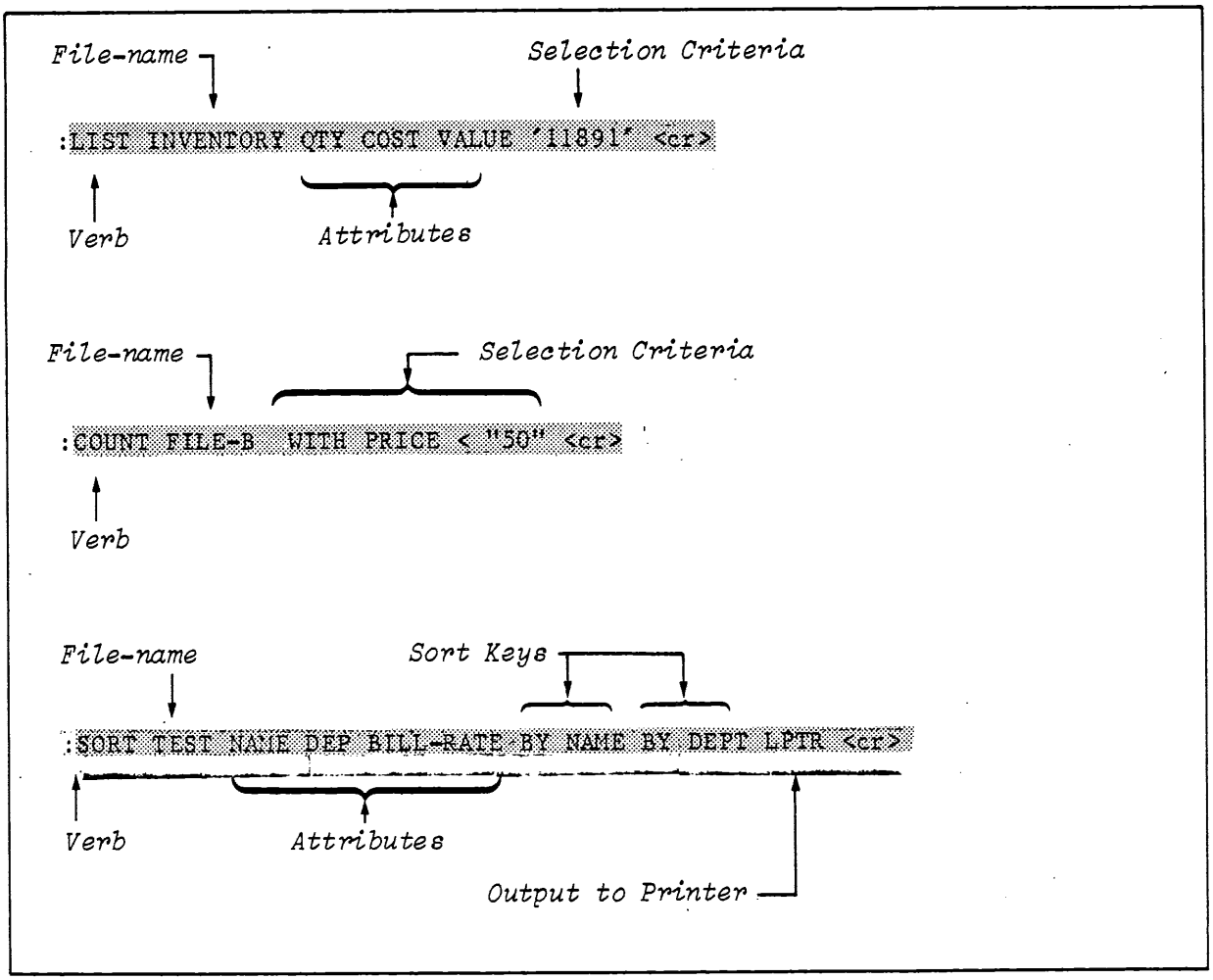


Figure B. Sample ENGLISH Sentences

5.3 ENGLISH Language Verbs

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

LIST and SORT; LIST-LABEL and SORT-LABEL

THE LIST and SORT verbs are used to generate formatted output. LIST simply lists the selected output, while SORT orders the output in a specified sorted order. Generated output will be formatted into a columnar output, if possible, taking into account the maximum size of 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 noncolumnar output will be generated with the attribute names down the side and associated attribute values to the right. LIST and SORT will automatically format multivalued attributes and subvalues. Subtotalling is possible 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. LIST-LABEL and SORT-LABEL may be used to generate formatted mailing labels, index cards, etc.

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 generate a sum for a specified attribute. Additionally, the STAT verb provides a count and average for the specified attribute. The outputs generated by these verbs are the derived statistics. Figure C illustrates the use of the SUM verb.

SELECT and SSELECT

The SELECT verb chooses a set of items. These selected items are then made available, one at a time, to certain REALITY processors. For example, you can select items meeting certain criteria and pass them to a DATA/BASIC program for updating. 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, T-LOAD, I-DUMP, ISTAT, HASH-TEST, and CHECK-SUM

T-DUMP and I-DUMP verbs allow you to selectively dump your dictionaries and data files to the magnetic tape or to the terminal. The T-LOAD verb loads files from tape. ISTAT and HASH-TEST verbs provide file hashing histograms. The CHECK-SUM verb determines if the data in a file has been changed.

```
.LIST ACCOUNT '23080' '23090' NAME ADDRESS START-DATE CURR-BALNC <cr>

PAGE 1                                              11:19:58  12 FEB 1980
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 (Noncolumnar 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 DATA/BASIC LANGUAGE PROCESSOR

6.1 Overview

Microdata's 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 solving 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 up to 32,267 characters
- 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
- Internal and external subroutines
- Math functions
- DATA/BASIC symbolic debugger

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 (*).

```
        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 uses the 'INPUT USING' statement (Lines 10 & 20) which          *
*    evokes 'SCREENPRO'.  All input validation (such as part number     *
*    format, etc.) are controlled by the previously created screen.     *
*    The first 'INPUT USING' brings up the screen display and prompts   *
*    for the part number which is the item-id of an item in the 'INV'   *
*    file.  Next, the program attempts to read that part from the file. *
*    The attributes 'DESC' and 'QTY' (if present) are then displayed    *
*    when the screen is re-entered with the second 'INPUT USING'        *
*    (Line 20).  The location and format of these attributes were       *
*    automatically derived when the names 'DESC' and 'QTY' were used    *
*    when the screen was created with the 'painter'.  Finally, the item *
*    (possibly modified) is written to the file and the program repeats. *
*                                                                       *
*************************************************************************
*
***       --Open the 'SCREENS' file that contains the desired screen.
     OPEN 'SCREENS' ELSE PRINT "CANNOT OPEN 'SCREENS' FILE"; STOP
***       --Read the compiled screen definition item.
     READ SCREEN FROM "#INV.UPDATE" ELSE PRINT "SCREEN NOT ON FILE"; STOP
***       --Open the data section of the Inventory file.
     OPEN 'INV' ELSE PRINT "CANNOT OPEN 'INV' FILE"; STOP
***       --Prompt for part number
10   INPUT PART.NUMBER USING SCREEN SETTING STEPNO ELSE STOP
***       --If just a carriage return, then clear screen and stop
     IF PART.NUMBER<1> = "" THEN PRINT CHAR(12); STOP
***       --Attempt to read that item from the 'INV' file.
     READ ITEM FROM PART.NUMBER<1> ELSE ITEM=""; * If not there, set to null
***       --Display and update item.
20   INPUT ITEM USING SCREEN,ITEM AT STEPNO ELSE GOTO 10
***       --Write the item to the file and repeat
     WRITE ITEM ON PART.NUMBER<1>
     GOTO 10
     END
```

Figure B. Sample DATA/BASIC Program Which Queries an Inventory File using SCREENPRO

6.2 DATA/BASIC Language Definition

A DATA/BASIC program consists of DATA/BASIC statements, which may contain variables, constants, expressions, and intrinsic functions.

A DATA/BASIC program is 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 those statements unique to the DATA/BASIC language.

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	DESCRIPTION
ABS	*Returns an absolute value.*
ASCII	*Converts a string from EBCDIC to ASCII.*
CHAR	*Converts a numeric value to an ASCII character.*
COL1	*Returns column position preceding FIELD-selected substring.*
COL2	*Returns column position following FIELD-selected substring.*
COS	*Returns the cosine of an angle.*
COUNT	*Returns the number of occurrences of a substring.*
DATE	*Returns the current internal date.*
DELETE	*Deletes an attribute, value, or subvalue from a dynamic array.*
EBCDIC	*Converts a string from ASCII to EBCDIC.*
EXP	*Returns 'e' raised to a power.*
EXTRACT	*Returns an attribute, value, or subvalue from a dynamic array.*
FIELD	*Returns a delimited substring.*
ICONV	*Provides for REALITY input conversion.*
INDEX	*Returns the column position of substring.*
INSERT	*Inserts an attribute, value, or subvalue into a dynamic array.*
INT	*Returns an integer value.*
LEN	*Returns the length of a string.*
LN	*Returns the natural logarithm (base 'e') of a numerical value.*
MOD	*Returns the modulo of two numerical expressions.*
NOT	*Returns logical inverse.*
NUM	*Tests for numeric value.*
OCONV	*Provides for REALITY output conversion.*
PWR	*Returns a variable raised to a power.*
REPLACE	*Replaces an attribute, value, or subvalue in a dynamic array.*
RND	*Generates a random number.*
SEQ	*Converts an ASCII character to a numerical value.*
SIN	*Returns the sine of an angle.*
SPACE	*Generates a string containing blanks.*
SQRT	*Returns the square root of a numerical expression.*
STR	*Generates the specified string.*
TAN	*Returns the tangent of an angle.*
TIME	*Returns internal time of day.*
TIMEDATE	*Returns external time and date.*
TRIM	*Removes trailing blanks from a string expression.*
@	*Controls terminal cursor.*

Figure A. Summary of DATA/BASIC Intrinsic Functions

STATEMENT	DESCRIPTION
BEGIN CASE	*Allows conditional selection of statements.*
CALL	*Passes control to an external DATA/BASIC subroutine.*
CHAIN	*Passes control to another DATA/BASIC program.*
CLEAR	*Initializes all variables to zero.*
CLEARFILE	*Clears the specified file.*
COMMON	*Allows variable data to be passed between programs.*
DEBUG	*Causes the DATA/BASIC debugger to be entered.*
DELETE	*Deletes a specified file item.*
EQUATE	*Declares a symbol equal to a variable or literal.*
FOOTING	*Causes the specified string to be printed at the bottom of each page.*
HEADING	*Prints a page heading.*
LOCATE	*Returns the position of an attribute, value or subvalue within a dynamic array.*
LOCK	*Sets an execution lock.*
LOOP	*Provides for structured program loops.*
NULL	*Specifies a non-operation.*
OPEN	*Selects a file for subsequent I/O.*
PAGE	*Pages output device and prints heading.*
PRECISION	*Selects the precision of calculations.*
PRINTER	*Selects the 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 value.*
REWIND	*Rewinds magnetic tape.*
SELECT	*Builds a list of item-ids for use by READNEXT.*
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.*

Figure B. Summary of Unique DATA/BASIC Statements

6.3 Creating, Compiling, and Executing DATA/BASIC Programs

A DATA/BASIC program, 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, you issue the EDIT verb. The general command format is:

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

You may then begin entering your 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 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. This makes the program "reentrant", enabling all users to share a single copy in memory. 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.

The DATA/BASIC debugger is entered when (1) fatal errors are produced (such as an attempt to divide by zero), (2) the <BREAK> key is pressed, or (3) a DEBUG statement is executed. At this time, you may display and change the value of any variable or array, suppress printing, set single-step execution, set break conditions, trace variables, and modify program flow of control.

```
: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>
***
LINE 003  [B0]  COMPILATION COMPLETED

:RUN PROGRAMS TESTING <cr>
THIS IS
A TEST

:
```

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

```
:RUN PROGRAMS UPDATE (D) <cr>      Run program with 'D' option to break
                                   before the first line is executed.
*E1                                Execution halted before line 1.
*BCITY=IRVINE <cr> +               Break when "CITY" equals "IRVINE".
*G <cr>                            Continue execution ("GO").

*B1                                Break condition satisfied.
*/CITY <cr> = IRVINE=TUSTIN <cr>   Display value of CITY. Change to TUSTIN.
*$ <cr> 7                          Display line about to be executed.
*G <cr>                            Continue execution.

*E10                               Execution break due to a DEBUG command in
                                   the program.
*TTAX.RATE <cr> +                  Set trace variable TAX.RATE.
*E1 <cr>                           Set for single step execution.
*G <cr>                            Continue execution.
*E11                               Execution break due to E1;
TAX.RATE=.10                       value of TAX.RATE is displayed.
*END <cr>                          End execution.

:                                  TCL prompt character.
```

Figure B. Sample DATA/BASIC Symbolic Debugger Session

7 SCREENPRO PROCESSOR

7.1 Overview

SCREENPRO is a system software tool which builds and processes screens.

Until recently, a major software gap existed between the creation of a file and the generation of reports via the dictionary based ENGLISH processor. End users had to develop their own methods to create and process screens (displaying text, inputting, validating, and updating data). These methods generally resulted in a cumbersome program; program development and documentation time could be extensive. Modifications required major program changes. SCREENPRO uses a dynamic technique to design and construct screens; it is written in ASSEMBLY language to increase data processing speed. With SCREENPRO, you may change the screen display without modifying the program.

SCREENPRO consists of three major sections: a Screen Builder, a Dictionary Builder, and a Screen Processor. The Screen Builder allows you to dynamically construct and test screens. Its output is a screen definition item which will be used by a DATA/BASIC program to display, input, validate, and update data. Documentation is easier because the information can be listed on the printer along with a 'snapshot' of the screen display. Program maintenance is simplified because most of the information is stored in dictionaries and screen definition items.

The Dictionary Builder prompts you to create the necessary Attribute Definition Items. To further define the data, the dictionary item format has been expanded to contain SCREENPRO input and ENGLISH output controls.

The Screen Processor is the interface to DATA/BASIC which uses the screen definition item to display the current data, and to prompt for, validate, and update the data to be passed back to the DATA/BASIC program. Instead of interpreting a lengthy series of compiled DATA/BASIC statements (as with most end-user programs), the Screen Processor works directly in ASSEMBLY language. The output of the processor is the data, formatted per your specifications, in a dynamic or dimensioned array accessible to the DATA/BASIC program.

SCREENPRO includes these features:

- Requires fewer program statements
- Increases programmer efficiency
- Increases operator efficiency
- Simplifies program maintenance and modification
- Allows independence from the DATA/BASIC program
- Increases data throughput
- A screen 'painter' to facilitate the design and construction of screens
- A step display to more completely define the step parameters
- A screen tester with a dummy data buffer
- Screen branching, which controls the prompting sequence
- Reentry facility, to combine SCREENPRO and DATA/BASIC operations
- Data validation
- User error message facility
- Associated steps facility
- Automatic literal input
- Global input conditional facility
- Fixed input location facility
- Completed screen printout on the printer

Figure A. Features of SCREENPRO

7.2 Screen Builder

The Screen Builder designs and constructs screens. Its output is a screen definition item.

The Screen Builder is used to easily design and construct screens for entering and/or updating data. Using the Screen Processor function, the Builder displays a variety of menus and prompt messages to guide you. The Screen Builder is entered at the TCL level via the ENTER verb. The Builder will then display the main menu, enabling you to select one of the six functions: Screen Definitions, Screen Painter, Steps Display, Screen Compiler, Screen Tester, and Screen Printout. The Screen Definitions consist of three global parameters which can be used by every step in the screen definition item. The three parameters are: Dictionary Filename(s), Fixed Input Location, and Global Input Conditional. Dictionary Filename(s) lists the files which contain the attribute definition items for the data items to be updated. The Builder can use the information in the attribute definition items to specify some of the parameters in the Steps Display. Fixed Input Location and Global Input Conditionals are inserted into each step automatically, saving you time.

The Screen Painter allows you to type text and indicate I/O positions directly on the terminal, thereby deriving the screen coordinates automatically. The information is inserted into the Steps Display.

The Steps Display shows the 35 parameters which define each step of the display and allow them to be modified. The information here will be condensed by the Builder into values of the corresponding attributes in the source screen definition item.

The Screen Compiler condenses the source screen definition item into an object screen definition item. This object screen definition item is used by the Screen Processor to display the screen and update the data passed from the DATA/BASIC program. If there is an error in the source screen definition item, the compiler will generate default data for the object screen definition item that will be passed to the Screen Tester. The default data is not accessible to you outside the Screen Builder.

The Screen Tester uses a temporary data buffer to test the compiled screen definition item without the use of a DATA/BASIC program. The processed test data is passed to the temporary buffer which can be displayed on the terminal.

The Screen Printout function prints a condensed listing of the Step parameters, divided by steps. It also prints a 'snapshot' of the painted screen. The printout reduces the need for, and simplifies the creation of documentation.

```
          * * *  MICRODATA REALITY SCREEN BUILER  * * *

               1) SCREEN DEFINITIONS

               2) PAINT SCREEN

               3) ENTER STEPS

               4) COMPILE SCREEN

               5) TEST SCREEN

               6) PRINT SCREEN ON LPTR


     ENTER FUNCTION DESIRED  (OR END):
```

Figure A. Primary Menu for the Screen Builder

```
          I N V E N T O R Y    U P D A T E

1)   PART NUMBER              ;1,PART

2)   DESCRIPTION              ;2,DESC

3)   QUANTITY                 ;3,QTY

4)   COST                     ;4,COST

IS ALL DATA CORRECT (FI/EX/#)          ;OK
```

Figure B. Sample Painted Screen for Inventory Update

7.3 Screen Processor and the Dictionary Builder

The Screen Processor is the interface to DATA/BASIC which uses the screen definition item to display the screen and current data (if any) on the terminal, and to prompt for, validate, and update the data passed back to the DATA/BASIC program. The Dictionary Builder creates and updates attribute definition items.

The Screen Processor uses the compiled object screen definition item produced by the Screen Builder to format the screen display(s), and to enter and process data. It is a system software processor, written in ASSEMBLY language, and is evoked by either the INPUT USING or the MATINPUT USING DATA/BASIC statement.

The processor uses two buffers: 1) a scratch buffer to receive the data passed to it by the DATA/BASIC program and entered by the user, and 2) a data buffer into which the processed data is placed. When the processor is evoked, it saves the data passed to it in the scratch buffer and initializes the data buffer to include null attributes for fields defined in the screen definition item but not included in data passed to it. When the processor returns control to the DATA/BASIC program, it passes back the data stored in the data buffer.

The Dictionary Builder is a software processor which creates attribute definition items. The Dictionary Builder is accessed by the ENTER-DICT verb. The Dictionary Builder will display the 16 currently used attributes which may be defined in a dictionary item, and will prompt for the information which may be included in an attribute definition item. The Dictionary Builder will then expand the information into the actual attribute definition item format, which consists of 20 attributes (four of which are reserved for future expansion).

For further information regarding the REALITY SCREENPRO processor, refer to the SCREENPRO Programming Manual.

```
                    CUSTOMERS DICTIONARY BUILDER

DICTIONARY ITEM-ID ?NAME <cr>


    1. D/CODE        S
    2. A/AMC         3
    3. S/NAME        CUSTOMER NAME
    4. S/AMC
    5. L/RET
    6. L/UPD
    7. V/CONV
    8. V/CORR
    9. V/TYPE        T
    10. V/MAX        20
    11. V/MIN
    12. I/LENGTH
    13. F/REALLOC
    14. I/CONV
    15. I/VALIDATE   (0A)
    16. I/ERROR


                                                ENTER COMMAND:_
```

Figure A. Dictionary Builder Format for the Item NAME in DICT CUSTOMERS

8 EDITOR PROCESSOR

8.1 Overview

The EDITOR is a REALITY processor which 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, PROCs, ASSEMBLY programs, 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.

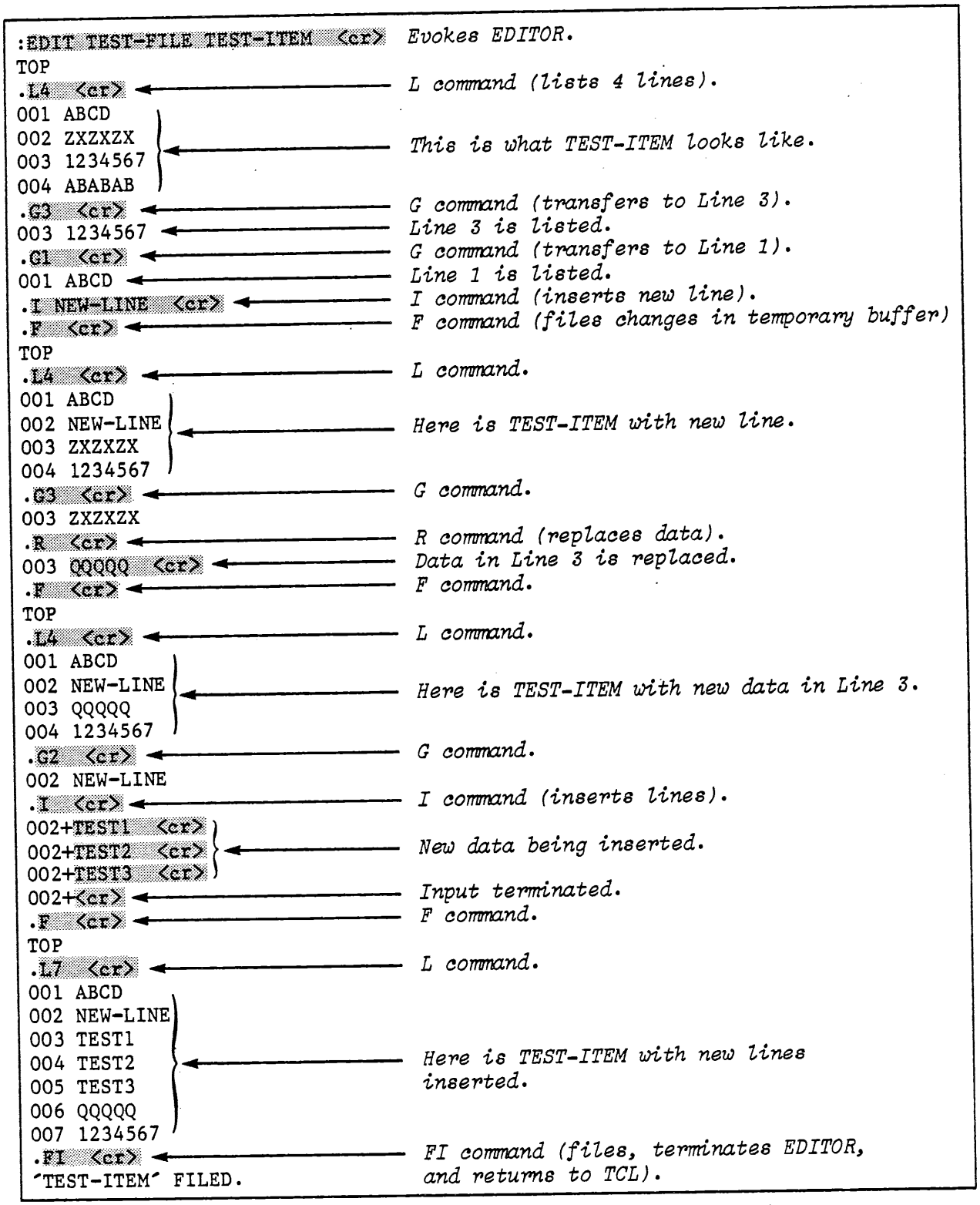


Figure A. Sample EDITOR Session

8.2 EDITOR Language Definition

The EDIT verb evokes the EDITOR processor. 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:

```
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" indicates 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 copied to the second buffer.

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

For further information regarding the REALITY EDITOR, refer to the EDITOR Programming Manual.

COMMAND	DESCRIPTION
A	*Executes last Locate (L) command again.*
B	*Moves current line pointer to bottom of item.*
C	*Displays the line column positions.*
DE	*Deletes (a) line(s) from the item.*
EX	*Returns to TCL without filing the updated item.*
F	*Files updates in a temporary buffer.*
FD	*Deletes item and returns control to TCL.*
FI	*Files the item and returns to TCL.*
FS	*Files item and returns control to EDITOR.*
G	*Moves the current line pointer to the specified line.*
I	*Used to input new lines.*
L	*Lists lines; or Locates strings.*
ME	*Merges lines from within the item or from another item.*
N	*Skips current line pointer over the specified number of lines.*
P	*Prestores EDITOR commands.*
R	*Replaces text on the specified lines.*
S	*Suppresses printing of line numbers.*
T	*Moves the current line pointer to the first line.*
TB	*Sets EDITOR tab markings.*
U	*Moves current line pointer up.*
W	*Displays the specified line plus the 21 lines that proceed it.*
X	*Deletes the effect of the last update command.*
Z	*Sets print column limits.*
?	*Displays the position of the current line pointer.*
?I	*Displays the file-name and item-id of the item.*
?P	*Displays the prestored command without executing it.*
?S	*Displays the size of the item in bytes.*

Figure A. Summary of EDITOR Commands

9 PROC LANGUAGE PROCESSOR

9.1 Overview

An integral part of the REALITY computer system is the ability to define stored procedures called PROCs.

The PROC processor allows you to prestore a complex sequence of Terminal Control Language (TCL) operations (and associated processor operations) which can be then be evoked by a single command. Any sequence of operations which can be performed from a terminal 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.

PROC 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
- File access and updating
- Optional command labels
- PROC calculations
- User-defined subroutine linkage
- Formatted terminal displays and printouts
- 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 exactly the 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. For example, if you wish to perform the same operation shown in Figure A, then the PROC named CHANGE can be evoked as shown in Figure C.

```
:EDIT ACCOUNT 11115 <cr>
TOP
.G3 <cr>
003 100 AVOCADO
.R <cr>
003 ABC <cr>
.FI <cr>
'11115' FILED.
```

Figure A. Sample EDITOR Operation

```
Item 'CHANGE' in M/DICT

001 PQN
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'

9.2 PROC Language Definition

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

Using 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 PQN. 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.

PROCs operate on four buffers: the primary input buffer, secondary input buffer, 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 tiem, 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 activated. 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 the data entered in response to an IN command. Usually the data in this buffer will be tested for correctness and then moved to an output buffer. When all desired data have been moved to the output buffers, 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 interactive processors such as DATA/BASIC or the 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 activated, it remains in control until it is exited. When the PROC temporarily relinquishes control to another processor (such as the EDITOR), 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 commands have been processed.

COMMAND	DESCRIPTION
A	*Moves data argument from input to output buffers.*
B	*Backs up input pointer.*
BO	*Backs up output pointer*
C	*Specifies comment.*
D	*Displays either input buffer to terminal.*
F	*Moves input pointer forward.*
F;	*Specifies an arithemetic calculation.*
F-CLEAR	*Clears the specified file buffer.*
F-DELETE	*Deletes an item from the specified file.*
F-OPEN	*Opens the specified file to a file buffer.*
F-READ	*Reads an item from the specified file.*
F-WRITE	*Writes an item to the specified file.*
GO	*Unconditionally transfers control.*
GOSUB	*Transfers control to a local subroutine.*
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.*
L	*Produces formatted printer output.*
MV	*Copies data from buffer to buffer, or from buffer to register.*
O	*Outputs text string to terminal.*
P	*Causes execution of TCL commands in output buffer.*
PP	*Displays content of output buffers and executes TCL commands in output buffer.*
RI	*Clears (resets) input buffers.*
RO	*Clears (resets) output buffers.*
RSUB	*Returns control from an internal subroutine.*
RTN	*Returns control from an external PROC subroutine.*
S	*Positions input pointer; selects primary input buffer.*
ST ON	*Selects secondary output buffer (stack on).*
ST OFF	*Selects primary output buffer (stack off).*
T	*Produces terminal output; displays buffer values.*
TR ON	*Activates the PROC trace function.*
TR OFF	*Deactivates the PROC trace function.*
X	*Exits back to TCL level.*
+	*Adds decimal number to parameter in input buffer.*
-	*Subtracts a number from a value in input buffer.*
(file-name item-id)	*Transfers control to another PROC. Control does not return to the initiating PROC.*
[file-name item-id]	*Transfer control to another PROC. Control returns to the initiating PROC.*

Figure A. Summary of PROC Commands

10 TERMINAL INDEPENDENT PROCESS HANDLER (TIPH)

10.1 Overview

A "process" on a REALITY system is defined as the session which occurs between logging on and logging off. Each process is associated with a line (or port), and (normally) each line is associated with a terminal. The Terminal Independent Process Handler (TIPH) initiates a process on a port without an associated terminal. Any terminal output (such as error messages, logon/logoff messages, etc.) will be placed in a spooler hold file, the hold file number being assigned by the spooler processor. TIPH may be used for those tasks which do not require operator responses, such as file-saves, massive updates, etc.

TIPH provides a kind of foreground/background capability that allows processes to run without the need of a terminal. "Resident" programs, such as a task scheduler or a transaction logger may use TIPH. TIPH will increase the efficiency of the system by freeing the terminals for their intended purpose, which is user interaction.

A TIPH process is like any other process, except that it is not associated with a terminal. It has its own workspace, and has available to it all of the resources of the REALITY system, with the exception of terminal I/O. You cannot access the system through a terminal after the line associated with it has been logged on by a TIPH process.

Although terminal I/O is not allowed, you may "stack" input in the command stream sent to the TIPH processor. This allows programs to be run which require operator input, where the input responses are known in advance. The processor will pass one command string to each input required by the program.

Certain lines may be designated as TIPH-only lines by the system hardware. Normal lines (those normally associated with a terminal) may be designated as being available for use by TIPH by using the PH-ALLOCATE verb. If a TIPH process is activated without specifying a line number, TIPH will automatically select any available line (either TIPH-only or designated via PH-ALLOCATE).

A PH-HISTORY file exists in the SYSTEM dictionary, which contains all of the information concerning each TIPH process. This information is updated as the process continues, relating information concerning the status of the TIPH process(es).

The verbs and PROCs listed in Figure A are used to communicate with the TIPH processor.

Command	Explanation
PH-ALLOCATE	*Designates a port as available for use by TIPH.*
PH-DELETE	*Removes the specified port from the list of ports available to the TIPH processor.*
PH-KILL	*Aborts the designated TIPH process.*
PH-LINES	*Lists the lines currently available on which to activate a TIPH process.*
PH-RESUME	*Reactivates a temporarily halted TIPH process.*
PH-START	*Initiates a TIPH process.*
PH-STATUS	*Displays the current status of all TIPH processes listed in the PH-HISTORY file.*
PH-SUSPEND	*Temporarily halts a TIPH process.*

Figure A. Summary of TIPH Verbs and PROCs

11 RUNOFF TEXT PROCESSOR

11.1 Overview

RUNOFF is a text processor used to create a wide variety of textual material.

RUNOFF prepares such written correspondence as memos, manuals, proposals, etc. RUNOFF formats and prints text previously entered with the EDITOR (although the information may be placed in a file by a program, such as in an automated documentation system). Interspersed within the text are RUNOFF commands that instruct the text processor to perform various functions such as centering, line spacing, etc. Formatted output may be directed to the terminal, printer, or other output device.

Textual material may be easily edited and corrected using the EDITOR and then reprinted with RUNOFF. Material may be inserted or deleted without restriction. Unchanged text need not be retyped. RUNOFF can combine separate copy into a single document and insert duplicate material into different reports.

The camera-ready masters for this manual and other REALITY documentation were prepared by a REALITY system using RUNOFF.

RUNOFF includes these features:

- Easy updating of existing documents
- Automatic/selectable formatting
- User-specified headings and footings
- Automatic page numbering (if desired)
- Automatic chapter and section number assignment (to 5 levels)
- Automatic table of contents generation based on section and chapter numbers
- Automatic index generation based on user-specified words and phrases
- Automatic margin justification (if desired)
- Left or right flush tabbing (invaluable for statistical tables)
- Ability to read additional text and/or commands from other file items
- Upper and lowercase capability on either upper/lowercase or uppercase only terminals
- Accepts terminal input for entry of variable information
- Overprint (boldface) printing, text underlining, and centering

The facing page shows text that is left and right justified. This can be defeated by changing one command. Margin and tab settings can also be changed without affecting the material already typed. For example:

> Textual material may be easily edited and corrected using the EDITOR and then reprinted with RUNOFF. Material may be inserted or deleted without restriction. Unchanged text need not be retyped.

Another important feature incorporated in RUNOFF is automatic tabbing of subsequent lines when printing indented blocks. For example:

1) This block of material has a tab control character before the "1)" which right justifies it at position 5. Another tab control appears before the word "This". The remainder of the text is entered free-form without tabbing. RUNOFF automatically lines up the subsequent lines.

    a. This makes adding or deleting text very easy since you do not have to retype part of a paragraph in order to maintain the indented block format.

11.2 Command Definition

RUNOFF processes information stored in a REALITY file item. This information consists of both text and RUNOFF commands.

All RUNOFF commands begin with a period (.), and are always on a line by themselves (i.e., the commands do not appear on text lines). Multiple commands may be placed on a line.

RUNOFF processes information in one of two ways. In the "fill" mode, RUNOFF prints a word at a time until no more words can fit on the line. If the justify option is chosen, RUNOFF adds spaces at random between words to create a right justified margin. This mode is typically used for text in sentence form (the text you are reading now was processed in the "fill and justify" mode). Using this mode, text may be entered free-form without concern for line endings. In the "nofill" mode, RUNOFF processes one line from the item at a time. This is primarily used for tables and figures (Figures A and B were processed in the "no-fill" mode).

Figure A presents a brief summary of RUNOFF commands. Figure B shows a portion of the item that produced this page.

.BEGIN PAGE (.BP)	*Causes a BREAK and a page advance.*
.BREAK (.B)	*Outputs any partially filled line before processing the next line.*
.CAPITALIZE SENTENCES (.CS)	*Capitalizes the first word of each sentence.*
.CENTER (.C)	*Centers the following text line.*
.CHAIN ({DICT} file-name item-id)	*Chains to the specified text file.*
.CHAPTER title	*Numbers and formats chapter headings.*
.CONTENTS	*Prints the table of contents accumulated by preceding CHAPTER and SECTION commands.*
.CRT	*Directs output to the user's terminal.*
.FILL (.F)	*Fills a line without overflowing it.*
.FOOTING	*Prints the next line on bottom of each page.*
.HEADING	*Prints the next line at the top of each page.*
.INDENT n (.I n)	*Indents the next line by 'n' spaces.*
.INDENT MARGIN n (.IM n)	*Positions the left margin.*
.INDEX text	*Stores 'text' in an index list.*
.INPUT	*Reads next line from the terminal.*
.JUSTIFY (.J)	*Justifies the right margin.*
.LEFT MARGIN n	*Sets the left margin.*
.LINE LENGTH n	*Sets the line length.*
.LOWER CASE (.LC)	*Causes all letters (except those specified) to be output in lower case.*
.LPTR	*Directs output to the system printer.*
.NOCAPITALIZE SENTENCES (.NCS)	*Resets the CAPITALIZE SENTENCES mode.*
.NOFILL (.NF)	*Resets the FILL mode.*

Figure A. Summary of RUNOFF Commands

.NOJUSTIFY (.NJ)	*Resets the JUSTIFY mode.*
.PAGE NUMBER n	*Sets the current page number.*
.PARAGRAPH n (.P n)	*Formats text into paragraphs (with the first line indented).*
.PRINT INDEX	*Prints the sorted index of words generated by the INDEX command.*
.PRINT	*Displays the next line on the user's terminal.*
.READ ({DICT} file-name item-id)	*Reads and processes the text item indicated.*
.SECTION level title	*Numbers the next section at depth 'n'.*
.SET TABS n{,n}...	*Sets tab positions.*
.SKIP n (.SK n)	*Outputs 'n' blank lines.*
.SPACING n	*Sets the line spacing.*
.STANDARD	*Resets the default parameters.*
.UPPER CASE (.UC)	*Prints characters as they are (lower or upper case).*

Figure A. Summary of RUNOFF Commands (continued)

```
001 .BP.J
002 .READ FOOTING.L
003 .INDEX ^RUNOFF command definition^
004 .SECTION 2 Command Definition
005 .SK 2
006 RUNOFF processes information stored in a REALITY file item.
007 This information consists of both text and RUNOFF commands.
008 .SK 2
009 All RUNOFF commands begin with a period (.), and are always
010 on a line by themselves (i.e., the commands do not
011 appear on text lines).
012 Multiple commands may be placed on a line.
013 .SK
014 RUNOFF processes information in one of two ways.
015 In the "fill" mode, RUNOFF prints a word at a time until
016 no more words can fit on the line.
017 If the justify option is present, RUNOFF adds spaces at
018 random between words to create a right justified margin.
019 This mode is typically used for text in sentence form
020 (the text you are reading now was processed in the "fill
021 and justify" mode).
022 Using this mode, text may be entered free-form without concern for
023 line endings.
024 In the "nofill" mode, RUNOFF processes one line from the
025 item at a time.  This is used primarily for tables
026 and figures (Figures A and B were processed in the "no-fill" mode).
```

Figure B. Sample RUNOFF Source Item

12 REALITY CPU AND INSTRUCTION SET

12.1 Overview

Although you need never be concerned with the architecture and instruction set of the REALITY computer, the following section is provided for those interested in REALITY's unique structure.

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 and priced in the minicomputer range, has the architecture of a medium scale computer. The main memory is expandable from 16,384 bytes to 524,288 bytes (depending on model). Its full cycle operation is 1 microsecond per byte for core memory equipped machines and 800 nanoseconds for MOS memory systems.

The virtual memory is disc and is oriented into 512-byte frames, expandable from 19,487 frames (10 million bytes) to 1,029,600 frames (514.8 million bytes). The CPU is capable of handling a large number of asynchronous processes, each associated with an input/output device. The advanced REALITY CPU will support up to 48 terminals (or asynchronous processes).

The CPU has 16 addressing registers and one extended accumulator for each terminal. Also provided is a variable return stack accommodating up to 11 recursive subroutine calls for each terminal. 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 offset 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 the virtual memory operating system, input/output processors, and the software instruction emulator. Complete 16-bit microinstructions are executed every 200 nanoseconds (i.e., 5,000 instructions per second). This very fast speed ensures that the complex overhead functions occur without interrupting user processing. This means fast response time and high system throughput.

The REALITY CPU also provides automatic power-fail/restart capability which allows the system to continue with the last instruction executed after power is restored. MOS memory machines are provided with a battery backup feature to retain memory contents in the event of a power failure.

The REALITY Insturction Set

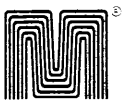
The REALITY computer system has an extensive instruction set. The main features include:

- 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 which permit setting, resetting, and branching on the condition of a specific bit
- Addressing register operations for incrementing, decrementing, saving, and restoring the addressing register
- Byte string operations for moving arbitrarily long byte strings from one place to another
- Byte string search instructions
- Buffered terminal input/output instructions
- All data and program address references, 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.

(THIS PAGE INTENTIONALLY LEFT BLANK)

INDEX



Microdata Corporation

17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/250-1000 · TWX: 910-595-1764