Burroughs

B 5500

TIME SHARING SYSTEM

TERMINAL USER'S GUIDE

Burroughs

B 5500

TIME SHARING SYSTEM

TERMINAL USER'S GUIDE

Burroughs Corporation
Detroit, Michigan 48232

Printed in U. S. America

6-69

1038205

Burroughs Corporation believes the program described in this manual to be accurate and reliable, and much care has been taken in its preparation. However, the Corporation cannot accept any responsibility, financial or otherwise, for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Systems Documentation, Sales Technical Services, Burroughs Corporation, 6071 Second Avenue, Detroit, Michigan 48232.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

INTRODUCTION

Recent improvement in the productivity of current generation computer systems has directed attention not only to making the computer operation more efficient and self-regulating, but also to distributing the solutions more dynamically to problems within a given system. One of the most innovative approaches to the question of quick "turnaround" time has evolved as the Burroughs time sharing concept. Time sharing, or time slicing, is Burroughs technique for sharing the resources of the B 5500 computer system concurrently among an extensive number of simultaneous computer users. The objective of the time sharing system is to provide a response to every user within a given time interval.

In the Burroughs B 5500 Time Sharing System, programs are successively rolled into memory from the rapid access head-per-track disk, are executed for a short time interval, and are then rolled out again. Each user of the system therefore receives a "slice" of computer time and, within a brief cycle of time, all programs receive a response from the B 5500.

The following advantages are gained from the Time Sharing System (TSS):

a. The first and most important effect of the B 5500 TSS is improved personal effectiveness in problem solving. The individual user now has instantaneous access to the computer without inhibiting its ability to service other users at the same time.

b. On-line programming and testing becomes a practical reality. The B 5500 TSS eliminates the traditional bottleneck to new application development, that is, computer access for testing and debugging.

c. The full power of commonly based application programs is available to all users of the system thereby

reducing programming effort and file maintenance.

d. Additional convenience and effectiveness are a by-product of locating the time sharing terminal right in the user's office. This, in essence, places the full power of a large-scale scientific and information processing system at the user's fingertips.

e. The full advantage of powerful programming languages designed to service every user need:

   1) COBOL for servicing business and information processing.

   2) ALGOL and FORTRAN IV for servicing the needs of scientific and engineering applications in their most advanced state.

   3) BASIC for developing an immediate system "usability" for people who have never used a computer before... easy-to-learn and a highly effective problem-solving tool.

f. Allowing a user these dynamic system features is a set of generalized software, including compilers of the above mentioned languages, led by an operating system, a system commonly known as the Master Control Program (MCP). The MCP maintains overall control of the system allowing the user to communicate with it through the Command and Edit language processor (CANDE) as described later in this manual.

g. The time sharing operation can co-exist with normal, multiprocessing "production" jobs back at the central computer site. The B 5500 TSS eliminates the need for separate "special purpose" computers to service the

needs of information processing, scientific work, and conversational time-sharing. All of these functions are accomplished in a single advanced computer system, the Burroughs B 5500.

The Burroughs B 5500 Time Sharing System incorporates, as realities, the most advanced state-of-the-art techniques in the industry. This manual is intended as a functional description of those technqiues and their application to the problem solving requirements of business, education, science, and industry.

SECTION 1
SYSTEM DESCRIPTION

GENERAL.

The Burroughs B 5500 embodies a successful implementation of a spectrum of advanced design objectives. Burroughs approach to computer system design has been to totally integrate both hardware and software. The success of this approach was realized through a policy of cross-training the computer design teams. Hardware engineers were educated in the intricacies of software architecture and software designers learned the subtleties of hardware design. These people were then merged into a single design team. This merger resulted in a system which featured:

a. Compiler Oriented Hardware.
b. Multiprocessing Master Control Program.
c. Dynamic Resource Allocation.
d. Automatic Program Segmentation.
e. Homogeneous Hardware Organization.

COMPILER ORIENTED HARDWARE.

Traditionally, most competent systems programmers have rejected the use of high-level compiler languages because the ease and convenience of their use demanded too great a sacrifice in compiling and operating efficiency.

The Burroughs B 5500, by contrast, has proved to be a complete unity of high-level languages with efficient hardware operation. It is a computer that, for all practical purposes, evolved without a need for assembly languages. Its operating environment involves the use of ALGOL, COBOL, FORTRAN, and BASIC.

A technique called "Polish notation" was adopted as the basic architecture for both the hardware and the software. Polish notation simplifies problem program expressions by eliminating the need for conventional rules of arithmetic precedence and "bracket grouping" of values within an expression. Using Polish notation, the

expression: E:= (A+B) /C becomes: AB + C/ E:=

The rule that applies is: follow two arithmetic values with the operation designated to work with those values. Thus every mathematical operator automatically works on the most recently obtained pair of operands.

High level Polish notation is the "machine language" of the Burroughs B 5500 System. The approach which makes Polish notation possible and powerful in its hardware implementation utilizes the Push Down Stack structure of these systems to allow programs to be considered as strings of elements which correspond to values, literals, and operators in the compiler languages.

The Push Down Stack is the hardware implementation of the Polish string software design architecture. In hardware terms, it consists of two registers (at the top of the stack) and a contiguous area of memory that permits the stack to extend beyond the registers. Working with a Polish string, the stack allows an arithmetic operator to work with whatever happens to be in the registers at a moment in time.

This unity of software and hardware design results in dramatic efficiencies in program compilation and execution. The algorithm for program compiling is simply a matter of translating the source program into a Polish string object program, which is also a high-level language representation. The effect is to produce extremely fast one-pass compilations of high-level languages. In addition, the hardware is designed to efficiently execute the specific object program code produced. This removes the need for storing and retrieving intermediate problem expressions, and allows straight-line non-redundant object execution.

The full power of Polish notation and the Push Down Stack is brought to bear on problem solving in both the Time-Sharing and the multiprocessing batch mode of operation on the Burroughs B 5500. The

machine language of the system is compiler-oriented, thereby removing the need for wasteful analysis of the program source language, thus speeding compilations and producing highly efficient machine code.

MULTIPROCESSING MASTER CONTROL PROGRAM.

The prime criterion for successful implementation of a comprehensive Master Control Program operating system is a clear identification of the need for such a capability. Burroughs identified that need in the late 1950's. At that time the design objective was set to develop a computer system capable of controlling its own resources and scheduling work on a dynamic basis. In this manner, a number of programs should feasibly be concurrently executed in less time than if they were executed serially. Moreover, at any moment in time more of the computer's physical resources should be in use by the multiplicity of programs than any single job could utilize. The result of this has been highly efficient utilization of the B 5500's problem solving facilities.

The Master Control Program operating system has provided an advanced multiprogramming/multiprocessing capability for Burroughs users for a number of years. Since many of the features of the standard MCP have been chosen as the nucleus of time-sharing systems in the industry, Burroughs enjoys a significant advantage in its proven capability. These features have been subjected to the most rigorous field testing possible, and they have been producing practical results in customer installations for years.

DYNAMIC RESOURCE ALLOCATION.

A Time Sharing System is perhaps the most sensitive to down time of all possible computer operating modes. When a system simultaneously servicing a multiplicity of remote users suffers a condition causing the system to go out of operation, the whole user community suffers immediately and dramatically. Burroughs has a powerful answer to this need for continuous, uninterrupted remote service, dynamic resource allocation (fail soft or graceful degradation).

Dynamic resource allocation was originally developed as an integral part of the multiprocessing MCP. In past years, it has served two key functions:

a. First, it has allowed new components to be added to the system with an immediate speed-up in throughput. As new devices are added to the B 5500, the Master Control Program immediately and automatically recognizes that it now has "more machine." More important, the MCP immediately begins using the new resources to process work faster than before. Further, this is accomplished without having to regenerate the operating system, re-write the programs, recompile the programs, or do anything except connect the components to the configurations.

b. Second, it allows the B 5500 to continue producing results in the face of circumstances which might ordinarily prove disastrous to other systems. Much as it senses and reacts to additions to the system resources, it also recognizes deletions from the system and reacts by re-routing the work around the trouble spot. A memory module, an I/O channel, a peripheral device, or even a central processor (in a two processor system) can be taken off-line and the rest of the system can be made to continue producing results.

The Master Control Program continually checks the status of the total system, and it automatically fits object programs to those system components that are available at any moment in time. Comprehensive but compact "availability tables" keep the MCP informed of the number of tapes, disks, printers, I/O channels, and other devices that are available and ready; and changes in the status of the system are easily detected because of advanced hardware/software integration.

As soon as the MCP has the status information, it can allocate the system's resources accordingly. The moment a Burroughs user plugs in a new memory increment, or another I/O channel, or even an additional central processor, the program that originally required 10 minutes may now take as little as 3 or 4 minutes to run to completion. More important, when a component is taken off-line for any reason, the system continues to function. The B 5500 continues responding with problem solutions to the whole network of remote users.

The B 5500 Time Sharing System effectively alleviates the most sensitive criticism of time sharing: accessibility and guaranteed response on a continuous stream of problem-solving demands.

AUTOMATIC PROGRAM SEGMENTATION "PAGING."

The computer industry has set as its objective, freeing the problem programmer as much as possible from the physical limitations of computer hardware. This objective has taken many forms and suffered various degrees of success and failure, particularly when the question of "practicality" is raised in connection with a particular feature and/or computer system. Burroughs B 5500 Time Sharing System addresses virtually all of the facilities required to satisfy the "hardware independence" objective, and provides practical and workable answers. A number of these facilities have been explored above.

Automatic Program Segmentation is one of the most powerful of Burroughs' third generation "realities." It provides a creative solution to two questions:

a. How can a multiprocessing computer system efficiently use its memory all of the time?

b. How can the computer handle programs that exceed its physical memory capacity?

Burroughs B 5500 Time Sharing System provides "virtual memory" for unlimited program size. Automatic Segmentation has, in fact, been a working reality on the standard B 5500 MCP for years, and it has been a practical reality. This capability is based on the use of compilers as program pre-processors. These Burroughs compilers logically segment all programs at compilation time and create a detailed record of how the segmenting was performed.

The effect of Automatic Program segmentation is to allow the programmer complete freedom from the physical limitations of hardware memory in developing application solutions.

HARDWARE ORGANIZATION.

The hardware architecture which makes multiprocessing and dynamic resource allocation practical is centered around two exchanges ... the memory exchange and the input/output exchange. These central controls operate as small integrated computers charged with responsibility for allocating the total resources of the system to job processing.

Each processor (on a dual processor system) accesses memory through the exchange. Likewise, the maximum of four input/output channels can access memory without interfering with the processors. The channels in turn, can access any of the peripheral devices through the input/output exchange.

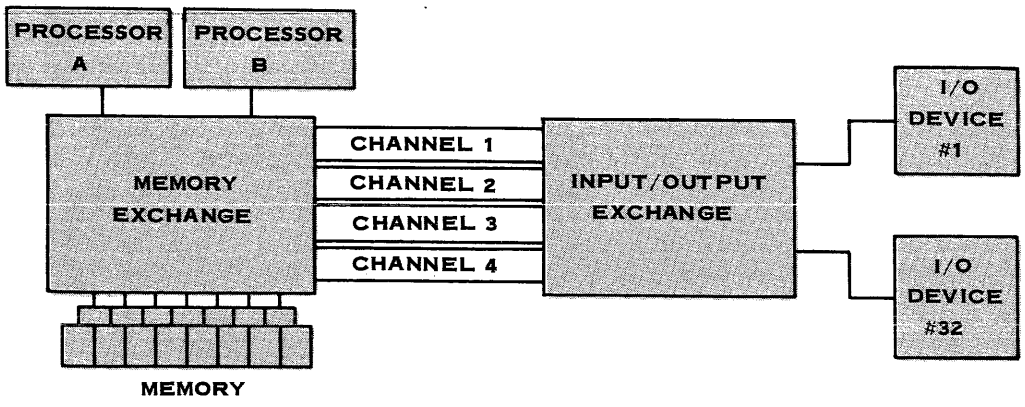


Figure 1-1. Hardware Organization

The exchanges contain the logic and registers to "float" the demand for accesses to whichever channel path happens to be free to handle it. With these floating channels and memory paths, both processors and all four channels can be executing memory accesses simultaneously. Therefore, under control of the Master Control Program operating system, the data paths and configuration of the system are continuously optimized to answer the demands of the moment. This is accomplished automatically, without programmer awareness or concern.

A maximum configuration of the Burroughs B 5500 allows for:

a. Two central processing units.
b. Eight independent memory modules (each 4,096 words).
c. Four floating input/output channels.
d. Sixteen magnetic tape drives.
e. Two billion characters of on-line disk storage.
f. Two line printers.
g. Two card readers.
h. Two paper tape readers.
i. One card punch.
j. One paper tape punch.
k. One supervisory console.
l. 240 communications line buffers.

The B 5500 Data Communications Network can expand dynamically to meet growing time sharing requirements. And the expansion or contraction of the system is accomplished without reprogramming or recompiling. The Time Sharing Master Control Program balances the current total program mix against the available system configuration, for the most efficient operation under continuously changing conditions.

SECTION 2

SOFTWARE SYSTEMS

GENERAL.

The software systems of the Burroughs B 5500 Time Sharing System consists of the Master Control Program, the Command and Edit language processor, and the programming languages.

THE MASTER CONTROL PROGRAM (MCP).

The primary purpose of the MCP is the efficient distribution of system resources among the users. It is capable of controlling its own resources as well as scheduling work on a dynamic basis. In this manner, a number of programs should feasibly be executed concurrently in less time than if they were executed serially. Moreover, at any moment in time, more of the computer's physical resources should be in use by the multiplicity of programs than any single job would normally utilize. The result of this has proven highly efficient utilization of the B 5500's problem solving facilities.

Memory in the B 5500 TSS is divided into two distinct areas by a "fence" or pre-determined boundary. The MCP and the Command and Edit language processor are run on one side of the fence. Jobs which are initiated by remote users are run on the other side of the fence and are swapped in and out of core; i.e., they are temporarily suspended and rolled out of core to disk to make room for other remote jobs. There are five main conditions which cause remote programs to be rolled out:

a. The program is input limited, that is, it is waiting for data which has not yet been entered from the remote unit. While the user transmits additional data needed to continue processing, the program resides on disk and makes its memory resources available to other users.

b. The program is output limited, that is, it has generated

enough data to fill the disk buffer area assigned to it. Therefore, the program turns its resources back to the system while the current data is transmitted to the user. The next time it receives its share of computer time, it can again generate additional data for transmission to the remote terminal.

c. The program has used the slice of time allocated to it without encountering an input or output limited condition.

d. The program has reached completion.

e. The program is forced out to make room for an entering or re-entering job.

When a job first enters the mix, it is assigned to an area in core which minimizes conflict with other active jobs and is then given an immediate time slice. If necessary, jobs in core are rolled out to make room for it, unless they too are getting their first time slice. In that case, the new job is placed at the head of the queue of jobs waiting for a time slice.

Some of the other important features of the MCP are:

a. Priority scheduling lets the individual user program govern its relationship and importance to other jobs being concurrently executed in the system. With multi-level priorities, each job's importance is recognized by the system and is given attention based on its ranking at the moment. In addition, job priorities can be changed dynamically, while processing, to answer more urgent demands on the system.

b. Automatic file recognition allows all program and data files to be addressed by name rather than by physical input/output unit. In this manner, jobs can be allowed to

use any device that happens to be available at the moment. This greatly facilitates the initiation and execution of jobs in a community of remote users and, at the same time, permits practical multiprocessing to be accomplished on regular production jobs.

c. True device independence allows magnetic tape and disk to be used as backup or "pseudo" devices for card readers, punches, and printers. This means that jobs are not delayed waiting for access to a busy peripheral unit. Instead, such jobs automatically (or at operator option) stream out to a backup device. It also means that traditional "satellite computer" operations such as card-to-tape, card-to-disk, tape-to-print, and the like, are now accommodated on a single computer system.

d. Comprehensive logging is a standard by-product of MCP processing. The log can provide a breakdown of time as follows:

1) By job - the time used for processing and input/output.

2) By job - the time used by each physical device to execute the job. This includes the processor, card readers and punches, printers, magnetic tapes, and magnetic disks.

3) By device - the amount of time each physical unit was used by the system ... and a summary of errors incurred by each unit.

4) By type of job - the amount of time and number of times the system was used for compilations (for each compiler), executions, and runs.

5) By job mix - the amount of time the system was run with a given number of jobs currently in the mix.

The accounting facilities of the Time Sharing System will then generally ensure that all time and disk space used by the remote users is accounted for. The time includes that involved for library maintenance, file editing, line usage and command language processing, as well as that related to the running of programs.

THE COMMAND AND EDIT LANGUAGE PROCESSOR.

Running with the MCP in an area of resident memory is the Command and Edit (CANDE) language processor. CANDE serves as the interface between the remote user and the MCP. The user communicates with the system using established Command and Edit verbs by which he can attach himself to the system, create and manipulate files, and compile and run desired programs which he either wrote or updated from his remote unit.

PROGRAMMING LANGUAGES.

Users of the B 5500 TSS have full language compatability of the following:

a. B 5500 Time Sharing ALGOL provides a fail-safe version of B 5500 Extended ALGOL, including string manipulation capabilities.

b. FORTRAN IV implemented to the specifications of FORTRAN IV, version 13.

c. B 5500 COBOL implemented to the specifications of D.O.D COBOL 61, with extensions.

d. BASIC[1] since it is widely accepted in the industry as a language which is easily understood and can be used by people without computer experience. In addition to serving as a learning tool for the computer layman, it can be readily applied to many business and scientific problems.

---

1. Developed at Dartmouth College

Programs written in these languages are automatically segmented on the logical boundaries of the language; e.g., ALGOL programs being segmented according to blocks, COBOL programs according to paragraphs, and FORTRAN programs according to subroutines. These segments allow a program of virtually unlimited size to be executed within a relatively small amount of memory.

SECTION 3
USING THE TIME SHARING SYSTEM

GENERAL.

Input to the system consists of messages typed in at the remote terminal. Each message must be ended with a left arrow (←). After a left arrow has been typed, the system responds with a carriage return and a line feed to automatically position the teletype for the next line of input. Any carriage returns and line feeds typed by the user are ignored by the system. The system will not accept messages which are more than 72 characters in length (i.e., one Teletype line).

USER IDENTIFICATION.

Before the user can utilize the system capabilities, he must first present his user code and password to the system. The password gets typed into an area that the Teletype has blacked out to ensure that only authorized persons gain access to the system. CANDE will check the user's identification against its file of authorized users. If the code entered is not in the file, the user will be so informed by CANDE and he will be prevented from using the system until a correct log-in procedure is entered. If the code is in the list of authorized users, CANDE will log the station in, record its current log-in time, and consider the station to be a bona fide user. (See LOG-IN for a further description of this procedure.) After a successful log-in, the user code is used to identify the accounting information for the session as well as any files created.

In addition, the user has the option of also entering a charge code which is added to the accounting information but is not used as verification by the system. This can be used to further assist internal billing and accounting.

COMMANDS.

Each command in the Command and Edit language consists of a verb and, in most cases, a list of parameters. The parameters in the list are separated from the verb and from each other by commas or blanks. In the examples contained herein, commas will be used, but they are not required.

When CANDE has finished processing a command, it will type a message either confirming successful completion of the command or indicating an error. Errors terminate the processing of a command and the entire line must be retyped correctly. After certain verbs, a number sign (#) is used to indicate completion. The user should not type anything between the time he types the command and the time the response from CANDE is typed.

It should be noted that more than one command may be included on a single line if the commands are separated by semicolons. However, if an error is contained in a command, any further commands on that line are ignored.

ERROR MESSAGES.

Error messages consist of the word ERR followed by a colon and usually a one word error description, such as ERR:NOFILE. The user can request further information about the error by typing a question mark followed by a left arrow. The error messages that may result from improper use of a given verb are discussed in the section containing the description of that verb.

LOG-IN.

Before he is allowed to use the system, the user must make contact with the computer and submit proper identification. To establish a connection, the following steps should be taken:

a. To turn the Teletype on and obtain a dial tone, push the ORIG (originate) button.

b. Dial the computing center. When the computer accepts the call, the ringing changes to a high-pitched tone and the system types:

B5500 TIME SHARING

If the phone keeps ringing, the system is temporarily unavailable. If a busy signal is heard, the system is loaded to capacity. In either case, re-dialing is necessary.

After a connection is made, the system initiates the log-in sequence which consists of the following steps:

a. The system types

ENTER USER CODE, PLEASE-

and the user complies by entering the user code assigned to him.

b. The system then responds with

AND YOUR PASSWORD

and blacks out seven spaces on the next line into which the user must enter his password.

c. If the system recognizes the user code and password, it proceeds to step d. If it does not recognize them, it types

BADCODE

ENTER USER CODE, PLEASE-

and the log-in procedure begins again. If the user does not enter his user code within 30 seconds or if he fails to log-in correctly on his second try, he is disconnected.

d. If it has been specified that a charge code be requested, the system types

ENTER CHARGE CODE, PLEASE-

and the user enters the charge code for the session.

e. Whether or not a charge code is requested, the system signals its readiness to accept a command by typing

YOU HAVE LINE logical line number, user name

mm/dd/yy, time

The logical line number is the systems recognition of the line dedicated to the user.

Example:

B5500 TIME SHARING

ENTER USER CODE, PLEASE-usercode←

AND YOUR PASSWORD

XXXXXXXX (user enters password in blacked out area)

ENTER CHARGE CODE, PLEASE-charge←

YOU HAVE LINE 07, user name

07/23/68 6:20 PM

FILES.

A file is any collection of information which the user regards as a unit. It is the primary means by which he establishes continuity

between one session at the Teletype and the next, the "session" being that sequence of activities mutual to the system and the user between one log-in and the next log-out.

FILE NAMES.

Each file in the system has a unique identification consisting of a file name and a user code. When a file is created, the user supplies the file name and CANDE adds his user code to form a complete file identifier. File names may be from one to six characters long. The first character must be a letter and any remaining characters must be a combination of letters and digits. If more than six characters are used, the right-most characters are truncated.

Examples:

X
FILE1
JONES
D23XY8

FILE TYPES.

Associated with every remote file is a file type. The file types are: ALGOL, COBOL, BASIC, FORTRAN, and SEQ (sequenced). The first four types specify that the file contains a program written in that language. A sequenced file may contain any kind of data including source code of given programs or parts of programs. Data files also may contain any kind of data but, unlike the other five types, they do not contain sequence numbers.

SEQUENCE NUMBERS.

All files which are edited by CANDE must contain sequence numbers. A sequence number is defined as a positive integer containing a maximum of eight digits which must appear as the first item on a teletype line. It is used to identify the line for editing and to specify the lines position in the file. Even if the lines are

typed out of order, CANDE will arrange them in ascending numerical sequence. Except for programs written in BASIC, the sequence numbers are not considered to be a part of the data in a file.

WORK FILE.

The Command and Edit language allows the user to create or modify the contents of only one file at a time. That file is referred to as the work file. Changes and additions made to the work file do not become permanent unless the user issues a SAVE command. In this way, a file can be created, used, and then discarded if it is no longer needed. Similarly, an existing file can be modified and used without affecting the original copy of the file. If a SAVE command for the modified version is not issued, the original version remains on disk and can be used again. The original file is only changed permanently when a SAVE is typed.

Changes are kept in the work file until either another file is specified as the work file or the users session is terminated. The work file, therefore, can be used like any other file; that is, it may be further changed, listed, compiled, or run. For example, a source file could be loaded and modified, and then compiled and tested until it was completely debugged before it was finally saved, thus ensuring that the running version of the program was available at all times.

USER LIBRARY.

The set of files that a user creates and saves are referred to as his user library. Each file in the library must have a different name. If an attempt is made to create a new file with a name that is currently being used, the system will notify the user via an error message. The necessary change or remove options must be exercised.

ACCESSING OTHER USER'S FILES.

A user may not access files in another user's library unless the owner of the library has authorized him to do so. To do this, the

owner uses the GUARD command to create a GUARD file which is a specially formatted file containing user codes and program names belonging to other users. He then associates the GUARD file with specific files in his library through the GUARD and LOCK commands. These files, called private files, can then be accessed by the users and programs listed on the GUARD file. Those files which do not have associated GUARD files are called sole user files and can be accessed only by the user who created them. In addition, files may be made available to all users with the UNLOCK command.

Once he has been authorized to do so, a user accesses a file in another user's library by using the file name followed by either a slash (/), or the word LIBRARY, and then by the user code of the owner of the file. It should be noted that a user can only read another user's file; he cannot write on them or perform library maintenance.

The file security system is further described in appendix C.

ENTERING DATA.

Data may be entered from the remote unit either by keyboard or paper tape. Data entered by hand (keyboard) must include sequence numbers. Once a work file is specified, all lines beginning with a sequence number are added to the work file.

Errors in the work file may be corrected by using the FIX command or by retyping the line in error using the same sequence number. The new line will overwrite the old. To delete a line which is already part of the work file, the sequence number and then a left arrow must be typed.

Example:

```
MAKE SAMPLE BASIC ←
FILE:SAMPLE TYPE:BASIC -- CREATED
"THIS IS AN EXAMPLE OF A BASIC PROGRAM ←
100 X=5 ←
```

```
150 Y=7←
200 B=X+Y←
250 PRIMT X,Y,A←
175 A=X+Y←
300 END←
"LIST WILL CAUSE THE SYSTEM TO SORT THE WORK FILE←
"AND TYPE IT OUT.←
LIST←

FILE:SAMPLE - TYPE:BASIC  --05/25/68  6:46 PM.
100 X=5
150 Y=7
175 A=X+Y
200 B=X+Y
250 PRIMT X,Y,A
300 END

END LIST 1.0 SEC.
"SINCE LINE 200 IS NOT USED, IT CAN BE ELIMINATED←
200←
"CORRECT MISSPELLING AT LINE 250←
250 PRINT X,Y,A←
"NOW WHAT DO WE HAVE←
LIST←

FILE:SAMPLE - TYPE:BASIC  --05/25/68  6:48 PM.
100 X=5
150 Y=7
175 A=X+Y
250 PRINT X,Y,A
300 END

END LIST 1.0 SEC.
"LOOKS OK NOW←
```

PAPER TAPE INPUT.

Most teletypes have an attached paper tape reader which can be used to enter input as an alternative to the keyboard. Thus, programs and data may be created and stored off-line and then entered at relatively high speed when the terminal is connected to the B 5500 Time Sharing System. The paper tape must be in the proper format for the system.

After the paper tape punch is turned on, 10 or 20 rub-outs should be typed to provide a leader for mounting the tape in the reader. Since reading the tape causes the teletype to print, each line punched into the tape must end with a left arrow (←), a carriage return, a line-feed, and one or two rub-outs. (The rub-outs, which are ignored by the system, make it easier to visually locate a particular record on the tape.) It should be noted that an X-OFF character is not needed. After all lines have been punched, 10 or 20 more rub-outs should be typed to provide a space in which the reader may be turned off.

Since the records from paper tape are read directly into the work file, system commands (with the exception of the FIX command) may not be punched in the tape. For further information, see the description of the TAPE command.

SPECIAL CHARACTERS.

The characters on the remote unit which are not letters or digits are called special characters. The special characters which can be used with the B 5500 Time Sharing System are listed in appendix A. Use of the other special characters will result in errors.

The left arrow (←), the backspace ('), and the delete (!) are always recognized by the system to function as described. All other valid special characters may be used in data, comments, and certain commands but are illegal elsewhere except for the following uses:

CHARACTER	
'	Backspaces one character.
!	Deletes the current line.

←	Marks the end of an input message.
:	Separates commands within an input message.
Blank, or ,	Separate parameters within a command.
#	Indicates successful completion of a command.
?	Requests additional information from CANDE.
"	Introduces a comment.
+	Specifies increment for the SEQ verb.
-	Used in lists of sequence numbers.

BACKSPACING.

The apostrophe (') is used as a backspace to delete the most recently typed character. Repeated use of the backspace will delete a corresponding number of characters, but only to the start of the current line. Thus,

```
MAKE XYZ'2
```

would appear to CANDE as

```
MAKE XY2
```

and

```
MAKE''''LIST
```

would appear to CANDE as

```
LIST
```

DELETING A LINE.

An entire line can be deleted by typing an exclamation point (!). After a delete is typed, the system types DEL and positions the print head at the start of the next line so that the user can continue his input. The delete cannot be used repeatedly, thus only the current line can be deleted.

Example:

```
X=SQRT(X**2+Z!DEL
100 X=SQRT(Y**2+Z**2) ←
```

COMMENTS.

Lines which begin with quotation marks (") are treated as comments, the contents of which have no effect on the use or operation of the file being created. When the user has finished a comment, he can continue to type in data, commands, or further comments.

Examples:

```
"THIS IS A COMMENT? ←
"IT BEGINS WITH A " AND IS COMPLETELY IGNORED ←
```

RUNNING A PROGRAM.

Two following steps are involved in running a program from a source file.

a. Compilation. The source file is translated into machine instructions which form an object file.

b. Execution. The machine instructions in the object file are executed.

The Command and Edit language allows the user to specify either compilation or execution with the COMPILE and EXECUTE verbs, or using the RUN command, to request that CANDE do whatever is necessary to execute the program.

All three of these verbs allow the user to specify a particular file or, by default, to specify the work file. If the command specifies a file which was loaded to form the work file, the version of the file on disk is used, not the work file. For instance, if the following had been done:

```
LOAD TEST ←
FILE:TEST - TYPE:BASIC -- LOADING
END LOAD 1.1 SEC.
100 X=Y**2
```

then the command

```
RUN TEST
```

would cause the old version of TEST without the change at line 100 to be run, whereas

```
RUN
```

would cause the work file version of TEST to be run, which includes the change.

If an error occurs during a compilation or execution, CANDE prints an error message and identifies the sequence number of the line which caused the error.

Anything not beginning with a question mark which is entered after a RUN or EXECUTE command and before the completion of an execution is assumed to be data for the program and is put in a special file. When the program requests data from the teletype, it will use any data that has already been entered before requesting further input from the teletype.

During a run, ?STATUS may be used to ask for information about the program. It is not considered to be data (see the STATUS command). Anything else following a question mark is ignored.

If, for any reason, the user wishes to discontinue his job after it has started running, he should type a WRU (control E) or a BREAK, depending on whether or not the teletype is typing under system control. If it is typing, the BREAK is used. If it is not, WRU should be used. In either case, the program will be terminated and the system will type

```
-USER DS-ED, LINE NO sequence number
```

and then a number sign (#) when it is ready for further input.

Compilations can be terminated by a BREAK or WRU like any other program. BREAK can also be used to suppress the output from a LIST command. However, the programs run by CANDE to process such commands as LIST, PRINT, APPEND, MERGE, COPY, DELETE, and RESEQ are not terminated by a BREAK or WRU.

In the rare event that something does go wrong with the system, a HALT/LOAD is performed. This causes all processing on the system to be halted and the system to be reinitialized. Data can be lost from the work file and any user programs which were running must be restarted.

After a HALT/LOAD, the system types

```
P
 L
  O
   P
     RESTARTING . . . PLEASE WAIT
```

and then asks the user for his user code and password to log him back on the system. If the user had data in the work file, CANDE will usually be able to recover it. However, the last few records should be listed to ensure that nothing was lost.

SECTION 4

TIME SHARING COMMANDS

GENERAL.

The commands available with the Burroughs B 5500 Time Sharing System are described in this section. These commands give the computer system direction as to specific actions which must be executed in order to perform the required tasks. The commands are presented in alphabetical order to make access to them more convenient to the user.

TYPOGRAPHIC CONVENTIONS.

The following conventions have been used throughout this manual to describe the syntax of the Command and Edit verbs:

a. Upper case letters and special characters in a given format denote the literal occurrence of the characters represented. For example,

    MAKE f

    denotes the literal occurrence of the verb MAKE followed by a file name.

b. Lower case letters denote variable elements of a command. For instance, f in the above example indicates that the verb MAKE must be followed by the name of a file. Thus the above construct could be written as

    MAKE UTIL2

    where UTIL2 is an acceptable file name.

c. Ellypsis marks, ..., denote the occurrence of the immediately preceding syntactical item one or more times. For example,

    SCHEDULE $f_1,f_2,...f_n$

    indicates that the verb SCHEDULE must be followed by a

file name which in turn may or may not be followed by one or more file names to form a longer list.

It also should be noted that, in the examples to follow, characters typed by the system are underlined. Those characters not underlined are understood to have been typed by the user.

NOTE

The complete Command and Edit syntax is also provided in COBOL format in appendix D.

APPEND.

Records from a given file may be copied onto the end of the existing work file with the APPEND (or ADD) command. The format is as follows:

1. APPEND f

2. APPEND f,q

where f is a file name and q is a list of one or more elements, in any combination, of the following two forms:

a. s

b. s_1-s_2

where s, s_1, and s_2 are sequence numbers and $s_2 > s_1$.

In addition, the last element of the list q may be one of the following two forms:

a. END

b. s TO END

where s is a sequence number.

If no sequence numbers are specified, the entire file is appended. Otherwise, only the specified portions are added. The appended lines are given sequence numbers equal to their old sequence numbers, plus the highest sequence number originally in the work file. If this results in a sequence number of more than eight digits, the error message

ERR:TOOBIG

is typed, and the remaining lines are not appended.

Example:

```
LIST TA←
FILE:TA - TYPE:SEQ  --6/28/68  7:20 PM
10 TEN.
20 TWENTY.

END LIST  .9 SEC.
MAKE T1←
FILE:T1 - TYPE:SEQ  -- CREATED
100 ONE.←
200 TWO.←
300 THREE.←
APPEND TA; LIST←
#

FILE:T1 - TYPE:SEQ  --6/15/68  7:22 PM
100 ONE.
200 TWO.
300 THREE.
310 TEN.
320 TWENTY.

END LIST 1.0 SEC.
```

BYE.

The BYE verb terminates a user's session. The format is:

After the user types BYE, the system responds with

ON FOR n_1 MIN, n_2 SEC.
C&E USE n_3 SEC.
OFF AT time
GOODBYE usercode

and then performs a disconnect. The user may also log-out by pressing the CLR (clear) button or by typing an End-of-Transmission character (control D). In either of these cases, the system response cannot be given.

Example:

```
BYE←
 ON FOR  27 MIN, 19.3 SEC.
 C&E USE 17.9 SEC.
 OFF AT   6:48 PM.
 GOODBYE HARRY
```

CHANGE.

The remote user can change his password or various file parameters using the CHANGE command. The format of the CHANGE command is:

1. CHANGE PASSWORD
2. CHANGE f_1 TO f_2
3. CHANGE f TYPE TO t
4. CHANGE TO f
5. CHANGE TYPE TO t
where f, f_1, and f_2 are file names, t is a file type, and s is a security file name.

The CHANGE command can be used to change the user's password. The system will ask the user for his old password, his choice for a new password, and a verification (re-entry) of his new password. If an error is made in entering the old password or verifying the new one, the system types

ERR:BADCODE

and the change procedure must be started again. Successful completion of the change procedure is signaled by a number sign (#).

Example:

```
CHANGE PASSWORD ←
ENTER OLD PASSWORD, PLEASE
XXXXXXX                          (old password entered)
NEXT ENTER NEW PASSWORD
XXXXXXX                          (new password entered)
PLEASE REPEAT YOUR NEW PASSWORD, FOR VERIFICATION
XXXXXXX                          (new password entered)
#
```

The CHANGE command can also be used to change the name or the type of

either a library of file or the work file.

The use of TO or TYPE TO causes the name or the type of the specified file to be changed to the file name or file type given in the command. If an attempt is made to change a file name to a name that is already in use, the error message is:

ERR:file name

If a file name does not appear immediately after the verb, the change applies to the work file. A successful change is indicated by a number sign (#).

Example:

```
LIST FILES←
FILE:STUFF    TYPE:SEQ        12/17/67
FILE:UTIL     TYPE:FORTRAN    03/21/68
#CHANGE STUFF TYPE TO COBOL←
#CHANGE STUFF TO UTIL←
ERR:UTIL
CHANGE UTIL TO UTIL2←
#CHANGE STUFF TO UTIL1←
#LIST FILES←
FILE:UTIL1    TYPE:COBOL      12/17/67
FILE:UTIL2    TYPE:FORTRAN    03/21/68
#
```

CHARGE.

The CHARGE command changes the charge code during a user's session. Its format is:

CHARGE c
where c is the charge code of up to seven alphanumeric characters.

Proper information is entered in the log to reflect a change of charge code.

Example:

```
CHARGE 1B5500←
#
```

COMPILE.

To compile a source program and create an object file, the user issues a COMPILE command. The format is:

COMPILE f, c
where f is the file name and c is the compiler name.

If a file name is not given, the work file is compiled; otherwise, the specified file is compiled and the resultant object file is saved. If the work file is compiled, the object file is kept with the work file so that both source and object versions can be saved with a SAVE command. It should be noted that a SAVE command is ignored if data has not been entered into the work file since it was created or last saved. Thus,

```
LOAD X;COMPILE;SAVE←
```

is not necessary to save the object version of X. The command which should be used is

```
COMPILE X←
```

which will automatically cause the object file to be saved.

The compiler name may be ALGOL, COBOL, FORTRAN or BASIC, or an abbreviation consisting of a colon followed by the first letter of the compiler name. It is required when files of type SEQ or type DATA are compiled but can be used for any type file and will override the original file type.

If the work file is not declared when needed, the error message is:

```
ERR:WRKFILE
```

If a source version of the file named is not in the library, the error message is:

```
ERR:NOFILE
```

When the file is found, CANDE types

COMPILING

and, when the compilation is finished,

END COMPILE n SEC.

is typed indicating the amount of processor time used during the compilation.

Example:

```
COMPILE TEST02←
COMPILING
200 Y:UNDEFINED
END COMPILE 1 SEC.
#LIST TEST02 200←
FILE:TEST02 - TYPE:ALGOL  --05/22/68
200 X:=X/Y;
END LIST 1.0 SEC.
LOAD TEST02←
FILE:TEST02 LOADED
200 X:=X*X;←
COMPILE←
COMPILING
END COMPILE 1 SEC.
```

COPY.

The COPY command is used to copy a file onto the work file or onto a peripheral unit. The format of the COPY command is:

1. COPY f TO h
2. COPY
3. COPY q
4. COPY r
5. COPY q r
6. COPY f
7. COPY f q
8. COPY f r
9. COPY f q r

where f is

a. a file name
b. file name / user code
c. file name LIBRARY user code

h is the hardware type, and

a. 2 - designates printer
b. 3 - designates punch
c. 4 - designates tape

q is a list of one or more elements, in any combination, of the following forms:

a. END
b. s TO END

where s is a sequence number.

r is

a. RESEQ
b. RESEQ s_1 - s_2
c. RESEQ base
d. RESEQ s_1 - s_2 + increment
e. RESEQ base + increment
f. RESEQ s_1 TO s_2
g. RESEQ s_1 TO s_2 + increment

where s_2 may be the word END. A list of the above would require 64 cases.

When a COPY TO hardware command is entered by a user, a copy of the file is placed on disk and identified by placing either a 2, 3, or 4 (for files to be printed, punched, or placed on tape) in front of the file names. Only alphanumeric files can be printed or punched. Files copied to tape are placed on multi-file tapes, identified as user code/file name, and are blocked (56, 10). These are processed by the program HARD/CANDE, which is initiated by the installation operator. The output will be sent to the user from the computing center. After the system schedules the output, CANDE responds with a number sign. If there are files to copy, the program asks:

FILE NAMES?

and the operator enters either a YES or NO via an AX message, depending on whether or not he wishes a list of names of all files waiting to be copied. The program will then ask:

WHATS NEXT?

and the acceptable responses are:

a. ALL.
b. STOP.
c. Hardware type.
d. User code.
e. File name/ user code.
f. Hardware type user code.
g. Hardware type file name/user code.

The corresponding actions taken by the program are:

a. COPY all files.

b. Terminate.

c. COPY all files designated for the specified unit.

d. COPY all files for this user.

e. COPY the designated file (may be to more than one unit).

f. COPY all files belonging to the user which are designated for the specific unit.

g. COPY the file.

The acceptable entries for hardware types are PRINTER, PUNCH, or TAPE.

When files are copied to tape, they are blocked (56, 10). All the files for a given user are placed on one tape using the user code as the multi-file identification and the file name as the file identification. Thus, a file identified on disk as file name/user code will appear on tape as user code/file name. When a file is punched, it is preceded by a blank card, a card identifying the file, and another blank card.

After a file is copied, the copy on disk is removed. The program will continue to ask "whats next", until told to stop or until all files are copied.

After the copy has been performed, CANDE will type:

END COPY.

Within the rules of file security, the COPY command may be used to access files in another user's library by including his user code after the file name. If the user issuing the COPY command has not been authorized by the other user to access his files, the COPY will not be performed and CANDE will type:

ERR:user code

If the file is not in the library, CANDE types:

ERR:file name

If there is no work file, the error message is:

ERR:WRKFILE

Examples:

```
LIST EXAMP←
FILE:EXAMP - TYPE:SEQ --06/06/68
100 THIS
200 HERE
300 IS
400 AN EXAMPLE
END LIST 1.0 SEC.
MAKE GOOD; COPY EXAMP 100,300-END←
FILE:GOOD - TYPE:SEQ -- CREATED
WAIT.
END COPY .n SEC.
LIST←

FILE:GOOD - TYPE:SEQ --07/24/68
100 THIS
300 IS
400 AN EXAMPLE
END LIST .9 SEC.
```

CREATE.

The CREATE command creates a new file and establishes it as the work file. The formats of the CREATE command are:

1. CREATE f
2. CREATE f t
3. CREATE f SIZE n
4. CREATE f t SIZE n
where f is a file name, t is a file type, and n is the number of lines to be contained in the file.

DELETE.

The DELETE verb is used to delete all or part of the current contents of a work file. The format is:

1. DELETE ALL
2. DELETE q
where q is a list of one or more elements, in any combination, of the following two forms: a. s b. s_1-s_2 where s, s_1, and s_2 are sequence numbers and $s_2 > s_1$. In addition, the last element of the list q may be one of the following two forms: a. END b. s TO END where s is a sequence number.

If there are no parameters following the DELETE verb, the ALL option is assumed. The parameter ALL causes the contents of the work file to be removed, but does not affect the file name associated with the work file.

If sequence number parameters are used, an entry of the form s causes the line with that sequence number to be deleted. An entry of the form s_1-s_2 causes the deletion of all lines from the first through the second sequence number, inclusively. The use of the word END is equivalent to using the highest sequence number in the file. Thus, if the last entry is END, the last line in the file will be deleted and, if the last entry has the form s TO END, all lines with a sequence number greater than or equal to the sequence

number will be deleted. The sequence numbers must be arranged in ascending numerical order. If one is out of order, it is flagged as an error by the message:

ERR:SEQNUM

A maximum of nine sequence numbers are allowed in a given list. A request to delete non-existant records according to sequence number reference is ignored.

After the specifield lines have been deleted, CANDE will type END DELETE.

Example:

```
MAKE TRASH←
FILE:TRASH - TYPE:SEQ -- CREATED
100 EXAMPLE OF←
200 DELETE VERB←
DELETE ALL←
WAIT.
END DELETE .n SEC.
100 THIS←
200 WILL←
300 BE←
400 USED←
500 AS AN←
600 EXAMPLE←
700 OF THE←
800 DELETE←
DELETE 200,400-550,700 TO END←
WAIT.
END DELETE .n SEC.
LIST←
FILE:TRASH - TYPE:SEQ  --06/16/68
100 THIS
300 BE
600 EXAMPLE
END LIST 1.0 SEC.
```

EXECUTE.

The work file or an object file on disk can be executed by using the EXECUTE (or the DO) command. Its formats are:

1. EXECUTE
2. EXECUTE f
3. EXECUTE f/u
4. EXECUTE f LIBRARY u
where f is the object file name and u is the user's code. The above may be repeated with DO instead of EXECUTE.

The program is run from the object file associated with the file specified in the command. If authorized to do so, a user may execute the object versions of files in another user's library by following the file name with either a slash (/) or the word LIBRARY and then the user code. If there is no object file, the error message is:

ERR:NOFILE

If the object file is found, the message typed by CANDE is:

RUNNING

When the program is finished, the message is:

END file name n SEC.

This indicates the processor time used to execute the job.

Example:

```
EXECUTE TEST←
RUNNING
END TEXT 4.6 SEC.
```

FILES.

The FILES command is used to obtain the names of the files in the user's library. Its formats are:

1. FILES
2. LIST FILES

The name of all the files in the user's library are listed. The end of the list is indicated by a number sign (#).

Examples:

```
FILES←
LINKS   SANTA
#

LIST FILES
LINKS   SANTA
#
```

FIX.

The FIX command is used to delete or replace a portion of a line of input. It has the following formats:

1. FIX s d 1 d n
2. * s d 1 d n
where s is the sequence number, d is a delimiter, 1 is the old string, and n is either a new string or is blank.

A FIX command causes CANDE to replace the characters specified by the old string with the characters in the new string. It does this by searching the line specified by the sequence number from left to right until it finds a string of characters in the line which is identical to the old string specified in the FIX command. It then discards those characters and, if a new string was included in the command, it inserts the characters of the new string in their place. Therefore, any given string of characters in a line may be deleted or replaced by another string.

The delimiter is used to mark the beginning and end of the old string. It may be any valid non-blank character that does not appear in the old string. The first non-blank character after the sequence number is taken as the delimiter. Everything, including blanks, between the first two appearances of the delimiter is taken as the old string. Everything following its second appearance is taken as the new string. Neither of the strings may exceed 63 characters in length. If the FIX command results in a record of more than 72 characters, the record is truncated to 72 characters.

CANDE does not apply the change when it is entered. Instead, it stores the contents as a record with other data entries in the work file. Then, when a command which affects the work file is issued, such as LIST, RUN, SAVE, etc., all the changes that have been stored

are made, and any errors in the FIX commands are noted. Thus, error messages are typed following the first command which uses the work file. Except for causing error messages to be typed, FIX commands in error are ignored, and processing continues as if they had not been entered.

Since CANDE initially treats a FIX command as if it were a record in the work file, more data or another command may be entered after the FIX command has been terminated by a left arrow (←). Furthermore, the FIX command may not be combined with other commands through the use of the semicolon, as described under COMMANDS.

If the specified string cannot be found, the following message is typed:

FIX IGNORED -- SEQUENCE NUMBER sequence number

Examples:

```
MAKE X←
FILE:X TYPE:SEQ -- CREATED
100 THIS IS A SAMPLE←
200 TO SHOW HOW FIX←
300 WORKS←
"CHANGE -SAMPLE- TO -EXAMPLE-←
FIX 100 . S.N EX←
"REPLACE -HOW-←
FIX 200 #HOW#THE WAY IN WHICH←
LIST←

FILE:X - TYPE:SEQ  --04/29/68
100 THIS IS AN EXAMPLE
200 TO STHE WAY IN WHICH HOW·FIX
300 WORKS

END LIST 1.1 SEC.

"OOPS- THE LETTERS -HOW- ARE A PART OF SHOW AND THE←
"FIX CHANGED THEM BECAUSE THEY CAME FIRST IN THE LINE←
```

```
*200 $THE WAY IN WHICH$←
*300 .S.ING←
LIST←

FILE:X - TYPE:SEQ  --04/29/68
100 THIS IS AN EXAMPLE
200 TO SHOW FIX
300 WORKING

END LIST .9 SEC.
```

GUARD.

The GUARD command permits the user to build or modify a guard file to allow other users or user's programs to read or read/write to a file. Its format is:

GUARD

CANDE starts by typing:

NEW OR OLD GUARD FILE?

and the user responds with the word NEW if he wishes to create a new guard file, or OLD if he wishes to update an existing guard file. CANDE then types

GUARD FILE NAME?

and the user types the name of the old file or the name he wants to use for the new file. If a current guard file is being updated, CANDE types:

ADD, DELETE, LIST, SAVE, OR QUIT?

The user then responds with his applicable choice causing CANDE to again type:

ADD, DELETE, LIST, SAVE, OR QUIT?

With the exception of QUIT, the user can type any of these words in any order until he accomplishes everything he wants to do with the file. Actions are taken for each option as follows:

a. ADD is used to add user codes and/or program names to the file. After ADD is typed, CANDE will respond:

READ ONLY NAMES?

The user may then enter a list of user codes and/or program names which will be added to the file. These user codes/ program names will be allowed to read but not change those files with which this guard file is associated. Program names must be entered in the format

file name/user code (of the owner of the file)

The items in the list must be separated by commas or blanks. If the user does not wish to add any read-only names to the guard file, he should type a group mark (←).

Next, CANDE will ask for

READ/WRITE NAMES?

Any user codes and/or program names entered in this case will be able to access and change those files with which this guard file is associated.

b. DELETE causes the program to type:

NAMES TO BE DELETED?

The user then enters those user codes and/or program names he wishes to remove from the guard file.

c. LIST produces a listing of all the user codes in the file. Read only names are preceded by an R in parentheses and read/write names are preceded by a W.

d. SAVE must be typed to save the guard file. The file can be saved more than once, in which case only the version last saved remains on disk.

e. QUIT causes the program to terminate. Any additions or deletions made since the last SAVE will not be entered into the guard file.

When a new file is being created, the program first asks for READ only names and then for read/write names just as it does for an ADD. It then asks

ADD, DELETE, LIST, SAVE, OR QUIT?

and the user may use any of the options described above.

Example:

```
GUARD←
 RUNNING

NEW OR OLD GUARD FILE?OLD←
LOCK FILE NAME?YALE←
ADD, DELETE, LIST, SAVE, OR QUIT?
LIST←
(R) ALICE
(R) WHITE
(W) RABBIT
(W) HATTER
ADD, DELETE, LIST, SAVE, OR QUIT?
ADD←
READ ONLY NAMES?RED, QUEEN←
READ/WRITE NAMES?←
ADD, DELETE, LIST, SAVE, OR QUIT?
DELETE←
NAMES TO BE DELETED?
HATTER←
ADD, DELETE, LIST, SAVE, OR QUIT?
SAVE←
LOCK FILE SAVED
ADD, DELETE, LIST, SAVE, OR QUIT?
QUIT←
THANK YOU.

 END GUARD 1.1 SEC.
#
```

HELLO.

The HELLO verb is used to initiate a log-in sequence. The format is:

The HELLO verb causes the session of the current user to be terminated and a new session initiated without physically performing a terminal disconnect.

LIST.

The LIST command is used to list the contents of a file. Its format is:

1. LIST
2. LIST f
3. LIST q
4. LIST f q
5. LIST f/u
6. LIST f LIBRARY u
7. LIST f/u q
8. LIST f LIBRARY u q

where f is a file name, u is a user code, and q is a list of one or more elements in any combination, of the following two forms:

a. s
b. s_1-s_2

where s, s_1, and s_2 are sequence numbers and $s_2 > s_1$.

In addition, the last element of the list q may be one of the following two forms:

a. END
b. s TO END

where s is a sequence number.

Without a file name, the LIST command lists the work file. With a file name, the specified file is listed. If a list of sequence numbers is not included, the entire file is listed. Otherwise, the lines with the specified sequence numbers are listed. An entry of the form s causes that line to be listed. An entry of the form s_1-s_2 will cause a listing of all lines with sequence numbers in the range from

the first sequence number through the second sequence number. The word END is equivalent to the highest sequence number in the file. A maximum of nine sequence numbers are allowed in the list. If more than nine are entered, the message typed is:

ERR:TOOMANY.

Any extra entries will be ignored.

The sequence numbers in the list must be in ascending numerical sequence. If a sequence number is out of order, CANDE types:

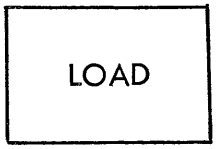
ERR:SEQNUM.

Requests to list non-existent records according to sequence number reference are ignored.

Examples:

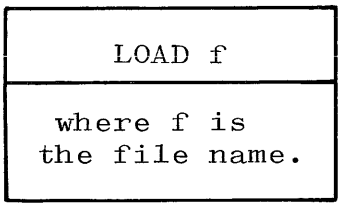
```
LIST←
ERR:WRKFILE
LOAD X;LIST←
FILE:X - TYPE:ALGOL -- LOADING
END LOAD 1.1 SEC.

FILE:X - TYPE:ALGOL  --09/24/67
100 BEGIN REAL X,Y;
200 X:=Y:=5;
300 X:=X/Y;
400 END.

END LIST 1.1 SEC.
LIST 50-250,END←

FILE:X - TYPE:ALGOL  --09/24/67
100 BEGIN REAL X,Y;
200 X:=Y:=5;
400 END.
```

```
END LIST 1.0 SEC.
LIST X 100, 250 TO END
FILE:X - TYPE:ALGOL  --09/24/67
100 BEGIN REAL X,Y;
300 X:=X/Y;
400 END.

END LIST .9 SEC.
```



LOAD.

The LOAD command loads an existing file as the work file. The format is:



If the load is successful, CANDE types:

```
FILE:file name TYPE:file type -- LOADING
END LOAD 1.0 SEC.
```

If, however, the file is not in the user's library, the message typed is:

```
ERR:file name
```

Examples:

1.
```
LOAD←
ERR:NOFILE
LOAD BIGFIL←
FILE:BIGFIL - TYPE:SEQ -- LOADING
END LOAD 1.0 SEC.
```

2.
```
LOAD JUNK←
ERR:JUNK
?←
JUNK   IS NOT IN YOUR LIBRARY
LOAD JUNKY←
FILE:JUNKY - TYPE:ALGOL -- LOADING
END LOAD 1.0 SEC.
```

LOCK.

The LOCK command restores a file to its original locked status, or attaches a guard file to a file. Its formats are:

1. LOCK f
2. LOCK f WITH g
3. LOCK SOURCE f
4. LOCK SOURCE f WITH g
5. LOCK OBJECT f
6. LOCK OBJECT f WITH g
where f is a file name and g is a guard file.

Only the owner may access his files with locked status (all files start off as locked). Only the programs and the users listed in the guard file may access the file (see GUARD to create a guard file). The owner may access without being in the guard file.

Examples:

1. LOCK SOURCE EXAMP←
   LOCK EXAMP←

2. LOCK SOURCE EXAMP WITH USEFLE←

MAKE

MAKE.

The MAKE command creates a new file and establishes it as the work file. The formats of the MAKE command are:

1. MAKE f
2. MAKE f t
3. MAKE f SIZE n
4. MAKE f t SIZE n
where f is a file name, t is a file type, and n is the number of lines to be contained in the file.

The file type may be ALGOL, COBOL, FORTRAN, BASIC, or SEQ which can be abbreviated as :t, where t is the first letter of the type. For instance, :A would be equivalent to ALGOL. If no type is specified, then sequenced (SEQ) is assumed.

A MAKE command which includes a SIZE specification may be immediately followed by the SAVE verb to reserve an area of the specified size. The output files for BASIC programs must be reserved in this way.

After the file has been declared with the MAKE verb, CANDE responds with:

FILE:file name - TYPE:type -- CREATED

If a file with the specified name already exists, CANDE types:

FILE:file name - TYPE:file type -- DUPLICATE NAME

Examples:

1. MAKE GROOVY BASIC←
   FILE:GROOVY - TYPE:BASIC -- CREATED

2. MAKE←
   ERR:NOFILE

```
?←
FILE NAME REQUIRED
MAKE FAB←
FILE:FAB - TYPE:SEQ -- DUPLICATE NAME
?←
FAB     ALREADY EXISTS IN YOUR LIBRARY
MAKE FAB2 :S
FILE:FAB2 - TYPE:SEQ -- CREATED
```

MERGE.

An existing file can be merged into the work file by using the MERGE command. It has the following formats:

1. MERGE f
2. MERGE f q
3. MERGE f r
4. MERGE f q r

where f and r are defined as in the COPY command, yielding 48 cases. q is a list of one or more elements, in any combination, of the following two forms:

a. s
b. s_1-s_2

where s, s_1, and s_2 are sequence numbers and $s_2 > s_1$.

In addition, the last element of the list q may be one of the following two forms:

a. END
b. s TO END

where s is a sequence number.

The specified file, or the indicated portions of it, will be merged into the work file according to sequence numbers. In case of duplicate sequence numbers, the record in the work file maintains priority. A successful merge is indicated by a number sign.

The numbers in the sequence list must be in ascending numerical order. END is equivalent to the last sequence number in the list. A maximum of nine entries are allowed in the list. If the work file has not been declared, the message typed is:

ERR:WRKFILE

Example:

```
LIST USA←

FILE:USA - TYPE:SEQ --07/04/68
100 ONE.
200 TWO.
300 THREE.

END LIST .9 SEC.
MAKE XYZ←
FILE:XYZ - TYPE:SEQ -- CREATED
200 TWO.X←
400 FOUR.X←
600 SIX.X←
MERGE USA←
#LIST←

FILE:XYZ - TYPE:SEQ  --11/07/68

100 ONE.
200 TWO.X
300 THREE.
400 FOUR.X
600 SIX.X

END LIST 1.0 SEC.
```

PRINT.

The PRINT command is used to print a line(s) or a file. It is different from the LIST command in that it suppresses the heading and causes CANDE to type only a number sign (#) when finished. The formats are:

1. PRINT
2. PRINT f
3. PRINT q
4. PRINT f q
5. PRINT f/u
6. PRINT f LIBRARY u
7. PRINT f/u q
8. PRINT f LIBRARY u q

where f is a file name, u is a user code, and q is a list of one or more elements in any combination, of the following two forms:

a. s
b. s_1-s_2

where s, s_1, and s_2 are sequence numbers and $s_2 > s_1$.

In addition, the last element of the list q may be one of the following two forms:

a. END
b. s TO END

where s is a sequence number.

The PRINT command causes the work file, the file specified, or a sequence range to be printed. Only the sequence numbers specified will be printed. This can be from one to nine sequence numbers.

Examples:

```
PRINT
ERR:WRKFILE
LOAD X;PRINT←
FILE:X - TYPE:ALGOL -- LOADING
END LOAD 1.1 SEC.
100 BEGIN REAL X,Y;
200 X:=Y:=5
300 X:=X/Y;
400 END.
#

PRINT 50-250,END←
100 BEGIN REAL X,Y;
200 X:=Y;=5;
400 END.
#

PRINT X 100, 250 TO END
100 BEGIN REAL X,Y;
300 X:=X/Y;
400 END.
#
```

PUNCH.

The PUNCH command may be used to punch the work file or a file on disk to paper tape. The format is:

PUNCH f
where f is the file name.

After entering a PUNCH command, the user must turn on the paper tape punch. The system will then send 10 rubouts, the name of the file, 40 rubouts, the contents of the file, and 40 more rubouts. Each line of data is ended with a carriage return, a line feed, a group mark, and a rubout. The tape can therefore be read back to the system using the TAPE command by initially positioning it in the first set of 40 rubouts.

If the specified file is not present, the error message is:

ERR:NOFILE

REMOVE.

The REMOVE command is used to remove files from the user's library. Its format is:

REMOVE s
where s is a list of one or more elements, in any combination, of the following three forms: a. f . . . b. SOURCE f . . . c. OBJECT f . . . where f is the file name of the file being removed.

If the optional words SOURCE and OBJECT are not used, both versions of the files named in the list are removed. The SOURCE and OBJECT options are included to indicate that the files following them in the list should have only the source or object versions removed. These options apply to all files following them in the list until another option is invoked or until the end of the list. For example,

REMOVE FILE1, FILE2, SOURCE FILE3, FILE4, OBJECT FILE5

would result in the removal of both versions of FILE1 and FILE2, the source versions of FILE3 and FILE4, and the object version of FILE5. Note that all files for which both the source and object versions are to be removed must appear in the beginning of the list.

If the source version is removed, a file can only be run or executed. A maximum of nine entries, i.e., file names and uses of the SOURCE and OBJECT options, are allowed in the list. If this maximum is exceeded, CANDE will respond with

ERR:TOOMANY

and will ignore the extra entries. After the REMOVE command has

been executed, a number sign is typed to indicate that CANDE is ready for the next command.

Reference to a non-existent file will be noted by the message:

ERR:file name

Examples:

1. REMOVE FILE1,FILE2,FILE3←
   #
   "BOTH VERSIONS OF ALL THREE FILES HAVE BEEN REMOVED.←

2. REMOVE FILE1,SOURCE FILE2,OBJECT FILE3←
   #
   "BOTH VERSIONS OF FILE1 HAVE BEEN REMOVED←
   "ONLY THE SOURCE VERSION OF FILE2 HAS BEEN REMOVED←
   "THE OBJECT VERSION, IF PRESENT, REMAINS ON DISK←
   "FILE3 -SAME AS FILE2 WITH SOURCE AND OBJECT REVERSED←

3. REMOVE OBJECT FILE1,FILE2,FILE3←
   #
   "OBJECT VERSIONS OF ALL THREE FILES HAVE BEEN REMOVED←
   "SOURCE VERSIONS, IF PRESENT, REMAIN ON DISK←

RENAME.

The RENAME command changes the name of the work file. The format is:

RENAME f
where f is the new file name.

RENAME may be used to give the work file a new name so that a subsequent SAVE will not destroy an existing file. A number sign (#) is used to indicate that the renaming has been performed. If there is no existing work file, CANDE responds with

```
ERR: WRKFILE
```

Example:

```
LIST FILES←
TESTX
LOAD TESTX←
FILE:TESTX - TYPE:BASIC  -- LOADING
END LOAD 1.0 SEC.
RENAME TSTING←
#LIST FILES←
TSTING
#300 X=Y←
RENAME TESTER←
#SAVE←
FILE:TESTER - TYPE:BASIC  -- SAVED
LIST FILES←
TSTING TESTER
#
```

RESEQ

RESEQ.

The RESEQ verb is used to change sequence numbers in the work file. It has the following formats:

1. RESEQ f
2. RESEQ f b
3. RESEQ f s_1-s_2
4. RESEQ f b+i
5. RESEQ f s_1+s_2+i
where f is the file name, b is the base number, s_1 and s_2 are sequence numbers, and i is the increment. If a file name is not specified, RESEQ will operate on the work file.

If a base number is used, the entire file will be resequenced using the base number as the first sequence number and increasing each successive number by the increment. If the base number and/or the increment are not given, they are assumed to be 100.

If a pair of sequence numbers is used, the lines between the two sequence numbers are resequenced using the given increment.

After the file has been resequenced, the number sign is typed. If resequencing results in a sequence number of more than eight digits, resequencing is abandoned and the message

ERR:TOOBIG

is typed. This leaves the last lines of the file with incorrect sequence numbers. A correct RESEQ must be given before proceeding.

Example:

```
MAKE TESTO2 :F←
FILE:TESTO2 - TYPE:FORTRAN -- CREATED
```

```
10 REAL I,J←
20 I=I/J←
30 END←
RESEQ +30←
#LIST←

FILE:TEST02 - TYPE:FORTRAN  --10/19/67
100 REAL I,J
130 I=I/J
160 END
END LIST .9 SEC.
RESEQ 100-130 +20←
#140 J=I**2←
LIST←

FILE:TEST02 - TYPE:FORTRAN  --10/19/67
100 REAL I,J
120 I=I/J
140 J=I**2
160 END
END LIST 1.0 SEC.
```

RUN

RUN.

The RUN command causes CANDE to take whatever actions are necessary to run the specified program. It has the following formats:

1. RUN
2. RUN f c
3. RUN f/u
4. RUN f LIBRARY u
where c is the compiler name or is blank, f is the file name, and u is the user's code.

CANDE will run the specified file by executing the object version if it is available or, if there is no object version, by compiling and executing the source version. If a file name is not included in the RUN command, the work file will be run.

If RUN f requires f to be compiled, the resulting object file is executed but not saved. However, if the work file is compiled with RUN, the object file is kept in the work file so that, if the work file has been changed since it was created or last saved, both versions can be saved.

The compiler name can be ALGOL, COBOL, FORTRAN, BASIC or an abbreviation consisting of a colon followed by the first letter of the compiler's name. It is required for files of type SEQ, but in any case, it will override the original file type and can be used for any file type.

If it has been arranged through the file security system, object files belonging to another user can be executed by including his user code in the RUN command. Note that one user can only execute the object version of another users file. He cannot compile the source version. Therefore, a RUN command specifying a file in another users library is equivalent to an EXECUTE command specifying that file.

The messages typed by CANDE are the same as those typed for the COMPILE and EXECUTE commands. Thus, if compiling is necessary, the messages are:

```
COMPILING
END COMPILE n SEC.
```

The messages for execution are:

```
RUNNING
END file name n SEC.
```

Examples:

```
LIST EXAMP←

FILE:EXAMP - TYPE:BASIC  --11/17/67
100 LET X=Y=4←
150 PRINT "X="X, "Y="Y←
200 END

END LIST .8 SEC.
LOAD EXAMP←
FILE:EXAMP - TYPE:BASIC  -- LOADING
END LOAD .2 SEC.
FIX 150."Y=".X*←
LIST←

FILE:EXAMP - TYPE:BASIC  --11/25/67
100 LET X=Y=4
150 PRINT "X="X,X*Y
200 END

END LIST 1.0 SEC.
RUN EXAMP←                    (original version)
RUNNING
X=4          Y=4
END EXAMP .4 SEC.
RUN←                          (new version)
```

RUN
continued

```
COMPILING
END COMPILE 1 SEC.
RUNNING
X=4                16
END EXAMP .4 SEC.
```

SAVE.

The SAVE command causes a copy of the current work file to be saved on disk for future use. Its format is:

SAVE

The SAVE command does not clear the work file, but merely establishes a permanent disk file which reflects the work file at the time the SAVE was entered. Any previous copies of the file are removed. If the work file is saved more than once, only the version last saved remains on disk.

If the work file has not been changed since it was initially created or last saved, the SAVE command is ignored. If the save is performed and there is an object version of the work file which agrees with the source version, both versions are saved. Otherwise, the object file is not saved and the disk space allocated for it is returned to the system. Whenever a save is done, the old versions of both the source and object files are removed.

Each file is saved as a locked file unless the user has previously specified a different form of file security through the use of the GUARD, LOCK, and UNLOCK commands.

When the file is saved, CANDE types:

```
FILE:file name - TYPE:file type  --SAVED
```

If there is no work file, CANDE types:

```
ERR:WRKFILE.
```

Example:

```
LOAD TESTX←
FILE:TESTX - TYPE:ALGOL  -- LOADING
END LOAD 1.0 SEC.
010 COMMENT **ALGOL PROGRAM**;←
SAVE←
FILE:TESTX - TYPE:ALGOL  -- SAVED
```

SCHEDULE.

Jobs may be added to the schedule of jobs to be run in the background mode by using the SCHEDULE command. The format is:

SCHEDULE $f_1, f_2, \ldots f_n$
where the f's are file names.

The object versions of the files named are scheduled to be run in background mode. This means that the job cannot send output to or receive input from the Teletype. A number sign is typed after the jobs have been scheduled. When the job has finished running, the message

file name FINISHED

is sent to the Teletype if the user is still connected to the system.

If the file is not present or does not have an object version, the message typed is:

ERR:NOFILE

Example:

```
SCHEDULE SD01B,SD01A←
#
```

SEQ.

The SEQ verb is used to request CANDE to type the sequence numbers for the user as he is inputting his file. It has the following formats:

1. SEQ
2. SEQ i_1
3. SEQ $+i_2$
4. SEQ $i_1 + i_2$
where i_1 is a line number and i_2 is an increment whose value is no greater than 500,000.

The first integer gives the starting line number and the second, which is always preceded by a plus sign, gives the increment between successive sequence numbers. If the increment is missing, 100 is assumed. If the starting sequence number is missing, the highest sequence number currently in the work file plus the increment is used. When automatic sequencing is used, the user must wait for the sequence number to be typed before entering his data. Automatic sequencing is terminated by typing a group mark (←) immediately after the sequence number. CANDE will type a number sign (#) to indicate that it is ready for further input.

Example:

```
MAKE XBASIC←
FILE:XBASIC - TYPE:SEQ  -- CREATED
"OOPS-- NEED A BLANK OR COMMA BETWEEN PARAMETERS←
MAKE X BASIC←
FILE:X - TYPE:BASIC  -- CREATED
SEQ←
100 DIM X(20)←
200 Y=3*X(2)←
```

SEQ
continued

```
300 Z=Y**2←
400←
#150 INPUT X←
#SEQ←
400 I=I+1←
500 END←
600←
#
```

STATUS.

The STATUS command is used to obtain the present status of the user or his programs. The format is:

1. STATUS
2. STATUS f
3. ? STATUS
where f is the file name.

A STATUS command with a file name is used to request information about a job which has been scheduled. It will return either the time the job has been in the schedule or the processor time it has used running. If the file has not been scheduled, the error message is:

ERR:file name

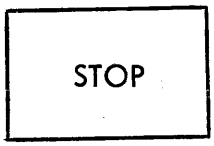
If a file name is not included and the user is not running or compiling a program, the STATUS command returns the date, the time of day, the time at which the user logged-in, the charge code being used (if any), the elapsed time since he logged in, the processor time he has used, and the status of any jobs he has scheduled.

If the user has a job running, the STATUS command may be used to find out how long it has been running. In this case, the STATUS verb must be preceded by a question mark to distinguish it from data for the program.

Example:

```
STATUS ←
        12/29/67          10:42
        ON AT             10:30
        ON LINE           12 MIN
        PROCESSOR TIME    31 SEC
```

```
SCHEDULED ITEMS
    JOB1    RUNNING    10 SECS
#

STATUS JOB1←
JOB1    SCHEDULED   6 MIN
#

RUN JOB2←
RUNNING
?STATUS←
JOB2    RUNNING   28 SECS
```



STOP.

A scheduled job may be terminated by the STOP command. The format is:

If the program is running, it is aborted and CANDE responds

file name...USER DS-ED. USED n SECS

where n is the number of seconds used by the program before it was discontinued.

If the program is not yet running, it is removed from the schedule and CANDE responds:

file name REMOVED FROM SCHEDULE.

If a schedule command has not been issued for that file, the response is:

ERR:file name

Example:

STOP SD01B←
SD01B REMOVED FROM SCHEDULE

TAPE.

The TAPE command is used to specify that a paper tape file is to be read. Its formats are:

1. TAPE
2. TAPE SEQ
3. TAPE SEQ i_1
4. TAPE SEQ $+i_2$
5. TAPE SEQ $i_1 + i_2$
where i_1 is a line number and i_2 is an increment whose value is no greater than 500,000.

The system will type an OK message and then send an X-ON character which initiates the tape reader if it is set to AUTO-START. If the reader is not set for AUTO-START, the user must manually start the reader after the OK message. After the tape has been read, the user must turn off the reader and type ?END← to terminate the tape mode. The system responds with a number sign to indicate that it is no longer in tape mode. All system output to a user in tape mode is suppressed between the time the X-ON and the # are sent.

When the SEQ option is not used, the data is treated the same way as ordinary input. Each line must have a sequence number, but the lines may be out of order. Corrections, in the form of FIX commands or retyped lines, may be included on the tape. When SEQ is used, the first line is given a sequence number equal to the base, and the sequence number for each succeeding line is increased by the increment. In the latter case, the lines must be in order and they cannot contain sequence numbers or FIX commands.

A work file must be specified by the use of LOAD or MAKE verb before the TAPE command may be typed. If this sequence is not followed, the system will type:

ERR:WRKFILE

Example:

```
MAKE WORK COBOL←
FILE:WORK TYPE:COBOL -- CREATED
TAPE←
OK
?END←
#
```

TO.

The TO command is used to send a message to other dialed-in users or to the computer operator at the central site. The formats are:

1. TO SPO m
2. TO ALL m
3. TO u m
4. TO i_1 m
where m is the message, u is a user code, and i_1 is a logical line number. SS can also be used in place of TO.

Depending upon whether SPO, ALL, a user code, or a logical line number is used in the command, the message will be typed at the operator's console, at the terminals of all the users logged on, at the terminals of any users logged on with the specified user code, or at the terminal connected to the specified line. SPO is a mnemonic for supervisory printer, the name given to the operator's console. In all cases, the message is typed in the following format:

FROM sender's user code (sender's logical line number) message

Example:

```
TO SPO I NEED A SCRATCH TAPE
```

If there are no users connected with the given user code, or if the specified line is not logged in, CANDE will respond with either

```
user code  NOT ON
```

or

```
logical line number  NOT ON
```

TO
continued

If the receiving user is in operation, such as running a program, CANDE will type to the sending user:

```
user code  BUSY
```

In either case, CANDE will discard the message.

TYPE.

The TYPE command is used to change the file type associated with the work file.

TYPE t
where t is the file type.

The file type options are ALGOL, COBOL, FORTRAN, BASIC, and SEQ which may be abbreviated with the first letter of the type preceded by a colon.

If a work file has not been assigned to the user, the following error message is given:

```
ERR:WRKFILE
```

CANDE responds with a number sign (#) after the file type has been changed.

Example:

```
LOAD IRS←
FILE:IRS - TYPE:SEQ  -- LOADING
END LOAD 1.0 SEC.
TYPE BASIC:LIST←
#

FILE:IRS - TYPE:BASIC  --04/15/68
100 X=4
200 PRINT X**Y
300 END

END LIST .8 SEC.
```

UNLOCK.

The UNLOCK command allows any user to access a file - for read/only (if SOURCE), or execute/only (if OBJECT). The owner may access in any manner. The formats are:

1. UNLOCK f
2. UNLOCK SOURCE f
3. UNLOCK OBJECT f
where f is a file name.

Example:

```
UNLOCK SOURCE EXAMP←
UNLOCK EXAMP←
```

WHATS

WHATS.

The WHATS command returns the name and type of a file and, for a file saved on disk, the number of records it contained. Its formats are:

1. WHATS
2. WHATS f
where f is a file name.

If the file name is not included or is included and is the work file only, the answer returned is:

FILE:file-name TYPE:type --- WRKFILE

If the file name is included and is that of a file saved on disk, the answer returned is:

FILE:file-name TYPE:type --- CONTAINS integer RECORDS

Examples:

```
WHATS XYZ←  (assuming XYZ to have been SAVEd)
FILE:XYZ  TYPE:COBOL --- CONTAINS 56 RECORDS
```

or

```
WHATS←
FILE:XYZ  TYPE:COBOL --- WRKFILE
```

APPENDIX A
CHARACTER SET

The following characters define the character set acceptable to the system. They are arranged by columns in collating sequence with the blank low and the ? high:

Blank	/	A	J	T	3
.	,	B	K	U	4
[	%	C	L	V	5
(	=	D	M	W	6
&	]	E	N	X	7
$	"	F	O	Y	8
*	#	G	P	Z	9
)	@	H	Q	0	?
;	:	I	R	1	
-	+	\	S	2	

Three other characters are acceptable to the system as control characters only. They are the group mark (←), the backspace ('), and the delete (!).

APPENDIX B
RESERVED WORDS

The following is a list of all words used in the Command and Edit language. Words preceded by an asterisk are reserved and may not be used as file names.

A	*COBOL	FIX	*OBJECT	SCHEDULE
ADD	COMPILE	*FROM	PASSWORD	*SEQ
*ALGOL	COPY	FORTRAN	PRINT	SIZE
ALL	CREATE	GUARD	PRINTER	*SOURCE
APPEND	D	HELLO	PUNCH	STATUS
B	DELETE	LIBRARY	REMOVE	STOP
*BASIC	DO	LIST	RENAME	TAPE
BYE	*END	LOAD	RESEQ	*TO
C	EXECUTE	LOCK	RUN	*TYPE
CHANGE	F	MAKE	S	UNLOCK
CHARGE	*FILES	MERGE	SAVE	WHATS

APPENDIX C
THE FILE SECURITY SYSTEM

The B 5500 Time Sharing System uses the file security system developed for the Data Communications MCP. This system recognizes one privileged user who is allowed access to all files in the system. On the Time Sharing System he is defined to be the installation running the computer.

All other users are subject to the constraints of the file security system. For them, there are three levels of file security:

a. A locked file may be accessed only by the person who created it. This type of security is created by the use of the SAVE command. All files start as locked files.

b. A guarded file may be accessed by the users and programs listed in the guard file associated with the file. A guarded file can be created by the use of the GUARD and the LOCK commands.

c. An unlocked file can be read (or executed, if the file has an object version) by anyone, but it can only be changed by the person who created it. Unlocked files are created by the use of the UNLOCK command.

The security status of a file is treated in much the same way as the file name or the file type. In effect, the security status is loaded with the file and, if not changed by the user, remains with it when the file is saved, even if the file name has been changed.

A guard file can specify two levels of access for either programs or users:

a. Read only - the program or user may only read the file (or execute if object code).

b. Read/write - the program or user may read from or write into the file (not allowed on object code).

APPENDIX D

COMMAND AND EDIT SYNTAX

```
{APPEND}
{ADD   }  file-name [{/      } user-code]  [sequence-list]
                    {LIBRARY}

BYE

                  {  PASSWORD    {TO File-name}  }
CHANGE  {[file-name] {TYPE TO type}  }

CHARGE charge-code

COMPILE [file-name] [compiler]

                  {  file-name TO hardware-unit
                  {[file-name [{/      } user-code]]  [sequence-list]
COPY                           {LIBRARY}
                  {     [RESEQ [{range}]  [+ increment]]
                  {            {base }
                  {[ALL]
DELETE  {  sequence-list}

{EXECUTE}  [file-name [{/      } user-code]]
{DO     }             {LIBRARY}

{FIX}  sequence-number delimiter old-string
{*  }                 delimiter [new-string]

GUARD

HELLO

{PRINT}  [file-name [{/      } user-code]]  [sequence-list]
{LIST }             {LIBRARY}
```

```
[LIST]FILES  [ {file-name}  ... ]
             {file-type}
```

```
LOAD file-name
```

```
LOCK  [ {SOURCE} ]  file-name  [WITH guard-file-name]
      [ {OBJECT} ]
```

```
{MAKE  }
{CREATE}  file-name [type] [SIZE integer]
```

```
MERGE file-name  [ {/      }  user code ]  [sequence-list]
                 {LIBRARY}
      [RESEQ  [ {range} ]  [+ increment] ]
                {base  }
```

```
REMOVE  [ {SOURCE} ]  file-name-1,  [file-name-2]...
        [ {OBJECT} ]
    [ [ {SOURCE} ]  file-name-3,  [file-name-4]...  ]  ...
      [ {OBJECT} ]
```

```
RENAME file-name
```

```
RESEQ [file-name]  [ {base  } ]  [+ increment]
                   {range }
```

```
RUN [file-name]  { [compiler]                          }
                 { [ {/      }  user-code ]            }
                 {   {LIBRARY}                         }
```

```
SAVE
```

```
SCHEDULE file-name-1 [, file-name-2]...
```

```
SEQ [base] [+ increment]
```

```
{STATUS  [file-name]}
{?STATUS            }
```

```
STOP
```

```
TAPE  [SEQ [base] [+ increment]]
```

```
{TO}   (SPO        )
{SS}   (ALL        )  message
       (user code  )
       (line number)
```

```
TYPE type
```

```
UNLOCK [{SOURCE}] file-name
       [{OBJECT}]
```

```
WHATS  [file-name]
```

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: _____ FORM: _____

_____ DATE: _____

_____

CHECK TYPE OF SUGGESTION:

☐ ADDITION ☐ DELETION ☐ REVISION ☐ ERROR

cut along dotted line

GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

STAPLE

FOLD DOWN SECOND FOLD DOWN

Postage Will Be Paid by Addressee

No Postage Stamp Necessary If Mailed in the United States

BUSINESS REPLY MAIL
First Class Permit No. 817, Detroit, Mich. 48232

Burroughs Corporation
6071 Second Avenue
Detroit, Michigan 48232

attn: Sales Technical Services
Systems Documentation

FOLD UP FIRST FOLD UP

1038205 6 69 Printed in U. S. America