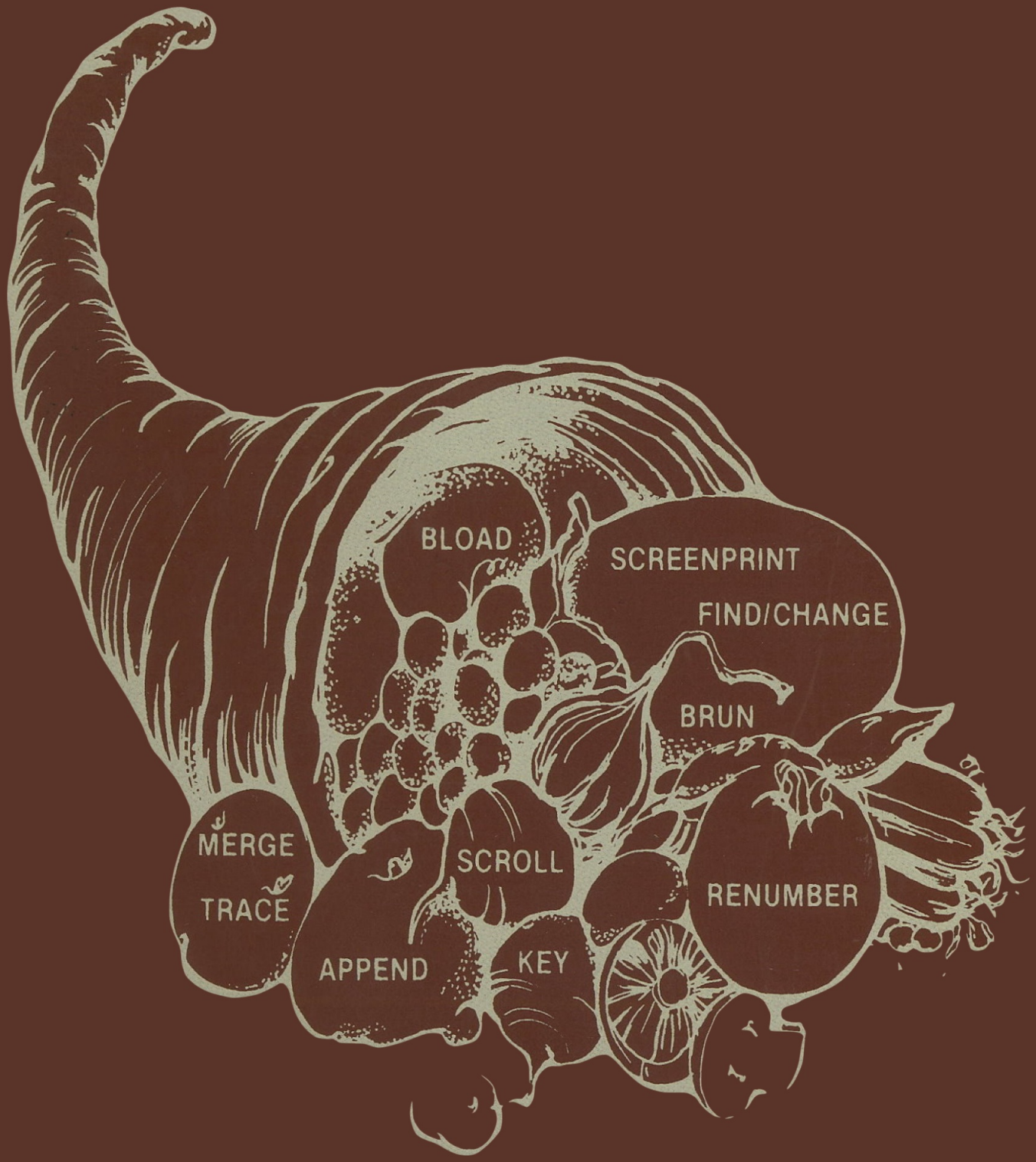
SYSRES™



HANDS ON SOFTWARE, INC.

SYSRES™

THE ULTIMATE RESIDENT PROGRAM MANIPULATION SYSTEM FOR PET™/CBM™ MICROCOMPUTERS

USER'S GUIDE

REVISED JUNE 1987

HANDS ON SOFTWARE, INC .

TABLE OF CONTENTS

TABLES

FIGURES

SYSRES™ by Don Lekei

User Guide recreated from the original by Alan Reed in 2025
Version 1

1. INTRODUCTION

SYSRES™ is an operating system which works in conjunction with the BASIC operating system in your Commodore-64™ to give you, the programmer, advanced program and file manipulation ability. The syntax has been carefully selected to be as similar as possible to many other such systems, while adding a host of additional features.

SYSRES™ is designed to be "addictive to programmers but NOT addictive to programs". This means that no commands are added or changed in the way in which they behave within a program. Software developed under SYSRES™ is fully compatible with a "non-SYSRES™" environment.

By using an extremely powerful syntax structure, SYSRES™ adds over 1100 new functions to BASIC, while only using 33 new command words (plus 11 dos-support commands). Even with all of this power, SYSRES™ is very simple to use, because most commands accept syntax with which you may already be familiar; the more powerful features are optional.

SYSRES hides away under BASIC, so it **USES NO RAM** which would normally be available to BASIC programs (except 256 bytes from $C000 to $CFFF for BASIC interface) (Ref: page 3). One reason SYSRES™ is not in a cartridge is that a cartridge takes 8K away from your BASIC RAM!

Booting SYSRES™

SYSRES™ should be booted with one of the following commands:

To boot from drive 0 (or on a single disk drive):

```
LOAD "0:*",8,3
```

To boot from drive 1 (dual drives only):

```
LOAD "1:*",8,3
```

If these commands are used, SYSRES™ will boot itself automatically, without disturbing any program in memory.

Using an IEEE conversion/expansion cartridge, such as "VLINK-64" from Richvale Telecommunications (which gives you a version of BASIC 4), the [RUN/STOP] key may provide an automatic disk load. If you use the [RUN /STOP] key, or if you accidentally omit the ",3" from the above command, enter "RUN" and SYSRES™ will continue booting. The only ill effect of this procedure will be that the program previously in memory will be lost.

Before using SYSRES™ make a copy of the diskette for use, and store the master away in a safe place. Then insert the COPY into drive 0 and enter:

```
LOAD "0:*",8:
```

and press [SHIFT] [RUN/STOP] instead of return.

Commodore-64 SYSRES™ Bonus Features

Long Line Editing

Because the Commodore-64™ is an off-shoot of the VIC-20™ computer, the input buffer is 88 characters long (the VIC can link up to four consecutive screen lines to form an 88 character editing line). In normal operation, the Commodore-64™ will only link two lines to edit 80 characters. To allow you to edit longer lines, SYSRES™ will link overflow characters onto the end of the current line **in the reverse scroll mode only** (when you move the cursor past the top of the screen).

If the line has more than 88 non-blank characters when you press [RETURN], the message "STRING TOO LONG ERROR" will be printed, and the original line in memory will be unchanged. Lines which cannot be shortened to 88 characters must be edited using the CHANGE commands (Ref: pp. 29-33).

When editing extended lines, you may notice a few anomalies in the way the cursor behaves. This is a harmless side effect of the Commodore-64™ line editor not recognizing the overflow condition, and is corrected once the screen is cleared or the line is scrolled off the screen.

Commodore-64™ BUG Fixed

On rare occasions, when editing the last line on the screen, the Commodore-64™ may crash with the following text at the bottom of the screen:

```
30:L
OAD
?SYNTAX  ERROR
RUN
READY
```

SYSRES™ fixes this problem. It cannot happen while SYSRES™ is in your Commodore-64™. If it does happen to you (this can only happen if SYSRES™ is not in the computer), follow this simple procedure to recover:

1) Take diskettes out of drives.
2) If you do not have BASIC 4 you will need a tape drive connected.
3) Hold the "N" key down and press the "9" key.
4) If you do not have BASIC 4 you must press PLAY on the tape, then press the [RUN/ STOP] key

Compatibility With Other Programs/Systems/Cartridges

Every attempt has been made to make SYSRES™ compatible with other programs, systems, or cartridges which may be in the BASIC environment on your Commodore-64™. Since SYSRES™ loads from disk, it does not prevent you from having another utility in cartridge. Programs which modify BASIC or the "KERNAL" operating system by moving them into RAM may not be used due to limitations in the memory mapping configurations.

If a cartridge is installed in the Commodore-64™, or there is a program at $C000 in memory, SYSRES™ tries to analyze it to determine if it is a BASIC extension. If SYSRES™ finds and recognizes such a program, it will change its listing and tracing routines to accommodate the extension.

SYSRES™ looks for a BASIC extension cartridge and inserts the added commands into its own syntax table for listings and programmed keys. This may cause the codes for some keys to change. Note: SYSRES™ will NOT recognize any cartridge or other BASIC extension If it does not follow the already defined token set of existing Commodore machines. (SYSRES™ looks for the command "CONCAT", which is the first extra Commodore BASIC token). When used with a non-standard cartridge, SYSRES™ will list the extended commands as their BASIC-4 token equivalents (see table 4, pg. 50). ":LIST" will bypass the SYSRES™ listing function.

Added Bonus

Also provided on the SYSRES™ disk is an EXEC file which defines some single key functions to help with editing programs and converting PET™/CBM™ programs to the Commodore-64™. To use this file enter:

```
EXEC K64
```

This will also activate the pre-programmed key functions. If you do not want the other key functions, enter:

```
KEY""""
```

before EXECing this file. You may change this file using the "GET" and "INPUT" commands.

2. SYSTEM OVERVIEW

SYSRES™ contains an advanced automatic repeat-key routine. When a cursor key or space is held down for more than about 1/3 of a second, it will rapidly repeat until it is released. For any other key, if it is held down for about 3/4 of a second, it will repeat at half speed. The RETURN key does not repeat.

If there is a program listing on the screen, and you attempt to move the cursor down (using the CRSR key) past the bottom of the screen, the listing on the screen will move up, and the NEXT LINE OF THE PROGRAM WILL BE DISPLAYED. Attempting to move past the top of the screen will cause the listing to move down and the PREVIOUS LINE OF THE PROGRAM WILL BE DISPLAYED at the top of the screen. If the end of the program has been reached (or there Is no listing on the screen) attempting to move past the top or bottom of the screen will have no effect, and the screen will not scroll.

The scrolling feature also works when in the machine language monitor, allowing you to display lines of the current BASIC program without having to exit from the monitor first.

SCREENPRINT FUNCTION

To produce a hardcopy of the screen at any time, press the [LOGO] and [OFF/RVS] keys at the same time. A "picture" of the screen will be sent to the printer. If the printer is ASCII, characters displayed in reverse on the screen will be underlined.

AUTOMATIC SHUT-DOWN

SYSRES™ reduces its interface to a minimum when a BASIC program Is running. It also disconnects repeat, scroll, screenprint and defined keys. To re-connect these functions after running a program, press [RETURN], or enter any command.

If you want to keep the above functions on in a program (perhaps to screenprint, or the defined keys to provide pre-determined inputs), enter ":RUN" to start the program. "GOTO" and "CONT" will also leave the interrupt functions active. Note that the keys will not repeat unless the cursor is flashing.

When operating under an EXEC file (see "EXEC"), SYSRES™ will remain fully connected at all times. This allows a program to be RUN with an EXEC file providing all of the inputs.

DOS SUPPORT

There are two main groups of commands, as illustrated in tables 1 and 2. Table 1 shows the EXTENDED DOS SUPPORT group; these are all preceded by one of the DOS SUPPORT keys, the variety of codes is to allow users of older, DOS Support utilities to feel comfortable. The usual SYSRES™ key is the back-arrow (top-left of your keyboard).

Table 1:

EXTENDED DOS SUPPORT

@	(type "N" keyboard)	These commands may be used
<	(type "B" keyboard)	Interchangeably, to perform
!	(original keyboard)	the following DOS SUPPORT
>	(for "wedge" users)	functions.

@	Display disk status
@N	Format (HEADER) a new diskette
@!	Force Initialize diskette
@V	Validate diskette (collect)
@D	Duplicate diskette
@C	Copy or concatenate disk file(s)*
@R	Rename file
@S	Scratch file(s)*
@$	List directory**
@U:	Reset disk drive
@L	List disk file**

Note: Some of the disk utility command set may also be used, if an appropriate direct access channel has been opened. Refer to the disk manual for further information.

* Standard command with added options.
** Added disk command.

Table 2: EXTENDED EDITOR COMMANDS

/	[1]	Quick load from disk
^	[1]	Quick load from disk with auto run
APPEND	[1]	Append from disk to end of current program
AUTO	[1]	Auto line number (allows header)
BLOAD	[1]	Load machine language (binary) file
BRUN	[1]	Load and execute machine language program
CHANGE	[1]	Change pattern to another pattern
CLOSE	[2]	Close one or all files
CMD	[2]	Set output to file (does not send "READY.")
DELETE	[1]	Delete a range of lines from program
DUMP	[1]	Dump all scalar variables to screen or file
EXEC	[1]	Execute a file as keyboard commands
FIND	[1]	Find occurrences of a pattern
GET	[1]	Read a sequential file into editor
KEY	[1]	Define a key as a special function
KEYS	[1]	Tum key functions on
KILL	[1]	Disable SYSRES
KILL-	[1]	Disable SYSRES and unreserve memory
LIST	[2]	Improved BASIC LIST command
LOAD	[2]	Defaults to disk drive
MERGE	[1]	Merle from disk into current program
MON	[1]	Break to current machine language monitor
OLD	[1]	Restore program after "NEW".
PUT	[1]	Send program to disk as text file
RENUMBER	[1]	Renumber all or part of program
RUN	[2]	Run current program, ignores screen garbage
SAVE	[2]	Defaults to disk drive, allows replace
SETD	[1]	Set disk device #, allows multiple drives
SETP	[1]	Set printer channel, format mode, paging
TRACE	[1]	Select 1 of 3 trace/step modes and speed
VERIFY	[2]	Compare current program against disk/tape
WHY	[1]	Print position of last error
WHY?	[1]	List line of break or error
*	[1]	Send output to printer
#	[1]	Display current version of SYSRES

[1] Added command
[2] Old command with improvements

The second group is the EXTENDED EDITOR set (table 2). These commands function exactly like BASIC commands except that they must always be the first command on a line, and only one command may be on a line (except "*", which may precede any SYSRES™ or BASIC command).

DIRECTORY ORIENTED COMMANDS

One of the unique features of SYSRES™ is that all commands which operate on disk files may be used from the directory (see table 3). To use a directory command, simply type the command in the margin (type over the file length) and press [RETURN]. This totally eliminates the possibility of typing the wrong filename, and can save a great deal of time.

Table 3:

DIRECTORY ORIENTED COMMANDS

/	Quick load disk file
^	Quick load file with auto run
APPEND	Append file to end of current program
BLOAD	Load machine language (binary) file
BRUN	Load and execute machine language program
EXEC	Execute file as keyboard commands
GET	Read file Into editor
LOAD	Same as "/"
MERGE	Merge file into current program
PUT	Send text to disk, allows replace
SAVE	Save program to disk, allows replace
VERIFY	Compare current program against file
@L	List file to screen
*@L	List file to printer (allows formatting)
@Cd:	Copy file to specified drive
@Sd:	Scratch file from specified drive

3. DOS SUPPORT EXTENSIONS

One of the unique features of Commodore-64™ computer systems is that the disk drive units are totally separate computers, allowing them to function independently of the main computer. The programmer merely sends a command to the disk drive, using Commodore's own DOS (disk operating system) language, and the disk drive interprets the command, and performs the necessary steps to execute the command. In fact, once an internal operation has commenced (loads, saves, and storage/retrieval do not apply, as they require a constant feed of data between the computer and disk drive) the disk drive may be totally disconnected from the computer, without interrupting the operation.

DOS SUPPORT

The primary function of the DOS SUPPORT COMMANDS (see table 1) of SYSRES™ is to allow access to the DOS language of the disk drive. This is done by typing one of the DOS SUPPORT KEYS (the ">" key or whatever key is in the top-left corner of your keyboard may be substituted for the "@" key which is the top-left key on the "N"-series keyboard), followed by a valid command in the DOS language. The manual from the disk drive gives a more detailed description of the DOS language. DOS commands are the commands for which the manual tells you to open a file to device 8, secondary address 15, and print the command to that channeL The DOS SUPPORT COMMANDS do not require that you open a channeL

The DOS instruction may be enclosed in quotes if the files used include cursor control characters (characters shown reverse-field in a directory listing).

SYSRES™ allows you to use a **calculated string expression** in the DOS command message. The only restriction is that the message MUST begin with a string in quotes, even if it is a null string (ie:@“”+A$-+CHR$(1)). For example, If you have a **4040** disk drive and you want to speed up disk access, you could use the command:

```
@“M-W”+CHR$(0)+CHR$(16)+CHR$(3)+CHR$(11)+CHR$(20)+CHR$(255)
```

If a DOS SUPPORT KEY is entered with nothing after it, the disk error channel is read, and the current disk message is displayed. Note that errors are only displayed the first time the error channel is read. Any subsequent attempts will return 100, OK,00,00”.

Errors occurring during file operations (such as @L, PUT, GET, etc.) will display the message:

```
DISK: ##, message, tt, ss
```
(displayed in reverse-video)

Errors resulting from DOS commands WILL NOT BE DISPLAYED UNLESS REQUESTED MANUALLY.

SYSRES™ also adds four commands which function as DOS commands, but which are unique to SYSRES™. These commands are listed on the following pages.

EXAMPLES

```
@N1:LIBRARY DISK,F3
@"n1:sales data disk,SI
```

These are two examples of the way in which a DOS language command may be sent to the disk. These examples were chosen because they are two functions which cannot be done with BASIC 4.0. They both send the "NEW" command, which is the same as the BASIC 4.0 "header" command, but with "header" you cannot use a disk ID beginning with the letter "F" (because IF is a reserved word), or use shifted letters in the ID (the second example is shown as if the computer were in the lower-case mode).

The shifted letters "SI" in the ID produce an interesting effect. When the directory is displayed (using @$ or by loading the directory), the shifted letters will be interpreted as BASIC tokens. In this case, the shifted "S" is the token for "COPY", and the shifted "I" is the token for "RIGHT$", so the ID displays as COPYRIGHT$.

COMMAND

@C

FUNCTION

COPY OR CONCATENATE DISK FILES.

SYNTAX

@C<dest>:"<filename>"
@C<dest>:<directory>
@["]C<dest>=<src>
@["]C<dest>:<filename>=[<src>:]<filename>[,<src>:<filename>] [,<src>:<filename>] [,< src>:<filename>]
Where:

<dest> is the destination drive (to)
<src> is the source drive (from)

DESCRIPTION

The first two options copy the opposite drive to the drive given. For example, the command:

```
@C1:  "TESTFILE" REL
```

will copy the relative file "TESTFILE" to drive 1 from drive 0.

When using the long-form of the @C command, the easiest way to remember which way the copy will proceed is to view the command as an equation. The long-form is useful when you want to change the name of the file, when you are using the concatenation feature, or both, as in the example:

```
@C0:COMBO=1:PART1,0:PART2
```

In this case, a new file, "COMBO", is created on drive 0 which is equal to the file called "PART1" on drive 1 plus the file "PART2" which is on drive 0.

The disk drive will buffer one "@C" command, so the cursor will return immediately after the first "@C" command is entered. If a second "@C" command is entered, the cursor will return as soon as the first scratch is completed.

When copying many files, it may be easier to define a "KEY" function to reduce the amount of typing necessary. The standard way of setting up such a function (shown in lowercase for clarity) is:

```
key"@","@c0:Z
```

EXAMPLES

@C1:"MYPROG

This copies the file "MYPROG" from drive 0 to drive 1.

@C1=0

This copies ALL files from drive 0 to drive 1. It DOES NOT format drive 1, and it does not erase files already on drive 1. This is the recommended command for making backups of diskettes. It allows you to create a backup copy of a diskette which has a different ID than the original diskette, thereby preventing the possible disastrous results caused by consecutively using two diskettes with the same ID.

@C1:MYFILE=YOURFILE

Creates a new file called "MYFILE" which is the same as "YOURFILE". In this case, "YOURFILE" may be on either drive.

COMMAND

@L

FUNCTION

LIST A PROGRAM OR FILE FROM THE DISK.

SYNTAX

@L"<filename>"[P] [RG]
@L"<filename>"S [EQ]
@L"<filename>"R [EL]
@L<directory>

DESCRIPTION

The most common use of this command is to type "@L" in the margin beside the directory listing of the file you want to view, then press [RETURN]. The selected file will be listed to the screen.

If a sequential file (SEQ) is selected, its contents will be displayed on the screen as text, with all cursor characters (and control characters) displayed as the reverse of the base character (eg., a cursor-home character is control-S which will print as a reverse-S on the screen).

A relative file (REL) will be displayed in record-by-record format, with the record number and a line of dashes above each record. The contents of each record are displayed in the same manner as a sequential file. When the end of the file has been reached, the channel will be closed and the screen will display:

```
DISK: 50, RECORD NOT PRESENT,00,00
```

A **program** file (PRG) will be LISTED to the screen as a BASIC program.The program is not loaded into memory, so it does not affect the current program in any way. If the program end is found before the end of the file, the extra code will be listed as program. This enables you to see why the program is too long (eg., the result of saving a program after a program chaining operation).

It is sometimes more useful, as in the case of a machine language program, to view the program as if it were a sequential file. To display a program file as data, force the file to be interpreted as a sequential file by using the form:

```
@L"<filename>"S
```

When using the directory form of the @L command, putting any character other than "$", "R", or "P" between the filename and the letters "PRG" will cause the file to be listed as a sequential file, as in the example:

```
@L "MYPROGRAM" X PRG
```

In this case, the program "MYPROGRAM" will be displayed as if it were a sequential file.

Listings and TRACE windows may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL], or stepped by holding down the SPACE BAR and tapping [LOGO] or ended by pressing [RUN/STOP].

LISTING TO THE PRINTER

The command "*@L" will send the listing to the current printer device. Using the "SETP" command, the printer may be set to any device which does not require secondary addressing or a filename. This may include such devices as modems, ASCII printers, or even the screen (device #3).

If output is sent to the current printer (SETP), the name of the file will be printed at the top of the first page of the listing. On Commodore printers, the name will be printed in enhanced mode, and on ASCII printers it will be underlined.

If the printer device has been selected as <device number> + 64, **program listings will be formatted** with one command per line, spaces between commands, and for-next loops indented by one space for each level of nesting. See the description of the "*" command for further details.

If you have the formatting mode enabled and you want to list a single program **without formatting** you can put a "$" after the filename, in the form:

```
@L"<filename>"$
```

LISTING TO A FILE

There are two methods of sending listings to a file oriented device, such as the disk drive or the tape (only the first method works with tape). The standard method is to open a write file in the usual manner:

```
OPEN1,8,2,"LISTING,S,W"
```

then, use the "CMD" command to set output to that file (must be on a separate line to prevent the word "READY." from being sent) such as:

```
CMD1,"";
```

The output of the standard @L command will now go to the file. When the listing is completed, use the command:

```
CLOSE
```

to suspend output to that file. Note that "CLOSE" and not "CLOSE1" is used, as the SYSRES™ "CLOSE" command also terminates the CMD if necessary ("CLOSE1" will close the file but leave the CMD active, hence the "sticking cursor" bug).

The second method of sending output to a file is usually used when the format option is desired. In this case use the special file number 224 in the example:

```
OPEN224,8,2,"1:FMLIST,S,W"
CMD224,"";
```

When the current CMD file is 224, you may use the "*@L" command to list to that file. This only works once. The file is automatically closed after a "*" command. Note: for formatting to take place, the actual printer device must be set to format mode (eg., SETP,4+64 has previously been done).

A listing may be added to the end of an existing file by opening the file a second time in the form "<filename>,A". This opens the file as an APPEND file, so that new data is added to the end of the original file. This might be used to connect several listings together, so that they may be printed at once, or be processed through some sort of output program.

EXAMPLES

@L"PROGRAM

This is the default form of the command. It will list the program "PROGRAM" to the screen.

@L"*

Re-lists the LAST PROGRAM listed.

@L"M*"

Lists the first program on the disk with a filename beginning with the letter "M".

@L"$0"

Functions exactly the same as "@$0". The "@$" command is actually one of the examples of where the syntax of SYSRES™ has been set-up to make users feel comfortable coming from other systems (in this case, Commodore's UNIVERSAL DOS SUPPORT utility … you can also use >$0).

@L34 "ADDRESS DATA" REL

This will display the contents of the file (taken from the directory) "ADDRESS DATA", in record-by-record format. Notice that the "34" left over from the file length is ignored by the "L" command. The print-out from this command might look something like:

```
1-----------------------------
JOE SMITH
124 ROSE ST.
BAKERSFIELD
CA
20014
2-----------------------------
BILL JONES
404 E.45TH ST.
CALGARY
ALTA
XSM 27R
3-----------------------------
```

Note: A listing of a relative file (if allowed to go to the end) will always end with the message:

```
DISK: 50, RECORD NOT PRESENT,00,00
```

COMMAND

@S

FUNCTION

SCRATCH (DELETE) A FILE.

SYNTAX

@S<drive>:"<filename>"
@S<drive>:<directory>
@["]S<drive>:<filename>[,<drive>:<filename>] [,<drive>:<filename>] [,<drive>:<filename>]

DESCRIPTION

A file or group of files may be deleted by typing "@S<drive>:" in the margin beside the file you wish to remove. If the file exists on both drives, the file will only be deleted from the drive specified, so it is not necessary to have the correct directory displayed on the screen. For example, if the file "NOT NEEDED" is on both drives, and the directory for drive 0 is on the screen, the command:

```
@S1:  "NOT NEEDED" PRG
```

will remove the file only from drive 1.

The disk drive will buffer one "@S" command, so the cursor will return immediately after the first '@S" command is entered. If a second "@S" command is entered, the cursor will return as soon as the first scratch is completed.

The full pattern matching options of the DOS "S" command may be used. See the disk manual for further information.

EXAMPLES

```
@S1:"XFIL*"
```

Will erase all files beginning with "XFIL".

```
@S0:A.1*,0:A.2
```

Will scratch all files on drive 0 beginning with "A.1" or "A.2".

```
@S1:OLD DATA" REL
```

Will scratch the file "OLD DATA" from drive 1, using the directory. Note that all types of files (except block access files, which must be removed with the command "@V" since they have no filename associated with them) may be deleted in this manner.

COMMAND

@$

FUNCTION

LIST THE DISK DIRECTORY.

SYNTAX

@["]$[<drive>] [:] [<pattern>] [,<drive>: <pattern>] [,<drive>:<pattern>] [,<drive>:<pattern>]

DESCRIPTION

The "@$" command displays the disk directory in exactly the same format as if it were loaded into memory. It allows ALL of the directory options provided by the Commodore DOS, including multiple pattern matching.

At any time, the output may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL], or stepped by holding down the SPACE BAR and tapping [LOGO] or ended by pressing [RUN/STOP].

It Is recommended that you use this command instead of the BASIC 4 "DIRECTORY" or "CATALOG" commands as this command protects you from accidentally entering a line from the directory into your program (eg., by pressing [RETURN] while on a directory line). It also ensures that the file size numbers will not interfere with scrolling.

The "@$" command performs an "@L" of the directory, except that the format option is disabled. If formatting were allowed, the directory, which is already formatted by the disk drive, would not look right.

Your disk drive manual has further Information on the "$" command.

LISTING THE DIRECTORY TO THE PRINTER

The command "*@$" will send the directory to the current printer device. Using the "SETP" command, the printer may be set to any device which does not require secondary addressing or a filename. This may include such devices as modems, ASCII printers, or even the screen (device #3).

LISTING THE DIRECTORY TO A FILE

The standard method of sending directory listings to a file oriented device, such as the disk drive or the tape is to open a write file in the usual manner:

```
OPEN1,8,2,"DIRECTORY,S,W"
```

then, use the "CMD" command to set output to that file (must be on a separate line to prevent the word "READY." from being sent) such as:

```
CMD1,"";
```

The output of the standard @$ command will now go to the file. When the directory is completed, use the command:

CLOSE

to suspend output to that file. Note that "CLOSE" and not "CLOSE1" is used, as the SYSRES™ "CLOSE" command also terminates the CMD if necessary ("CLOSE1" will close the file but leave the CMD active, hence the "sticking cursor" bug).

A directory listing may be added to the end of an existing file by opening the file a second time in the form "<filename>,A". This opens the file as an APPEND file, so that new data is added to the end of the original file. This might be used to connect several directory listings together, so that they may be printed at once, or be processed through some sort of output program.

EXAMPLES

@$

List the directories for both drives.

@$0

List the entire directory for drive 0.

@$1:

Display drive 1 name, ID, and free space only.

**@$A*

Display all files (both drives) beginning with the letter "A".

@$1:RE*,0:???

List all files on drive 1 beginning with "RE" and all files on drive 0 with three-letter names.

@$0:$=s

List all sequential files on drive 0.

4. EXTENDED EDITOR COMMANDS

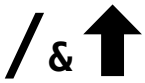
The EXTENDED EDITOR COMMANDS function as if they were BASIC commands, and are essentially an extension of the BASIC language. The "feel" of the SYSRES™ commands is very similar to BASIC in that the command words are English oriented so that by reading the command, one could make a reasonable guess as to its function.

Any SYSRES™ command which produces an output to the screen may be sent to the current printer device by preceding it with the "*" command. Printouts may also be directed to a disk (in some cases the tape may also be used) file for future processing.

A SYSRES™ command must be the only command on a line (except "*"). The reason for this is that the SYSRES commands are not tokenized as BASIC commands since SYSRES commands may not be used within a BASIC program. If SYSRES commands were allowed in BASIC programs, then programs would not be transportable from machine-to-machine (ie, you could not give a copy of your program to a friend, and you certainly could not sell your program if it required that they have SYSRES first). The only exceptions to this rule are the fact that "interrupt" and TRACE functions may be turned on and off from within a program; however, these are not actually commands, they are merely de-bugging tools.

The following pages give a detailed description of how each SYSRES command works.

COMMAND



FUNCTION

QUICK LOAD (/) AND RUN (↑).

SYNTAX

/"<filename>"
/<directory>
/<filename>
/

↑"<filename>" [<start line>]

↑<directory>

↑<filename>

↑

DESCRIPTION

The "/" and "↑"commands provide a quick and easy way to load a program from disk. If used from the directory, all numbers before the quotation mark (") will be ignored. If no name is given, the last program loaded will be restored to memory, or the program just listed with the "@L" command will be loaded. This has two common uses. If you load a program and accidentally change It (by perhaps typing a number) then you can quickly re-load it. Alternately, you may want to check that you are loading the right program by using the "@L" command; then you can easily load the program you were just looking at.

These commands first check for the program on the drive you last used (also set by reading the directory). If it is not found there, the other drive will be checked. You may disable this feature by beginning the filename with "<drive>:" to specify which drive to look on, or by beginning the filename with a colon (no drive number) to only check the drive last used.

The "↑" command functions exactly the same as the "/" command, except that it runs the program as soon as it is loaded.

The "↑" will normally run the program starting from the first line. If you want the program to run starting at another line, you may **specify a starting line number.** For example:

```
↑"MYPROG"100
```

will load the program "MYPROG" from the disk and run the program starting at line 100.

Like the "RUN" command, "↑" will disable SYSRES™ "interrupt" functions (repeat, scroll, screenprint, and defined keys) unless performed from within an "exec" file.

NOTE: These commands will not function with a tape drive. See "LOAD" for tape operation.

EXAMPLES

```
/MAIL LIST
```

Loads the program "MAIL LIST" from whichever drive it is on.

```
↑"0:CREATE"
```

Loads and runs "CREATE" from drive 0

```
/"123COUNT"
```

Loads "123COUNT". Note that if a filename begins with numbers, it must be enclosed in quotes.

```
↑28 ` "REWRITER 3.0" PRG
```

Loads and runs "REWRITER 3.0" from the directory. Note that the file length is ignored, so you don1t have to remove it when using the ↑ or / commands. In order to allow this feature, filenames which begin with numbers (such as when specifying the drive as in "1:NAME") must be enclosed in quotes.

```
↑""+CHR$(34)+"DUMTHINGTODO"
```

If you accidentally get into trouble by saving a program with an un-printable name, you can use calculated strings in the filename as long as the expression begins with text in quotes. This example will load the file ["DUMTHINGTODO]. If the expression does not begin with quotes, it will be interpreted LITERALLY.

COMMAND

APPEND

FUNCTION

APPEND A PROGRAM FROM THE DISK ONTO THE PROGRAM IN MEMORY.

SYNTAX

APPEND "<filename>"
APPEND <directory>

DESCRIPTION

The APPEND function takes a program from the disk drive and adds It to the end of the BASIC program currently in memory. Its most common use is to add a pre-written ending to a program, and is used in place of the MERGE command. APPEND is much faster than MERGE since MERGE moves pieces of the BASIC program around to insert lines into their proper places. Appending a program containing lines lower than the highest line of current program will result in out of sequence line numbers.

EXAMPLES

```
APPEND"END1
```

Adds the program "END1" to the end of the program in memory.

```
APPEND"SET-UP"          PRG
```

Example of using APPEND from the directory.

COMMAND

AUTO

FUNCTION

SET / CLEAR AUTO LINE NUMBERING MODE.

SYNTAX

AUTO [<step>] [,<string expression>] [,<start line>]

DESCRIPTION

The unique auto line number feature of SYSRES™ not only automatically supplies line numbers, but may also be set to supply part or even all of the line as well. After a line is entered, the computer will generate the next line number based on the line just entered. If you change the number, the computer will automatically pick-up your new line and calculate the next number from that point. For example, if AUTO was set by the command "AUT010" and you enter line 100, the computer would feed you line 110. If you now change that to line 150, the computer would pick-up the change and automatically feed line 160.

If a string expression is included, that text will be fed after each line number. The expression is evaluated only once, so any variables will be interpreted as of the time that the AUTO command is given. This header string may be up to 127 characters long, so that cursor control characters may be used to print the entire line, then move back to where data is to be entered. For example, the following command will prepare a set of lines which call a printing subroutine (chr$(34) is the code for a quotation mark):

```
AUTO10,"A$="+CHR$(34)+"<10 spaces>"+CHR$(34)+":GOSUB500<20 cursor-back>",
1000
```

Notice that the starting line (1000) was given in this example. The start line is not necessary because the computer normally picks-up the line number when a line is entered, however; when a complex header is used, it is often easier if you do not have to enter the first line by hand.

If the header is too long to be entered directly in the AUTO command, you may define a string variable to be the header string (remember, only the value of the string at the time that the auto command is given will be used). The previous example could have been done as:

```
H$="A$="+CHR$(34)+"<10 spaces>"+CHR$(34)
H$=H$+":GOSUB500<20 cursor-backs>"
AUTO10,H$,1000
```

The computer would immediately display:

```
1000 A$="_       ":gosub500
```

(The "_" indicates where the cursor is flashing.)

If the [RETURN] key were now pressed, line 1000 would be entered into the BASIC program, and the screen would display:

```
1010 A$="_       ":gosub500
```

You may even include a carriage-return (CHR$(13)) in the header string to cause the automatic generation of a whole group of lines. For example, to create a block of blank data statements, you might use the following command:

```
AUTO10,"DATA"+CHR$(34)+CHR$(13),2000
```

The computer would print:

```
2000 DATA"
2010 DATA"
2020 DATA"
     •
     •
     •
```

The lines would continue to be generated until the [RUN/STOP] key was pressed.

DISABLING AUTO

To turn the AUTO function off, enter (without a line number) "AUTO" with no parameters. AUTO is automatically turned off when a program is run.

AUTO will not feed a new line number unless you enter a line; it is temporarily disabled automatically when you enter a blank line or any immediate mode command. It will re-start when you enter the next program line.

A BETTER WAY

You have probably noticed how often you must use the expressions "+CHR$(34)+" (for quotation marks) and "+CHR$(13)" (for carriage-return) in this command. To make life easier, this command will accept shifted-Q for quotes and shifted-Z for carriage-return within the string expression. In the graphics mode, the PET will display a large dot for shift-Q and a diamond for shifted-Z (the lowercase mode will display capital Q and Z). For example, the expression (as seen in the lowercase mode):

```
"saveQtempQ+ti$Z"
```

is the same as the expression:

```
"save"+chr$(34)+"temp"+chr$(34)+"+ti$"+chr$(13)
```

and would evaluate to be:

```
save"temp"+ti$<cr>
```

The shifted-Q and shifted-Z were selected because they are not used in command abbreviations and are rarely needed.

NOTE: This is a feature of the "AUTO" and "KEY" commands only and does not affect the interpretation of other strings during normal operation.

EXAMPLES

AUTO10,"<cursor back><space>"

This will set auto in steps of 10 and force the first character after the line number to be removed. AUTO normally skips one character instead of blanking it to allow for the times when you don't want to lose the first character.

auto1,"<cursor back><insert><insert>?Q",10

This is an example (shown in lowercase mode for clarity) of a commonly used command for transferring the contents of the screen into PRINT statements. This will automatically insert (chr$(148) is an INSERT) a PRINT and a quotation mark at the beginning of each line, and will only lose the first 3 characters of each line. Variations on this theme may also be used to transfer directory entries into DATA statements (among other things). A challenge would be to do the same thing without losing any of the characters on the screen (HINT: <down> <4back> <5inst> <up> <3right>.

COMMAND

BLOAD

FUNCTION

LOAD A MACHINE LANGUAGE (BINARY) FILE.

SYNTAX

BLOAD ["]<filename>["]
BLOAD <directory>

DESCRIPTION

The BLOAD command performs a normal disk load, except that it does not alter the program pointers. It does, however, automatically perform a clear (CLR), so that programs which load into high memory will not be scrambled if BASIC attempts a garbage collection (garbage collection is an internal function of BASIC which periodically moves string variables around to conserve memory).

BLOAD only works from the disk drive in order to allow the routine to be optimized for disk operation. To load a machine language file from the tape, use the machine language monitor, or load from within a BASIC program.

BLOAD may also be used to simulate the result of chaining BASIC programs. To set-up the program pointers as a normal "LOAD" would, you could use the "OLD" command.

EXAMPLES

```
BLOAD BIG MACRO
```

Loads the machine language program "BIG MACRO" into RAM without changing memory pointers.

```
BLOAD "SCREEN DISPLAY"  PRG
```

Loads "SCREEN DISPLAY" using the directory.

```
BLOAD"1:SETXY"
```

Loads "SETXY" from drive 1.

```
BLOAD
```

Loads the last program viewed (with @L) or loaded.

```
LOAD "THE LONGEST"
BLOAD "THE FIRST"
SAVE "@:THE FIRST"
```

When using the LOAD command from within a program to chain a second program, the first program must be the longest. This example uses the BLOAD command to make "THE FIRST"

the same length as "THE LONGEST". To restore the program "THE FIRST" to its original size, use the "OLD" command. The program BLOADed must be shorter than the program LOADed or part of the program will be lost.

COMMAND

BRUN

FUNCTION

LOAD AND EXECUTE A MACHINE LANGUAGE (BINARY) FILE.

SYNTAX

BRUN "<filename>" [<user parms>]
BRUN <filename>
BRUN <directory>

DESCRIPTION

BRUN performs a "BLOAD", then executes the program via a machine language "JSR" (equivalent of BASIC SYS) to the first address loaded. Like BLOAD, BRUN performs a clear to prevent memory conflicts with string variables.

One unique feature of the SYSRES™ BRUN command is that it allows parameter passing via the BASIC character-get routine. It should be noted that literal expressions only may be passed, since all variables are lost. This parameter passing feature enables the implementation of routines which behave like BASIC commands. For example, a routine to encode a program with an access code might be called by:

```
BRUN"ENCODE","1:PROGRAM",21145
```

In this case, the BRUN command would pass the parameters "1:PROGRAM" (perhaps a filename?) and "21145" (an encryption code?) to our imaginary machine language routine. The routine would be coded in the same manner as if it were to be called via the command:

```
SYS(ENCODE),"1:PROGRAM",21145
```

If parameter passing is not used, the filename does not need to be in quotes except when the name begins with numbers.

BRUN turns off the repeat-key routine, so that the IRQ vector will be pointing to the normal address.

Programs called with "BRUN" should end with an "RTS" instruction. In general, any routine which could normally be called with a "SYS" to the first address loaded, can be called by "BRUN".

There are 3 files on your SYSRES™ disk which you may use with the BRUN command. They are "XMON64C(50135)", "XMON64H(29900)", and "X MON64T(38083) ".These are copies of the public-domain machine-language-monitor, "XMON-64", which operate in conjunction with SYSRES™ or as a stand-alone monitor (see the "MON" command and Appendix-A for details).

List the disk directory (@$) and move the cursor to the left of the screen beside the appropriate version, type "BRUN", and press [RETURN]. In a moment, the computer will print "READY.". By this time, EXTRAMON will already be loaded into your computer, and it will have protected

itself from BASIC and patched in to the "MON" command. From then on, the "MON" command will put you into EXTRAMON (instead of the normal machine-language-monitor in the PET™).

If you attempt to "BRUN" a BASIC program, the computer will "BLOAD" the program, but will not execute it. It will instead return the message:

```
? MEMORY ERROR
```

To prevent the computer from interpreting your machine language program as a BASIC program, it must not start at memory location $0401 (decimal 1025).

EXAMPLES

**BRUN XMON64C*

Loads and executes "XMON64C(50135)".

BRUN "1:RTN1"

Loads and executes "RTN1" from drive 1.

BRUN "FILEX" PRG

Calls "FILEX" using the disk directory.

BRUN

Calls the last program viewed (with @L) or loaded.

COMMAND

CHANGE

FUNCTION

CHANGE OCCURRENCES OF ONE PATTERN TO ANOTHER PATTERN.

SYNTAX

CHANGE [B] [E] [V] [P] [C] [R] ["]<del><pattern><del>["] [:] <del><new pattern><del> [<line-range>]

With the following options:

B	Change only if at the **B**eginning of a line
E	Change only if at the **E**nd of a line
V	Change only if an exact **V**ariable name match
P	Enable **P**attern matching ("#" as "wild-card")
C	Change only if in **C**ommand area (not in quotes, DATA, or REM)
R	Change **R**emainder of line to new pattern
"	Turn quote mode on (disable crunching)

Between strings, correcting characters may be used, such as:

"	Toggle quote mode
:	Turn "DATA" mode off (when "DATA" is in the search pattern)

<del> may be:

Any character not in either pattern except B, E, V, P, C, R, or ". Characters which are used by BASIC (such as *, /, +, =, etc.) are not recommended as they are not always what they appear to be.

<line-range> may be:

-<line A>	Up to (including) line A
<line A>-	From line A to end of program
<line A>-<line B>	From line A to line B
<line A>	Only in line A
<none>	Entire program

DESCRIPTION

The CHANGE command finds all locations within a BASIC program containing a given pattern, and changes them to another pattern, displaying the resulting line(s). This is the most flexible command in SYSRES™, since by combining all of the search / replace options there are more than 700 valid combinations (eg., CHANGE, CHANGEB, CHANGEPC, etc.).

The options B, E, V, P ,C and R may be combined to give the desired selectivity. For example, "CHANGEBPR" will change the line only if the match is found at the beginning of the line, using

the pattern matching mode (a "#" in the search string will match any character in the text), and when a match is found, the remainder of the line will be replaced with the new text.

The **pattern matching** mode allows the use of the "#" character as a "wild-card". For example, the command:

CHANGEP"@PRESS "#"@@PRESS"C"@

will change anything where the "#" is to a "C".

If pattern matching is used, the **the new pattern may contain wild card characters** as well, in which case each wild card will be replaced with the characters (or commands) skipped with the search pattern, for example:

CHANGEP"@A#C#E#G@@###-DEMO@
will change "ABCDEFG"
to
"BDF-DEMO"

If there are more "#" characters in the second pattern than in the first pattern, the excess "#" characters will be left as they are. For example:

CHANGEP@TEST#@@DEMO###@
will change "TEST2"
to
"DEM02##"

Beware of BASIC commands, as they are stored as one byte each, so the command "CHANGECP@####F$@@####FF$@" will not find "LOADF$" or "SAVEF$" because "LOAD" and "SAVE" are stored as single bytes.

The **quotation mark is outside of the delimiters** to allow the quote mode to be turned on when there is a quotation mark within either the search string or the new string.

At any time, the output may be paused with the SPACE BAR, continued with [LOGO), slowed by pressing [CTRL), or stepped by holding down the SPACE BAR and tapping [LOGO) or ended by pressing [RUN/STOP].

If the result of a change would be to eliminate a line totally, the line will be changed to a colon (:). This provides BASIC with a null statement which will be ignored when the program is run. If the line were to be removed entirely, a branch to that line (GOTO, GOSUB, etc.) would result in an "UNDEF'D STATEMENT ERROR". The line may be removed manually if desired (see also "RENUMBER").

The variable ("V") mode will only change exact variable name matches, for example, the command:

CHANGEV@A@@B@

will only change the variable "A" to "B" and will not affect such variables as "AM", "RA", "A$" or "A%". Care must be taken when long variable names are used within a program. Since BASIC uses only the first two letters of a variable, sometimes variables such as "CO" and "COUNT" are used interchangeably within a program. The command:

CHANGEV@CO@@CT@

will ignore the variable "COUNT" because it is not an exact match. This may often be to your advantage as when you accidentally use two long names with the same first two letters.

At first glance, it would seem reasonable to think that when editing a source code, you could isolate a label by using the "V" option. This will work sometimes, but not always. The reason is that a source code is compacted by tokenizing it in the same manner as a BASIC program, in order to conserve memory. Therefore, although a label such as "MK34" would work nicely and would be distinguished from labels such as "MK342", the label "NEXT1" is stored as [NEXT] [1] (two bytes) and would be ignored by the CHANGEV command because it is not a legal variable name.

If you prefer to use descriptive variable names so that the operation of your program is clear, you can now use **variable names which include BASIC reserved words**. The trick is that reserved words are only "reserved" when they have been tokenized. A variable name such as "SCORE" is not allowed since it is stored in memory as [S] [C] [OR] [E]. After the program is written, you can enter:

CHANGE@SCORE@"@SCORE@

which will change the tokenized form of [S] [C] [OR] [E] to "SCORE" in ASCII form. The program will then run properly, and "SCORE" will be interpreted as a legal variable ("SC"). Remember, however, that if you re-enter or edit any line containing such a variable name, the name will tokenized again and the CHANGE command must be used before the program can be run. To CHANGE or FIND a variable created in this way, you must use a form such as:

CHANGEV"@SCORE@@XX@
or
FINDV"@SCORE@

The variables TI, TI$, ST, OS, and DS$ are reserved variables (NOT reserved words), so you cannot create variables which evaluate as any of them ("PAST" is allowed but "STEPS" is not).

The command ("C") option is used to prevent accidentally matching a character in a string which has the same ASCII value as the token for the command. For example, the token for "LOAD" is the same as the character for clear-screen, so to change "LOAD" to "SYS7000," you should use the command:

CHANGEC@LOAD@@SYS7000,@

The CHANGEC command will ignore any match beginning within quotes; however, it allows the match string to extend into quoted text.

The "R" (remainder) option is of particular use in removing REM statements from programs. When a match Is found, the remainder of the line will be deleted, and the new text will be put in its place. The command to remove REM statements is:

CHANGER@REM@@@

If the removal of the REM statement would eliminate a line totally, the line will be changed to a colon as previously described.

CONCATENATING PROGRAM LINES

One of the creative uses of the CHANGE command is to create lines longer than 80 characters. This is often desirable when time and memory space are at a premium. A long program will run much faster if there are fewer lines, and pUtting an entire loop on one line makes it execute faster. Also you save four bytes of code every time you remove one line number.

To concatenate two lines, put a character at the end of the first line (the "!" is used in this example) and change that character to the second line as in this example:

```
10 FOR I=0T099:GET#1,A$,B$,C$,D$,E$,F$,G$:A$=C$+D$+E $+F$+G$:!
   CHANGE@!@@A=VAL(A$+B$):A(A)=I:A$(A)=A$:IFST=0THENNEXT@10
```

Lines up to 245 bytes (when tokenized - count only one byte for each COMMAND) long may be created in this manner, but if the code is intended as a subroutine for your library, care must be taken because the MERGE command can only take lines up to 120 bytes long. For longer lines you must use "LOAD" or "APPEND".

SENDING RESULTS TO THE PRINTER

The command "*CHANGE" will list the result of each change to the current printer device. Using the "SETP" command, the printer may be set to any device which does not require secondary addressing or a filename. This may include such devices as modems, ASCII printers, or even the screen (device #3).

SENDING RESULTS TO A FILE

The standard method of sending CHANGE results to a file oriented device, such as the disk drive or the tape, is to open a write file in the usual manner:

```
OPEN1,8,2,"CHANGE RESULTS,S,W"
```

then, use the "CMD" command to set output to that file (must be on a separate line to prevent the word "READY." from being sent) such as:

```
CMD1,"";
```

The output of the standard CHANGE command will now go to the file. When the CHANGE command Is finished, use the command:

```
CLOSE
```

to suspend output to that file. Note that "CLOSE" and not "CLOSE1" Is used, as the SYSRES™ "CLOSED" command also terminates the CMD if necessary ("CLOSE1" will close the file but leave the CMD active. hence the sticking cursor bug).

The CHANGE results may be added to the end of an existing file by opening the file a second time in the form "<filename>,A". This opens the file as an APPEND file, so that new data is added to the end of the original file.

EXAMPLES

CHANGEC@ @@@

Remove spaces from the program (ignores spaces within quotes or in REM or DATA statements)

CHANGE"$GOTO 34$$GOTO 39$

Changes "GOTO 34" as it would be found within quotes to "GOTO 39" (as in the message, "ENTER 'GOTO 34' AND PRESS [RETURN]").

CHANGE!THENGOTO!!THEN!

Changes any "THENGOTO"s to "THEN".

```
CHANGE@DATA"TEXT"@:@DATA"NEWTEXT"@
CHANGE@DATA"NO END QUOTE@":@DATA"NEW TEXT@
CHANGE@DATA NOT IN QUOTES@:@DATA NEW TEXT@
```

Three methods of handling data statements with mixed quote modes.

CHANGE &SYS50456&&SYSTEM(5)&-59999

Changes "SYS50456" to "SYSTEM(5)" in lines up to and including line 59999.

COMMAND

CLOSE

FUNCTION

CLOSE ONE OR ALL FILE(S).

SYNTAX

CLOSE [<lfn>]

DESCRIPTION

When a logical file number(<lfn>) is given, the CLOSE command works in the normal fashion (closes the specified file). If no file number is given, all disk files will be properly closed, all logical files in the computer will be terminated, and any current CMD will be discontinued. Output will be restored to the screen, and input will be restored to the keyboard.

The special command "CLOSE" (with no file number) is commonly used to clear the disk drive when a program has been stopped leaving a disk file open, or to terminate a CMD without sending anything to the CMD file.

In some cases, when files have been messed-up, the cursor may not return fully to the left column of the screen. This effect is commonly called the "sticking cursor bug". The CLOSE command will fix the CMD /file pointers to restore normal operation (sometimes the first line after the CLOSE may still exhibit the problem).

EXAMPLES

CLOSE 1

Closes logical file number 1 as from OPEN 1,8,2,"0:DATA,A" (standard BASIC "CLOSE" command).

CLOSE

Closes all files.

COMMAND

CMD

FUNCTION

SET A FILE TO BE THE OUTPUT DEVICE.

SYNTAX

CMD <lfn>[,<print expression>]

DESCRIPTION

The CMD command works normally, except that the word "READY." is not sent to the file. The recommended procedure for setting a file as the output device is:

```
CMD <lfn>,"";
```

This format will not send any characters to the specified file; it will only set the file to be the output device.

The CMD command and the PRINT command (or "?") are processed by BASIC, therefore any text is not processed through the ASCII converter.

Note that the CMD command sets output to a LOGICAL FILE not to a DEVICE. The number used in the command must refer to an existing file which has been opened with a previous BASIC "OPEN" command (or equivalent).

A special CMD command may be used with the "*" command. It is:

```
*CMD224, <print expression>;
```

This will print the expression to the current printer device. You do not need to open or close file 224, as the "*" command does it for you. Although the output is not sent through the ASCII converter, the carriage-return/linefeed option will be handled (see "SETP").

File 224 has other special uses. If you open file 224 and use the command:

```
CMD 224,"";
```

the output of the next "*" command or SCREENPRINT will go to the file instead of the printer. Also, the file will automatically be closed.

EXAMPLES

```
CMD 205
```

Sets output to file number 205 as set by a command such as OPEN 205,8,1,"0:PROGFILE". This method sends a carriage-return and a linefeed to the file, so it is not recommended as it may produce undesired results (even with Commodore printers).

CMD 205,"";

This sets the output file 205 as in the previous example, except that it does not send any characters to the file.

CMD 205,"TEXT"CHR$(13);

Sends the word TEXT" to file 205 then leaves the file ready to accept more data.

```
OPEN 224,8,2,"0:TEMP,S,W"
CMD 224,"1";
<perform SCREENPRINT>
<enter the following program>
NEW
10 Z$=CHR$(0):R$=CHR$(13)
20 OPEN1,8,2,"TEMP":GET#1,A$
30 OPEN2,8,3,"0:TEMP2,S,W"
40 PRINT#2,"?"CHR$(34);
50 FORI=1 TO1E9:GET#1,A$:S=ST:PRINT#2,CHR$(ASC(A$+Z$));:IFS=0ANDA$<>R$
   THENNEXT
60 IFS=0THENPRINT#2,"?"CHR$(34);: NEXT
70 CLOSE2:CLOSE1
RUN
GET "TEMP2"
CHANGEEP@#@@@1240
CHANGEEP@#@@#";@
10 ?"<clr>";
9999 goto 9999
```

This sequence will convert the screen into a program segment (note: line 9999 simply waits for the [RUN/STOP] key). The screen may be cleared after the first two commands, and the screen then set-up using the keyboard.

COMMAND

DELETE

FUNCTION

DELETE A RANGE OF LINES FROM THE CURRENT PROGRAM.

SYNTAX

DELETE <line range>

Where <line range> may be:

- <line A>	Up to (including) line A
<line A> -	From line A to end of program
<line A> - <line B>	From line A to line B
<line A>	Only line A

DESCRIPTION

The DELETE command removes an entire group of lines at one time. Although it seems reasonable that the DELETE command with no line range should delete the entire program, a "NEW· command must be used in its place. This is because the DELETE command actually REMOVES the lines from the program, therefore an OLD command could not be used to recover from an accidental "DELETE".

EXAMPLE

To demonstrate the DELETE command we will use it on a small program. User input is BOLDFACE.

```
     10 REM THIS IS AN EXAMPLE
     20 REM OF THE USE OF THE
     30 REM DELETE COMMAND TO
     40 REM REMOVE A SECTION
     50 REM OF A PROGRAM WITH THE
     60 REM DELETE COMMAND
```

DELETE 30-50

```
READY.
LIST
     10 REM THIS IS AN EXAMPLE
     20 REM OF THE USE OF THE
     60 REM DELETE COMMAND
```

DELETE 15-

```
READY.
LIST
 10 REM THIS IS AN EXAMPLE
```

COMMAND

DUMP

FUNCTION

DISPLAY THE VALUES OF ALL SCALAR VARIABLES.

SYNTAX

DUMP

DESCRIPTION

The DUMP command displays the values of all! scalar (non-array) variables In the form:

<variable name>=<value> or <string variable>="<characters>"

DUMP only displays variables which have been used in the program last run, so it may only be used AFTER a program has been run. Variables are always displayed in the order in which they were created.

At any time, the output may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL], or stepped by holding down the SPACE BAR and tapping [LOGO] or ended by pressing [RUN/STOP].

Array variables are not displayed by the DUMP command since arrays could have thousands of elements, and may only have meaning if displayed In a particular order. However, you can easily display the contents of an array with one line of code such as this example for an array (A(10,3)):

```
FORI=0T010:FORJ=-0T03:?"A("I","J")="A(I,J)CHR$(13);:NEXTJ,I
```

Note: You may want to defined a key (see "KEY") to this so that one keystroke will dump your array for you.

SENDING OUTPUT TO THE PRINTER

The command "*DUMP" will list the result of the DUMP command to the current printer device. Using the "SETP" command, the printer may be set to any device which does not require secondary addressing or a filename. This may Include such devices as modems, ASCII printers, or even the screen (device #3).

SENDING OUTPUT TO A FILE

The standard method of sending the printout to a file oriented device, such as the disk drive or the tape, is to open a write file in the usual manner:

```
OPEN1,1,2,"DUMP,S,W"
```

then, use the "CMD" command to set output to that file (must be on a separate line to prevent the word "READY." from being sent) such as:

```
CMD1,"";
```

The output of the standard DUMP command will now 10 to the file. When the DUMP command has finished, use the command:

```
CLOSE
```

to suspend output to that file. Note that "CLOSE" and not "CLOSE1" is used, as the SYSRES™ "CLOSE" command also terminates the CMD If necessary ("CLOSE1" will close the file but leave the CMD active, hence the "sticking cursor" bug).

The DUMP output may be added to the end of an existing file. by opening the file a second time in the form "<filename>,A". This opens the file as an APPEND file, so that new data Is added to the end of the original file.

ADVANCED USES

Since the output format of the DUMP command Is the same as a variable assignment (default form of "LET"), the output can be sent to a disk file, then, using the EXEC command, all variables could be reset to the same values. Note that the character CHR$(13) (carriage-return) and CHR$(34) (quote) will cause problems, so you may have to edit the file (using GET and PUT) before It can be used with an EXEC command. If the program contains array variables, they must be printed to the file separately. A typical sequence for sending a variable dump for a program which also contains an array (A$(9,3)) might be:

```
OPEN 9,8,2,"VARIABLES,S,W"
CMD 9"";
DUMP
FORI=0T09:FORJ=0T03:?"A("I","J")="CHR$(34)A$CHR$(34)CHR$(13);:NEXTJ,I
CLOSE
```

The DUMP command may also be used to change the values of variables by performing a DUMP to the screen, then using cursor editing to change the value on the screen. "CONT" may be used to continue running the program.

COMMAND

EXEC

FUNCTION

EXECUTE A SEQUENTIAL FILE AS KEYBOARD COMMANDS.

SYNTAX

EXEC "<filename>"
EXEC <directory>
EXEC <filename>

DESCRIPTION

EXEC is an extremely powerful command which allows you to prepare a long list of commands which will be automatically entered as if they were typed on the keyboard.

Common uses of the EXEC command include:

Visual program merge
Partial program merge
Convert programs from other computers
Enter programs brought in via modem
Enter program-generated lines
Perform identical editing on several files (global editing)
Set-up "KEY" functions
Perform multiple-line functions for "KEY" commands
Pre-set variables (see DUMP)

"EXEC" files may be created using the editor functions of SYSRES™ and stored to disk with the "PUT" command, or they may be created by a BASIC program.

When using the editor to create an "EXEC" file, notice that a number cannot normally be the first item on a line. To generate a file which will automatically enter program lines, you must put some character between the line number used when editing and the number in the input line, for example:

```
1000 10 PRINT "THIS LINE IS ADDED"
```

would not work because the editor would think you were trying to enter line 100010 and would print "?SYNTAX ERROR". An acceptable solution would be to enter:

```
1000&10 PRINT "THIS LINE IS ADDED"
```

The "&" character would prevent the editor from combining the line number with the • number in your text. Before storing the file to disk, you would use the command:

```
CHANGEC@&@@@
```

This will remove all of the "&" characters from your program (leaving any which may be within quotes).

A second solution is to put any shifted character at the beginning of the line. The shifted character is automatically removed from the line as it is entered, so the CHANGE command becomes unnecessary. This method is a bit dangerous however, since you must remember to put the shifted character into the line again whenever you go to edit, or you may get the same problem as before.

You must be between quotes for the editor to accept cursor control characters, but if you used quotes in the file, they would put the computer into the "quote-mode", causing cursor control characters to be displayed as reverse characters instead of performing their functions (<home> would print a reverse-s instead of moving the cursor to the home position). To solve this problem, the EXEC command will ignore a quote if it is the first character on a line. For example, to clear the screen from your EXEC file, put in a line such as:

```
1100 "<clear screen>
```

Once edited, the file may be stored on disk with a command such as:

```
PUT "0:FILEX"
```

The file "FILEX" may then be executed with the command:

```
EXEC "FILEX"
     or
EXEC FILEX
```

Any tape operation will halt the "EXEC" function, but manually pressing [RETURN] will allow the "EXEC" to continue.

USING EXEC TO RUN A PROGRAM

EXEC may be used to supply inputs for a program. Just set-up a file with the answers to all input prompts (in order), and EXEC will provide one answer for each input as the program runs. Remember that EXEC needs the cursor to be flashing to indicate that the computer is ready for input. BASIC "GET" commands will not work because they do not turn the cursor on (it must be the real PET cursor). If it is necessary for the EXEC command to provide characters for a "GET" command, you could signal EXEC to get a line with the command POKE 204,0". The EXEC command would immediately provide the response, and the cursor would actually never flash; however, the program would leave the cursor flashing if used without EXEC.

A program operated under EXEC must NEVER close a file to the disk command channel (secondary address = 15). CLOSING THE COMMAND CHANNEL WILL TERMINATE EXEC (control will be returned to the keyboard).

COMMAND

FIND

FUNCTION

FIND AND DISPLAY ALL OCCURRENCES OF A PATTERN WITHIN THE CURRENT PROGRAM.

SYNTAX

FIND [B] [E] [V] [P] [C] ["]<del><pattern><del>[<line-range>]

With the following options:

B	Display only if at the **B**eginning of a line
E	Display only if at the **E**nd of a line
V	Display only if an exact **V**ariable name match
P	Enable **P**attern matching ("#" as "wild-card")
C	Change only if in **C**ommand area (not in quotes, DATA, or REM)
"	Turn quote mode on (disable crunching)

<del> may be:

Any character not in either pattern except B, E, V, P, C, R, or ". Characters which are used by BASIC (such as *, /, +, =, etc.) are not recommended as they are not always what they appear to be.

<line-range> may be:

-<line A>	Up to (including) line A
<line A>-	From line A to end of program
<line A>-<line B>	From line A to line B
<line A>	Only in line A
<none>	Entire program

DESCRIPTION

The FIND command finds all locations within a BASIC program containing a given pattern, and displays the line. In the event that multiple matches are found In one line, the line will only be displayed once. This is an extremely flexible command, since by combining all of the search options there are more than 200 valid combinations (ego, FIND, FINDB, FINDPC, etc.).

The options B, E, V, P and C may be combined to give the desired selectivity. For example, "FINDBP" will display the line only if the match Is found at the beginning of the line and will use the pattern matching mode (a "#" in the search string will match any character In the text).

The **pattern matching mode** allows the use of the "#" character as a "wild-card". For example, the command:

```
FINDP"@PRESS"#" TO CONTINUE@
```

will ignore the character under the "#" and match all similar prompts.

Beware of BASIC commands, as they are stored as one byte each, so the command "FINDCP@####F$@" will not find "LOADF$" or "SAVEF$" because "LOAD"and "SAVE" are stored as single bytes.

The **quotation mark is outside the delimiters** to allow the quote mode to be turned on when the search pattern starts within quotes but there Is also a quote within the search string Itself.

At any time, the output may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL], or stepped by holding down the SPACE BAR and tapping [LOGO] or ended by pressing [RUN/STOP].

The variable ("V") mode will only display **exact variable name matches,** for example, the command:

```
FINDV@A@
```

will only display the variable "A" and will not find such variables as "AM ", "RA", "A$" or "A%". Care must be taken when long variable names are used within a program. Since BASIC uses only the first two letters of a variable, sometimes variables such as "CO" and "COUNT" are used interchangeably within a program. The command:

```
FINDV@CO@
```

will ignore the variable "COUNT" because it is not an exact match. This is a safety feature for the "CHANGE" command, to allow you to fix the problem when you accidentally use two long names with the same first two letters.

At first glance, It would seem reasonable to think that when editing a source code, you could isolate a label by using the "V" option. This will work sometimes, but not always. Tho reason Is that a source code is compacted by tokenizing it in tho same manner as a BASIC program in order to conserve memory. Therefore, although a label such as "MK34" would work nicely and would be distinguished from labels such as "MK342", the label "NEXT1" Is stored as [NEXT] [1] (two bytes) and would be ignored by the FINDV command because it is not a legal variable name.

The command ("C") option Is used to prevent accidentally matching a character in a string which has the same ASCII value as the token for the command. For example, the token for "LOAD" Is the same as the character for clear-screen, so to display all "LOAD" commands in a program you should use the command:

```
FINDC@LOAD@
```

The FINDC command will ignore any match **beginning with quotes**, however, it allows the match string to extend into quoted text.

SENDING OUTPUT TO THE PRINTER

The command "FIND" will list the result of the FIND command to the current printer device. Using the "SETP" command, the printer may be set to my device which does not require secondary addressing or a filename. This may include such devices as modems, ASCII printers, or even the screen (device #3).

SENDING OUTPUT TO A FILE

The standard method of sending the printout to a file oriented device, such as the disk drive or the tape, is to open a write file in the usual manner:

```
OPEN1,8,2,"FIND,S,W"
```

then, use the "CMD" command to set output to that file (must be on a separate line to prevent the word "READY." from being sent) such as:

```
CMD1,"";
```

The output of the standard FIND command will now go to the file. When the FIND command has finished, use the command:

```
CLOSE
```

to suspend output to that file. Note that "CLOSE" and not "CLOSE1" is used, as the SYSRES™ "CLOSE" command also terminates the CMD If necessary ('CLOSE1" will close the file but leave the CMD active, hence the "sticking cursor" bug).

The FIND output may be added to the end of an existing file by opening the file a second time in the form "<filename>,A ". This opens the file as an APPEND file, so That new data is added to the end of the original file.

EXAMPLES

```
FINDV@90@
```

Will find only the number 90. It will Ignore 900, 390, etc. and will ignore "90" within quotes. It will only find references to line 90 or the number 90 in algebraic expressions.

```
FIND"$GOTO 34$
```

Finds "GOTO 34" as it would be found within quotes (as in the message, "ENTER 'GOTO 34' AND PRESS [RETURN]").

```
FIND!THENGOTO!
```

Finds any "THENGOTO"s in the program.

```
FIND&SYSTEM(5)&-59999
```

Finds occurrences of "SYSTEM(5)" in lines up to and including line 59999.

COMMAND

GET

FUNCTION

READ A SEQUENTIAL FILE INTO THE EDITOR.

SYNTAX

GET "<filename>"[:start line]
GET <directory>

DESCRIPTION

SYSRES™ allows editing and storage/retrieval of sequential text files for use with the Commodore assembler, or any other application using sequential files. The GET command is used to read a sequential file into memory. It automatically tokenizes lines to provide as much as a five to one reduction in memory usage (at the cost of a slight increase in the time required to "GET" or "PUT" a data file).

The GET command essentially creates a program in memory, assigning arbitrary line numbers starting at 1000 (or a specified line) in steps of 10. A new line is generated each time a carriage-return (CHR$(13)) or a null (CHR$(0)) is received. If a starting line number is given, existing program lines up to that line will be left as they are, and new text will be added, in steps of 10, starting at the specified line. Note that the line range is preceded by a colon, instead of a comma. This is to prevent confusion with other file commands which allow device numbers causing the common error of getting a file starting at line 8 (GET"SOMETHING",8).

Text files containing shifted characters which are not in quotes will cause problems. If the program which is using the data doesn't mind (ego, it uses "INPUT#" to read the data), then you can "GET" the file, and immediately perform the command:

```
CHANGEBP@#@@"#@
```

This will put a quotation mark at the beginning of each line, allowing editing of the file. You cannot remove the quotes when you "PUT" the file back to disk, because the "uncrunching" function of the "PUT" command will cause unwanted words to replace the shifted characters.

In order to be totally compatible with BASIC, when entering a line, a question mark ("?") which is not within quotes will be converted to the word "PRINT", Although this is rarely an inconvenience, if a question mark is REQUIRED in a line, start the line with a quotation mark then use the command:

```
CHANGEB@"@@@<line#>
```

to delete the quote before storing the file to disk. The file will contain a question mark, as can be verified with an "@L" command, even though the question mark will be changed to the word "PRINT" if you "GET" the file again.

Since the GET command translates the data file into a format which is totally compatible with BASIC program encoding, all of the powerful SYSRES™ program manipulation commands are

usable when editing data files. You can move back and forth between text editing and BASIC programming with ease. You can even write basic routines to test out your theories without having to leave the editor and go to BASIC first … because you ARE in BASIC!

When using the editor, a number cannot normally be the first item on a line. To enter a line which begins with a number (as when creating an "EXEC" file to enter program lines) you must put some character between the line number used when editing and the number in the input line. For example:

```
1000 10 PRINT "THIS LINE IS ADDED"
```

would not work because the editor would think you were trying to enter line 100010 and would print "?SYNTAX ERROR". An acceptable solution would be to enter:

```
1000&10 PRINT "THIS LINE IS ADDED"
```

The "&" character would prevent the editor from combining the line number with the number in your text. Before storing the file to disk, you would use the command:

```
CHANGEC@&@@@
```

This will remove all of the "&" characters from your program (leaving any which may be within quotes).

A second solution is to put any shifted character at the beginning of the line. The shifted character is automatically removed from the line as it is entered, so the CHANGE command becomes unnecessary. This method is a bit dangerous however, since you must remember to put the shifted character into the line again whenever you go to edit, or you may get the same problem as before.

You must be between quotes for the editor to accept cursor control characters or shifted characters, otherwise shifted characters will be ignored, and cursor characters will perform their normal functions (eg., <home> would move the cursor to the home position).

Once edited, the file may be stored on disk with a "PUT" command.

USING THE TAPE DRIVE

The GET command will read a data file from tape if the disk device # is set to 1 using the command "SETD,1". The file may be edited in the normal manner and sent back to the tape with the "PUT" command.

To transfer a data file from tape to disk, you could use the following sequence:

```
SETD,1
GET""
SETD,8
PUT"DATA"
```

EXAMPLES

```
GET"DATA":1050
```

Reads the file "DATA" into the editor starting at line 1050. Any program lines (in memory) lower than line 1050 will be left, and the new data will be added from line 1050 on.

```
GET "LOADER/SRC" PRG
```

Reads the file "LOADER/SRC" using the directory.

COMMAND

KEY
KEYS

FUNCTION

DEFINE A KEY AS A SPECIAL FUNCTION.

SYNTAX

KEY
KEY"
KEY""
KEYS
KEY <character expression>, <code> [+128]
KEY <character expression>, <string expression>

DESCRIPTION

The KEY command allows any SHIFTED key to be defined as a BASIC or SYSRES™ command or as a string. All keys may be defined as commands a the same time, as well as up to 15 keys defined as strings with a total length of up to 255 characters.

KEY functions are automatically disabled when a BASIC program is running. To re-connect the KEY functions .after running a program, or after using the tape drive, simply press [RETURN), or enter any command (the "#" command maybe used since it ignores the rest of the line and tells you which version of SYSRES™ is active).

If you want to keep the KEY functions on in a program (perhaps to provide pre-determined inputs for debugging purposes), enter ":RUN" to start the program. "GOTO" and "CONT" will also leave the key functions active. A speed decrease of from 1 to 5 percent will result from executing a program with interrupt functions active.

The "KEY" command can program any SHIFTED key, CONTROL key (key with [CTRL] pressed), LOGO key (key with the LOGO key pressed), or F1 through F8. Some CONTROL keys should not be used since they coincide with cursor, color, and function keys.

None of the KEY functions are active when the cursor is within quotes or when there are inserts outstanding (quote or insert mode). This allows all normal shifted characters to be entered into strings. If you need to type a shifted letter and you are not within quotes (as when editing a line and you go between quotes without typing a quote), simply press the insert key ([SHIFT] and [INST/DEL]) before typing the shifted letter. This will disable the KEY function for the next character only.

SETTING UP KEY FUNCTIONS

Mode 1: Automatic Keywords

To set-up a key to generate a standard BASIC or SYSRES™ command, enter the command:

KEY <key to replace>,<command code>

where the COMMAND CODE is taken from table 4. The key to replace may be any string expression (only the first character Is used) and may be given as the shifted or unshifted character (except for some keys on the business keyboard which do not print the same character, a graphic symbol, or a capital letter when pressed). This feature allows the number pad keys to be set to produce different commands than the top-row keys.

For example, if you wanted the command "PRINT#" to be entered each time you press [SHIFT] [P], you would enter:

```
KEY "P",24
```

Mode 2: Automatic Short-Form Keywords

If you want the commands to come out in abbreviated form (the first one or two letters of the word and the next letter shifted), add 128 to the code for the command. For example, to use the "pR" (the short-form of "PRINT#") in the above example, you would use:

```
KEY "P",24+128
```
or
```
KEY "P",152
```

SYSRES™ takes into account which commands have two character abbreviations, such as DATA (dA) or VERIFY (vE) and which ones require three, like RESTORE (reS) and GOSUB (goS). Commands which do not have a short form because they must be entered completely will be taken to have a two character short-form, for example, "INPUT" will be shortened to "iN" which is the abbreviation for "INPUT#".

If a command requires an open-bracket, SYSRES™ will automatically provide one, as in the command "LEFT$(". This alleviates the confusion where "tA" is the abbreviation for "TAB(" while the abbreviation for "MID$(" is "mI(".

Some file commands, such as GET#, RECORD# and DOPEN#, must be followed by a "#" which is not provided by SYSRES™, except with the commands "INPUT#" and "PRINT#".

Table 4.

LIST OF COMMAND CODES

LONG	SHRT	COMMAND	LONG	SHRT	COMMAND	LONG	SHRT	COMMAND	LONG	SHRT	COMMAND
113		!	26	154	CONT	72	200	LEFT$	59	187	RND
115		#	83	211	COPY	67		LEN	117	245	RUN
44		*	62		COS	8	136	LET	10	138	RUN
42		+	3	131	DATA	122	250	LIST	105	233	SAVE
43		-	78	206	DCLOSE	27	155	LIST	20	148	SAVE
100		/	22	150	DEF	109	237	LOAD	89	217	SCRATCH
45		/	97	225	DELETE	19	147	LOAD	111	239	SETD
51		<	6	134	DIM	60		LOG	112		SETP
50		=	90	218	DIRECTORY	103	231	MERGE	52	180	SGN
108		>	86	214	DLOAD	74	202	MID$	63	191	SIN
49		>	77	205	DOPEN	93	221	MON	38	166	SPC(
107		@	85	213	DSAVE	34	162	NEW	58	186	SQR
54	182	ABS	124	252	DUMP	2	130	NEXT	41	169	STEP
47	175	AND		128	END	40	168	NOT	16	144	STOP
116	244	APPEND	118	246	EXEC	94	222	OLD	68	196	STR$
84	212	APPEND	61	189	EXP	17	145	ON	30	158	SYS
70	198	ASC	101	229	FIND	31	159	OPEN	35	163	TAB(
65	193	ATN	37		FN	48	176	OR	64		TAN
92	220	AUTO	1	129	FOR	66	194	PEEK	39	167	THEN
82	210	BACKUP	56	184	FRE	23	151	POKE	36	164	TO
120	248	BLOAD	99	227	GET	57		POS	106	234	TRACE
121	249	BRUN	33	161	GET	25		PRINT	55	183	USR
87	215	CATALOG	75	203	GO	24	152	PRINT#	69	197	VAL
96	224	CHANGE	13	141	GOSUB	98	226	PUT	125	253	VERIFY
71	199	CHR$	9	137	GOTO	7	135	READ	21	149	VERIFY
110	238	CLOSE	80	208	HEADER	79	207	RECORD	18	146	WAIT
32	160	CLOSE	11		IF	15		REM	91	219	WHY
28	156	CLR	5		INPUT	88	216	RENAME	95		
123	251	CMD	4	132	INPUT#	104	232	RENUMBER	46		
29	157	CMD	53		INT	12	140	RESTORE	114		
81	209	COLLECT	119	247	KEY	14	142	RETURN			
76		CONCAT	102	230	KILL	73	201	RIGHT$			

Mode 3: Automatic Strings

The third method of defining a key is the form:

KEY <key to replace>,<string expression>

This mode allows any key (up to 15 keys at my one time) to be defined as any string. The total length of all of the defined strings may be up to 255 characters. If you attempt to define more than 15 string keys, or if the total length of string keys would exceed 255 characters, a "? STRING TOO LONG" message will appear.

Defining key functions Is so simple that you will probably find many occasions to use them. For example, If you are working on a subroutine at line 400, you might want to define the function:

```
KEY"1","<clear-screen>LlST 400-499"+CHR$(13)
```

When a shifted "1" (from the number pad) Is pressed, the screen would clear, and lines 400 to 499 would be listed.

If you like to keep a trail of you work (If you have Murphyphobia) you might like to replace the shifted "@" key with the command:

```
KEY"@","SAVE"+CHR$(34)+"TEMP"+CHR$(34)+"+TI$"+CHR$(13)
```

(Read-on before you type this in!)

This will enter the command:

```
SAVE"TEMP"+TI$
```

each time a shifted "@" is pressed.

Remember that the string is evaluated in the same manner as the header in the "AUTO" command, so If you are trying to set-up a complex KEY function, it may be wise to define a string variable and try printing It first, then put it into a KEY function. For example, the previous command could have been tested first by using the following sequence:

```
Q$=CHR$(34)

READY.
CR$=CHR$(13)

READY.

A$="SAVE"+Q$+"TEMP"+Q$+"+ TI$"+CR$

READY.
?A$
SAVE"TEMP"+TI$

READY.
KEY "@",A$
```

A BETTER WAY

You have probably noticed how often you must use the expressions "+CHR$(34)+" (for quotation marks) and "+CHR$(13)" (for carriage-return) in this command. To make life easier, this command will accept shifted-Q for quotes and shifted-Z for carriage-return within the string expression. In the graphics mode, the PET will display a large dot for shift-Q and a diamond for shifted-Z (the lowercase mode will display capital Q and Z). For example, the expression (as seen in the lowercase mode):

```
"saveQtempQ+ti$Z"
```

is the same as the expression:

```
"save"+chr$(34)+"temp"+chr$(34)+"+ti$"+chr$(13)
```

and would evaluate to be:

```
save"temp"+ti$ <cr>
```

The shifted-Q and shifted-Z were selected because they are not used in command abbreviations and are rarely used.

NOTE: This is a feature of the "AUTO" and "KEY" commands only and does not affect the interpretation of other strings during normal operation.

CLEARING KEY FUNCTIONS

To clear a string function from an individual key, enter the command:

```
KEY  "<key to clear>",""
```

The specified key will be reset to the command which had previously been set to that key or, if no command has been set, the key will revert to its normal character. All string functions may be cleared at once with the command:

```
KEY""
```

To clear a keyword function, use the command:

```
KEY"<key to clear>",0
```

The specified key will return to normaL

If you want to clear all key functions use the special command:

```
KEY """
```

This will reset all keys to their normal condition (actually, the third character can be ANYTHING; it is easiest to simply type a third quote).

ENABLING AND DISABLING KEY FUNCTIONS

The KEY functions are automatically enabled whenever you define a key. To turn on previously defined KEY functions without entering a new key definition, use the command:

```
KEYS
```

To disable the KEY functions without clearing the definitions (so the command "KEYS" can re-enable the same functions later), enter the command:

```
KEY
```

DEFAULT KEY DEFINITIONS

When you first boot SYSRES™, some of the keys are pre-defined (see table 5). To activate the default KEY functions, simply enter the command:

KEYS

The default KEY definitions use the LOGO key. The key definitions are the same as listed in Table 5 except that the LOGO key is used (eg. pressing [LOGO] and [R] will produce "RENUMBER"). This means that programmed keys do not interfere with command abbreviations.

If you want to make your own set of default key definitions, you can set up an EXEC file to automatically set-up a complete set of KEY functions. The first line in your EXEC file should clear the existing KEY functions in case there are any string-definitions which could result in a "?STRING TOO LONG" error.

Table 5:

DEFAULT KEY DEFINITIONS	
A	AUTO
C	CHANGE
D	DUMP
E	EXEC
F	FIND
G	GET
I	INPUT
K	KEY
L	LIST
M	MERGE
N	NEXT
O	OPEN
P	PRINT#
R	RENUMBER
S	SAVE
T	TRACE
V	VERIFY
W	WHY

EXAMPLES

NOTE: The following examples are shown as they would appear when the computer is in the lowercase mode ([CTRL] [N]) to make them more readable.

```
key,"x","gosub1000:Z"
```

This example will call a subroutine at line 1000 in your program, then return to the "READY." mode. Notice that a colon must follow the line number so that BASIC will know that it came from an input line and not from within the program.

```
key"[","a$= <3 spc> <19 right> <3 spc> Z <clr> ?Q/Qa$:?Q:sAQa$Z <home> <down> ZZ <clr> @$Z
```

Although this may seem a bit complex and confusing, it is worth it if the directory is displayed on the screen, press a shifted-"]" when the cursor is beside the desired program and it will be transferred from the disk to the tape. Similar functions can be set-up for sequential files (use a

SETD,8/GET/SETD,1/,PUT sequence) or for VIC programs (use NEW/APPEND/:SAVE). A challenge would be to write function to transfer from tape to disk. Remember that by the time the tape is running, there must be no more than 9 characters pending in the keyboard buffer. You may have to use two keys.

COMMAND

KILL
KILL*

FUNCTION

DISABLE SYSRES™.

SYNTAX

KILL
KILL* (Optional syntax)

DESCRIPTION

The KILL command turns SYSRES™ completely OFF.

After a "KILL" command, SYSRES™ may be re-activated with the command: SYS 52992

The KILL command will leave the current BASIC program and any machine-language monitor extensions intact.

Booting SYSRES™ will not affect a BASIC program in memory, so you may re-boot SYSRES™ at any time.

COMMAND

LIST

FUNCTION

LIST THE CURRENT PROGRAM IN BASIC MEMORY.

SYNTAX

LIST <line-range>

<line-range> may be:

- <line A>	Up to (including) line A
<line A> -	From line A to end of program
<line A> - <line B>	From line A to line B
<line A>	Only line A
<none>	Entire program

DESCRIPTION

The LIST command will list a BASiC program (or the lines of text currently in the editor) in the same manner as the BASIC "LIST" command, except that the word "READY." will not be listed. This allows listings to be sent to a printer or to the disk drive without any extra garbage (extra linefeeds, carriage-returns, and the infamous "READY.") being sent. The suppression of "READY." also makes the transition between listing with the "LIST" command and listing with the scrolling feature much smoother. It may look a bit strange, however, to type the word "LIST" when there is no program in memory and simply have the cursor return, with no message indicating the absence of a program.

At any time, the output may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL], or stepped by holding down the SPACE BAR and tapping [LOGO] or ended by pressing [RUN/STOP].

LISTING TO THE PRINTER

The command "*LIST" will send the listing to the current printer device. Using the "SETP" command, the printer may be set to any device which does not require secondary addressing or a filename. This may include such devices as modems, ASCII printers, or even the screen (device #3).

If the printer device has been selected as <device number>+64, program listings will be formatted with one command per line, spaces between commands, and for-next loops indented by one space for each level of nesting. See the description of the "*" command for further details.

LISTING TO A FILE

There are two methods of sending listings to a file oriented device, such as the disk drive or the tape, (only the first method works with tape). The standard method is to open a write file in the usual manner:

OPEN1,1,2,"LISTlNG,S,W"

then, use the "CMD" command to set output to that file (must be on a separate line to prevent the word "READY." from being sent) such as:

CMD1,"";

The output of the standard LIST command will now go to the file. When the listing is completed, use the command:

CLOSE

to suspend output to that file. Note that "CLOSE" and not "CLOSE1" is used, as the SYSRES™ "CLOSE" command also terminates the CMD if necessary ("CLOSE1" will close the file but leave the CMD active, hence the "sticking cursor" bug).

The second method of sending output to a file is usually used when the format option is desired. In this case use the special file number 224 In the example:

OPEN224,8,2,"1:FMLlST,S,W"
CMD224,"";

When the current CMD file is 224, you may use the "*LIST" command to list to that file. This only works once. The file is automatically closed after a "*" command. Note: for formatting to take place, the actual printer device must be set to format mode (eg., SETP,4+64 has previously been done).

A listing may be added to the end of an existing file by opening the file a second time In the form "<filename>,A". This opens the file as an APPEND file, so that new data Is added to the end of the original file. This might be used to connect several listings together, so that they may be printed at once, or be processed through some sort of output program.

EXAMPLES

```
LIST 100-
```

This will list the prop-am from line 100 to the end of the program.

```
*LIST
```

This will list the entire program to the printer.

```
SETP,3+64 -LIST
*LIST
```

This will print a formatted listing to the screen (It looks the best on the 80 column screen).

COMMAND

LOAD

FUNCTION

LOAD A BASIC PROGRAM (DEFAULTS TO THE DISK DRIVE).

SYNTAX

LOAD
LOAD ["<filename>") [,<device #>]
LOAD <directory>

DESCRIPTION

The LOAD command, like all file commands under SYSRES™, may be used from the directory by typing the word "LOAD" in the left margin beside the name of the desired program, then pressing [RETURN]. If the file is not a program ("PRG") then a "?SYNTAX ERROR" will be returned.

When a filename is specified, LOAD will default to the disk drive if no device number is specified. If just the word "LOAD" alone is entered, the next program will be loaded from the TAPE drive.

If you want to load a program, with a specified name, from the tape drive, you must use the form:

```
LOAD "<filename>",1
```

or you could force the normal BASIC interpretation of the LOAD command by preceding the command with a colon, as in:

:LOAD "<filename>

This will load the specified program from the tape.

EXAMPLES

```
LOAD "MYPROG
```

This will load the program "MYPROG" from the disk.

```
LOAD "REWRITER 3.0" PRG
```

This is an example of using the LOAD command from the directory. The "/" command is preferred for this because it doesn't print the "SEARCHING/ LOADING/ READY." sequence.

COMMAND

MERGE

FUNCTION

MERGE (OVERLAY) A PROGRAM FROM DISK INTO THE PROGRAM IN MEMORY.

SYNTAX

MERGE "<filename>"
MERGE <directory>

DESCRIPTION

The MERGE command performs a true program merge. The lines from the specified file are inserted into their proper places. If a line exists in the program in memory with the same line-number as a line in the program from disk, the line in memory will be deleted and the line from disk will be put in its place.

If the lines in the program on disk are out-of-sequence, they will automatically be re-ordered as they are brought into memory. This feature 'can be used in conjunction with the RENUMBER command to move blocks of program lines with relative ease.

The MERGE command, like all file commands under SYSRES™, may be used from the directory by typing the word "MERGE" in the left margin beside the name of the desired program, then pressing [RETURN]. For example:

```
MERGE "SUB.INPUT/100" PRG
```

will merge the subroutine "SUB.INPUT/100" into the program in memory. This makes it very easy to maintain a subroutine library on disk. Imagine never having to type in another input routine!

The MERGE command is considerably slower than the LOAD and APPEND commands because the program in memory must continually be moved around to make room for new lines. Part of the reason it is slow is that every time a line is added, it is properly linked into place. This means that if an "?OUT OF MEMORY ERROR" occurs the program will be intact (the MERGE command will continue to bring in each line to see if it will fit, until the end of the file is reached). This is the only command which allows you to load part of a program into your machine which is too long to fit completely.

Due to a bug in Microsoft BASIC, the MERGE command **will THROW AWAY any incoming lines with more than 121 characters.** If a routine has a line more than 121 bytes long when tokenized, use the APPEND command.

It is recommended that you use the APPEND command if all of the line-numbers in the incoming file are greater than any lines in the program in memory. Since the APPEND command does not need to move the BASIC code around, it will be faster.

EXAMPLE

To demonstrate the MERGE command we will use it on a small program. User input is BOLDFACE.

```
LIST
      200 REM THIS PROGRAM IS IN MEMORY
      210 GOSUB 500
      220 IF A > 2 THEN PRINT A"-"A$
      230 IF S THEN 900
      240 GOTO 210
      900 PRINT"END OF FILE"
      910 END
@L"SUBROUTINE

      100 REM THIS PROGRAM IS ON DISK
      110 OPEN 1,8,2,"DATA"
      500 INPUT#1,A,A$
      510 S=ST
      520 RETURN
      910 CLOSE 1
      920 END

MERGE"SUBROUTINE
LIST
      100 REM THIS PROGRAM IS ON DISK
      110 OPEN 1,8,2,"DATA"
      200 REM THIS PROGRAM IS IN MEMORY
      210 GOSUB 500
      220 IF A> 2 THEN PRINT A"-"A$
      230 IF S THEN 900
      240 GOTO 210
      500 INPUT#1,A,A$
      510 S=ST
      520 RETURN
      900 PRINT"END OF FILE"
      910 CLOSE 1
      920 END
```

COMMAND

MON

FUNCTION

BREAK TO THE CURRENT MACHINE LANGUAGE MONITOR.

SYNTAX

MON

DESCRIPTION

The MON command will transfer control of the computer to the current machine-language-monitor via the 6502 BRK instruction. Unlike the command "SYS", the MON instruction does not increment the stack pointer each time the monitor is entered. The monitor is also properly initialized so that an accidental "G" instruction will return you to BASIC (as opposed to the deep reaches of memory space).

The MON instruction should always be used to go into the monitor because it inserts a trap to allow the "*" command to be used to send the output of any command to the printer. Note that output from the monitor is NOT vectored through the ASCII converter routine, but the monitor only outputs uppercase characters anyway. For special cases you can always use the SCREENPRINT function.

The monitor can be driven by an EXEC file, although with some monitors you may have to press [RETURN] once in the monitor to get the EXEC to continue (of course the EXEC command can only be started from BASIC).

MONITOR VERSIONS

Since there is no built-in Machine-Language-Monitor (MLM) in the Commodore-64, the "MON" command is only functional if a MLM has been loaded. (If you have a "VLINK-64" cartridge, "MON" will put you into the VLINK mini-monitor). A BRK trap is provided when no MLM is available. The "MON" command (or a SYS to a location containing a zero) will print:

B<page>:Y<yreg>,X <xreg> ,A<acc> ,S<psw> ,@<address>

where:

<page> = Memory page where BRK occurred.
<yreg> = Decimal contents of 6510 register Y.
<xreg> = Decimal contents of 6510 register X.
<acc> = Decimal contents of 6510 register.
<psw> = Decimal contents of 6510 processor status.
<address> = Decimal address where BRK occurred (+2)

Included with the SYSRES™ system is a copy of the public-domain machine language monitor "XMON". It has been specially written to be 100% compatible with SYSRES™, though it will run independently from SYSRES™ also. XMON commands are listed in Appendix A.

There are 4 copies of XMON on your SYSRES™ disk: XMON64C(50135), XMON64H(29900), and XMON64T(38083), which you may use with the BRUN command, and XMON64L(BASIC) which you load and run. The number in parenthesis indicates the load address (and SYS to re-connect after a [RESTORE].

Note: On initial entry via BRUN, the SYSRES™ repeating-key functions will be disabled. Exit (with X) and re-enter with MON to connect the repeat and scroll functions.

COMMAND

OLD

FUNCTION

RESTORE THE BASIC PROGRAM AFTER A "NEW".

SYNTAX

OLD

DESCRIPTION

If you accidentally "NEW" a program and you want to get it back, type "OLD" and your old program will be returned to you.

The "OLD" command assumes that you would rather run the risk of crashing the system than to not get your program back, so if you type "OLD" and there never was a program there in the first place … you will need to reset the computer.

With most similar commands on other systems, the program will be lost forever if you enter a variable name between the time you entered the "NEW" command and the "OLD" command (most syntax errors will create a variable). SYSRES™ recognizes this condition and puts in a line:

```
22616 XXXXXTHE REST OF THE FIRST LINE
```

The first line may now be removed and re-entered correctly.

COMMAND

PUT

FUNCTION

SEND PROGRAM TEXT TO DISK AS A SEQUENTIAL FILE.

SYNTAX

PUT"<filename>":<line-range>
PUT<directory>

<line-range> may be:

- <line A>	Up to (including) line A
<line A> - <line A>	From line A to end of program
- <line B>	From line A to line B
<line A>	Only line A
<none>	Entire program

<filename> may be contain (in these positions):

"[@][<drive>:]<name>[,A]"

with the following options:

@	Replace an existing file with the same name.
<drive>:	Specified drive (otherwise use the current drive).
,A	Add to the end of an existing file.

DESCRIPTION

SYSRES™ allows editing and storage/retrieval of sequential text files for use with the Commodore assembler, or any other application using sequential files. The PUT command is used to write the contents of memory to disk as a sequential file. It automatically de-tokenizes lines to provide as much as a five-to-one reduction in memory usage (at the cost of a slight increase in the time required to "GET" or "PUT" a data file).

The PUT command essentially lists the program in memory (or a specified range of lines), stripping off the line numbers, to create a sequential file on disk containing the text of the program. Each line is terminated with a carriage-return (CHR$(13)). If a line-range is given, only the specified section of text will be sent to the file. **Note that the line range is preceded by a colon, instead of a comma.** This is to prevent confusion with other file commands which allow device numbers, causing the common error of putting only line 8 to a file (PUT"SOMETHING",8).

If you want to send the program to disk **including the line numbers,** see the "LIST" and "@L" commands.

You do not need to fear using the "@" at the beginning of a filename to replace a file. SYSRES™ recognizes the "@" scratches the file, then writes the new version. If you do not

specify a drive number, the last drive referenced will be used (SYSRES™ also fixes the problems with opening a write file with no drive number).

Text files containing shifted characters which are not in quotes will cause problems. If the program which is using the data doesn't mind (eg., it uses "INPUT#1" to read the data), then you can enter a quotation mark at the beginning of each line, allowing editing of the file. You cannot remove the quotes when you "PUT" the file back to disk, because the "uncrunching" function of the "PUT" command will cause un-wanted words to replace the shifted characters.

In order to be totally compatible with BASIC, when entering a line, a question mark ("?") which is not within quotes will be converted to the word "PRINT". Although this is rarely an inconvenience, if a question mark is REQUIRED in a line, start the line with a quotation mark then use the command:

```
CHANGEB@"@@@<line#>
```

to delete the quote before storing the file to disk. The file will contain a question mark, as can be verified with an "@L" command, even though the question mark will be changed to the word "PRINT" if you "GET" the file again.

Since the data is edited in a format which is totally compatible with BASIC program encoding, all of the powerful SYSRES™ program manipulation commands are usable when editing data files. You can move back and forth between text editing and BASIC programming with ease. You can even write BASIC routines to test out your theories without having to leave the editor and go to BASIC first … because you ARE in BASIC!

When using the editor, a number cannot normally be the first item on a line. To enter a line which begins with a number (as when creating an "EXEC" file to enter program lines) you must put some character between the line number used when editing and the number in the input line. For example:

```
1000 10 PRINT"THIS LINE IS ADDED"
```

would not work because the editor would think you were trying to enter line 100010 and would print "?SYNTAX ERROR". An acceptable solution would be to enter:

```
1000&10 PRINT"THIS LINE IS ADDED"
```

The "&" character would prevent the editor from combining the line number with the number in your text. Before storing the file to disk, you would use the command:

```
CHANGEC@&@@@
```

This will remove all of the "&" characters from your program leaving any which may be within quotes.

A second solution is to put any shifted character at the beginning of the line. The shifted character is automatically removed from the line as it Is entered, so the CHANGE command becomes unnecessary. This method is a bit dangerous however, since you must remember to put the shifted character Into the line again Whenever you go to edit, or you may get the same problem as before.

You must be between quotes for the editor to accept cursor control characters or shifted characters. Otherwise shifted characters will be ignored, and cursor characters will perform their normal functions (eg., <home> would move the cursor to the home position).

Once edited, the file may be stored on disk with a "PUT" command.

USING THE TAPE DRIVE

The PUT command will write a data file to the tape if the disk device # is set to 1 using the command "SETD,1".

To transfer a data file from disk to tape, you could use the following sequence:

```
    SETD,8
GET"<filename>"
    SETD,1
PUT"<filename>"
```

EXAMPLES

To demonstrate the PUT command we will use it on a small source code (for the Commodore assembler). User input is **BOLDFACE**.

```
LIST
     1000 *=$2000
     1010 INIT CLC ;CLEAR CARRY
     1020 LOA VECTOR ;GET VECTOR
     1030 ADC OFFSET ;ADD OFFSET
     1040 STA POINT ;SAVE RESULT (L)
     1050 LDA VECTOR+1 ;DO FOR HIGH TOO
     1060 ADC OFFSET+1
     1070 STA POINT+1 ;SAVE RESULT (H)

PUT"@0:INIT/SRC"              PRG  (Note: done from directory)
PUT":INIT.1:-1030
PUT"CLC":1010

@L"INIT /SRC"S
     *= $2000
     INIT CLC ;CLEAR CARRY
     LDA VECTOR ;GET VECTOR
     ADC OFFSET ;ADD OFFSET
     STA POINT ;SAVE RESULT (L)
     LDA VECTOR+l ;DO FOR HIGH TOO
     ADC OFFSET +1
     STA POINT+l ;SAVE RESULT (H)

@L"INIT.1"S

*= $2000
INIT CLC ;CLEAR CARRY
```

```
      LDA VECTOR ;GET VECTOR
      ADC OFFSET ADD OFFSET
```

@L"CLC"S

```
INIT CLC ;CLEAR CARRY
```

COMMAND

RENUMBER

FUNCTION

RENUMBER ALL OR PART OF A PROGRAM.

SYNTAX

RENUMBER <first number>,<step>[,<line-range>]

<line-range> may be:

- <line A>	Up to (including) line A
<line A> -	From line A to end of program
<line A> - <line B>	from line A to line B
<line A>	Only line A
<none>	Entire program

DESCRIPTION

The RENUMBER command will change the line numbers of all or part of the current BASIC program into even increments. All line references following IF… THEN, GOTO, GOSUB, ON … GOTO, ON… GOSUB, LIST, RUN, and GO TO are corrected (note: GO TO is not handled by most other renumbering routines!).

Instead of changing references to non-existing lines to a useless number like 63999 or 65535, RENUMBER in SYSRES™ will flag all non-existing lines with an ampersand ("&"), leaving the number as it was before renumbering. To display all bad references you can use:

```
FINDC@&@
```

The RENUMBER command will renumber the entire program if you do not specify a line-range. You must always specify a first line and a step (an early evaluation version had a default start and step of [1000,10] and of the five we sent out, we got four complaints that the default should be [1,1], [10,10], [100,10], and [100,1], so we dropped the default). For example:

```
RENUMBER 100,10
```

will renumber the entire program in steps of 10 with the first line being 100.

If you want to renumber a part of the program, specify a line range in the standard format. For example:

```
RENUMBER 100,10,100-399
```

will renumber only lines between 100 and 399. When a line-range is specified, only references to undefined lines which would be within the specified range will be flagged. If, in the previous example, line 50 contained "GOSUB 103" and there was no line 103 in the program, after the RENUMBER command line 50 would read "GOSUB&103". If line 110 or line 450 contained "GOTO 90" (there mayor may not be a line 90) they would not be affected by RENUMBER because 90 is not in the range "100 - 399".

CHANGING THE ORDER OF LINES

You can use the RENUMBER command to move blocks of lines from one part of the program to another. The following example will move lines 200 - 299 to 500. User input is **BOLDFACE**.

```
LIST
      100 IF A < 5 THEN 200
      105 B =0:GOSUB 300:GOTO 250
      200 B=3
      202 A=A+1
      250 FOR I = 1 TO 10:PRINT X(I,A):NEXT
      270 GOTO 600
      300 A=0
      310 RETURN
      600 IF B = 3 THEN 100
      610 END
RENUMBER 500,10,200-299

READY.
LIST
      100 IF A < 5 THEN 500
      105 B =0:GOSUB 300:GOTO 520
      500 B=3
      510 A = A + 1
      520 FOR I = 1 TO 10:PRINT X(I,A) :NEXT I
      530 GOTO 600
      300 A=0
      310 RETURN
      600 IF B = 3 THEN 100
      610 END
```

Notice that the program lines are now out of sequence (the program will usually not work at this point). To put the lines in the correct order, the following commands must be performed (using the filename of your choice):

```
SAVE"TEMP
NEW
MERGE"TEMP"
```

The program lines are now in sequence, as can be verified with the LIST command. If we list our example program, the result would be:

```
LIST
      100 IF A < 5 THEN 500
      105 B=0:GOSUB 300:GOTO 520
      300 A=0
      310 RETURN
      500 B=3
      510 A=A + 1
```

```
520 FOR I = 1 TO 10:PRINT X(I,A):NEXT I
530 GOTO 600
600 IF B = 3 THEN 100
610 END
```

WARNING: **DO NOT attempt to move two sections of a program at once. If a RENUMBER command with line-range is performed on a program which has lines out of sequence unexpected results may occur.**

The RENUMBER command will not allow you to create line numbers greater than 6340972. The range Is checked BEFORE the renumbering process begins, so the message "?SYNTAX ERROR" will be displayed and the program will be unchanged. No attempt is made to determine if renumbering will cause line number conflicts as such checks would hamper the ability to move lines. BE SURE TO DOUBLE-CHECK BEFORE ENTERING A RENUMBER WITH LINE-RANGE.

REMOVING PROGRAM LINES

The RENUMBER command can help when removing lines from a complex program. The following examples deal with removal of REM statements from a program; however, the same techniques may be applied to other applications. To remove the REM statements from a program, you would use the command:

CHANGECR@REM@@@

This would leave a colon (":") on all lines which contained only a REM statement. To find those lines, we could now use the command:

FINDBE@:@

If you have a printer it would probably be more convenient to use "*FINDBE@:@" to print the list of blank lines for reference (With a little ingenuity you might, for long programs with many REM's, send the results to the disk drive, then have a program which builds an EXEC file to perform the following steps automatically).

Next, to safely eliminate the blank lines, we must make sure that all references to each blank line are pointed to the following line. This is done by renumbering each blank line with the same line as the one after it. If you remove the line (by typing the line number and pressing [RETURN]) ONLY THE FIRST OCCURRENCE OF THAT LINE NUMBER WILL BE REMOVED (type the number again if there is more than one line to remove). The following sequence demonstrates the entire procedure.

```
LIST
      10 GOSUB 30
      20 GOT060
      30 REM INPUT A$
      40 REM
      50 INPUT A$:RETURN
      60 END
CHANGECR@REM@@@

30 :
```

```
40 :

READY.
LIST
      10 GOSUB 30
      20 GOT060
      30 :
      40 :
      50 INPUT A$ :RETURN
      60 END
RENUMBER 50,0,30-50

READY.
LIST
      10 GOSUB 50
      20 GOT060
      50 :
      50 :
      50 INPUT A$:RETURN
      60 END
      50
      50
LIST
      10 GOSUB 50
      20 GOT060
      50 INPUT A$:RETURN
      60 END
```

SPEED IMPROVEMENTS AND MEMORY SAVINGS

There are many ways in which the RENUMBER command can help to make programs run faster and use less memory. If you follow these tips, you may notice a remarkable improvement.

Move all DATA statements to the end of the program and NEVER put them in a place where they will be executed.

Move commonly used subroutines to the beginning of the program. Move routines which are seldom used to the end (and completely remove routines which are never used).

Renumber the program in steps of 1 (this will save memory). If a program uses many forward branches (to higher line numbers) then renumber the program so that all forward branches go to a line number higher than the next multiple of 256 ("300 GOTO 512" may be up to **200 TIMES FASTER** than "300 GOTO 301"!)

Remove unnecessary REMs and put notices (eg., copyright) at the END of the program (they don't seem to be noticed at the beginning anyway).

COMMAND

RUN

FUNCTION

EXECUTE THE BASIC PROGRAM (TURN OFF INTERRUPT FUNCTIONS).

SYNTAX

RUN [<starting line>]

DESCRIPTION

The RUN command functions the same way as BASIC "RUN" except that any garbage on the screen is ignored.

The RUN command automatically turns SYSRES™ off and reduces its interface to a minimum when a BASIC program is running. It also disconnects "interrupt" functions (the functions which are always active and do not have an actual command-word ie. repeat, scroll, screenprint and defined keys). To re-connect these functions after running a program, or after using the tape drive, simply press [RETURN], or enter any command (the "#" command may be used since it ignores the rest of the line and tells you which version of SYSRES™ is active).

If you want to keep the interrupt functions on in a program (perhaps to use the repeating keys or screenprint, to use the defined keys to provide pre-determined inputs, or even to use scrolling to list parts of the program as it is running), enter ":RUN" to start the program. "GOTO" and "CONT" will also leave the interrupt functions active. Note that the keys will not repeat unless the real cursor is flashing, as in an INPUT statement. A speed decrease of from 1 to 5 percent will result from executing a program with interrupt functions active.

When operating under an EXEC file (see "EXEC"), SYSRES™ will remain fully connected at all times. This allows a program to be RUN with an EXEC file providing all of the inputs.

The special command "*RUN" WILL NOT send all program output to the printer as might be expected, it is used to send the TRACE output to the printer (see the TRACE command). To send program output to the printer, use the "CMD" and "PRINT#" commands within your program. Note: "CMD" will not work if "*RUN" is used for program tracing.

COMMAND

SAVE

FUNCTION

SAVE THE CURRENT BASIC PROGRAM TO DISK (OR TAPE).

SYNTAX

SAVE
SAVE ["<filename>"] [<device #>]
SAVE <directory>

<filename> may contain (in these positions):

"[@][<drive>:] <name>"

with the following options:

@	Replace an existing file with the same name.
<drive>:	Specified drive (otherwise use the current drive).

DESCRIPTION

When a filename is specified, SAVE will default to the disk drive if no device number is specified. If just the word "SAVE" alone is entered, the next program will be saved to the TAPE drive.

If you want to save a program, with a specified name, to the tape drive, you must use the form:

```
SAVE "<filename>",1
```

or you could force the normal BASIC interpretation of the SAVE command by preceding the command with a colon, as in:

```
:SAVE "<filename>
```

This will save the specified program to the tape.

The SAVE command, like all file commands under SYSRES™, may be used from the directory. You may "SAVE" the program using the directory by typing:

```
SAVE"@:
```

in the left margin beside the name of the desired program, then pressing [RETURN]. Notice that this fits exactly into the margin so that the colon (":") covers the first quotation mark of the directory entry. This command will replace the program listed in the directory with the current program from memory. If the file is not a program ("PRG") then a "?SYNTAX ERROR" will be returned.

You do not need to fear using the "@" at the beginning of a filename to replace a file. SYSRES™ recognizes the "@", scratches the file, then writes the new version. If you do not specify a drive number, the last drive referenced will be used (SYSRES™ also fixes the problems with opening a write file with no drive number).

EXAMPLES

SAVE"MYPROG

This will save the program "MYPROG" to the disk.

SAVE"@:REWRITER 3.0" PRG

This is an example of using the SAVE command from the directory. The program "REWRITER 3.0" will be erased from the disk and the new version from memory will replace it.

COMMAND

SETD

FUNCTION

SET DEFAULT DISK DEVICE NUMBER.

SYNTAX

SETD,<device #>

DESCRIPTION

The SETD command defines the default device number to be used for all disk-oriented commands. The primary purpose of this command is to facilitate multiple disk drive operation.

After loading some programs which extend down into the cassette buffer area, the disk may appear to become inoperative. This is because the program has over-written the disk device number. You can correct the problem by simply entering a SETD command (usually "SETD,8").

SPECIAL USES

Some commands will allow you to use the tape drive for some of their functions if you set the device number to 1 with the command:

SETD,1

This is applicable with the following commands:

@L"<name>"S (list tape data file)
@L"<name> (list program stored in tape data format)
GET
LOAD
PUT
SAVE
VERIFY

With some printers, it may be possible to use SAVE and PUT with the disk output sent to the printer. Output is sent to secondary address #1 so this will not work with CBM printers.

COMMAND

SETP

FUNCTION

SET PRINTER DEVICE, FORMAT MODE, AUTO PAGING, ASCII CONVERSION.

SYNTAX

SETP, <device #> [+128] [+64]

DESCRIPTION

The SETP command defines the default device number to be used for the "*" command. The primary functions of the SETP command are to facilitate multiple printer operation and to select the various printer options.

After loading some programs which extend down into the cassette buffer area, the "*" command may appear to become inoperative. This is because the program has over-written the printer device number. You can correct the problem by simply entering a SETP command (usually "SETP,4").

AUTO PAGING

The auto-paging mode will cause the printer to automatically skip the page perforations on continuous-feed paper. When a printout is complete, the paper will be advanced to the top of the next page. While the auto-paging mode is active, a "*" command alone on a line will feed one page. To activate the auto-paging mode, enter the command:

SETP,<device 1>+128

FORMATTED LISTINGS

The formatting mode greatly increases the readability of BASIC program listings. When the formatting mode is active, program listings made with the "*LIST" and "*@L" commands are printed to the following specifications:

LINE NUMBERS ARE RIGHT-JUSTIFIED
ONE COMMAND PER LINE
SPACES BETWEEN COMMANDS
FOR-NEXT LOOPS INDENTED BY LEVEL OF NESTING

Printouts on an ASCII printer have the additional features:

COMMANDS PRINTED IN CAPITALS
TEXT PRINTED IN LOWERCASE MODE

Formatted listings may be sent to the SCREEN by using a printer device number of 3. To set the formatting mode on, use the command:

SETP,<device #>+64

The formatting and auto-paging modes may be selected at the same time with the command:

SETP, <device #>+128+64
or
SETP, <device #>+192

USING A NON-Commodore™ PRINTER

SYSRES™ allows you to use a non-Commodore-64™ (ASCII) printer to print program listings, or any other outputs produced by SYSRES™ commands. Normally, SYSRES™ will recognize device #5 as an ASCII printer and will overstrike symbols to represent special characters in the Commodore-64™ character set. It will also overstrike the number zero with a slash to differentiate it from the letter "O". When text is within quotes, it is printed in lower case, with shifted letters in capitals. Shifted letters not represented in ASCII (such as shifted-number graphics) are printed as the unshifted character, underscored. Control characters are over-struck with a circumflex (or a "¶" on QUME WP sequence printers). Outside of quotes, all text Is printed as capitals; with shifted letters underlined. Back-arrows and up-arrows are also generated using overstrikes. This full conversion mode works best with letter-quality printers such as NEC SPINWRITER, C.ITOH STARWRITER, TEC, QUME, and DIABLO, or any printer which accepts the back-space character (CHR$(8)).

SYSRES™ also handles paging for non-Commodore-64™ printers, printing 60 lines per page, then skipping six lines for the page perforations (6 lines per inch on an 11 inch page is 66 lines per page).

CHANGING THE PRINTER SPECIFICATIONS

The SYSRES™ printer handling routines are very flexible and are designed to accommodate a wide variety of hard-copy devices. Once you have determined the correct set-up instructions for your system, you could put them into an EXEC file (along with your key definitions etc.) to auto-configure SYSRES™ for your system.

The **ASCII printer device number and conversion modes** are stored in a flag at memory location 40970 (hex $03E4). You can change the ASCII conversion specifications by "POKEing" the appropriate value into that location. To calculate the value to use, answer the following questions, then add up the values given for your responses:

	QUESTION	YES	NO
1.	Does the printer require ASCII?	0	16
2.	Does the printer allow overstrikes?	0	64
3.	Do you want a slash through zero?	32	0
4.	Does the printer require linefeeds?	128	0
5.	What is the printer device # (4 -15)?	(use actual value)	

Take the SUM of your answers (N) and use the command POKE 40970,N. Notice that you may set-up the printer paging and linefeed options even if the answer to question #1 is "no".

Most printers use the standard line-spacing of six lines per inch. If your printer uses a non-standard spacing, or if your paper is not 11 inches per page, you may change the number of lines per page. To set the number of printed lines on a page, use the commands:

POKE 40971, <number of printed lines>
POKE 40972, <number of skipped lines>

When SYSRES™ is Initialized, location 40971 contains 60 and location 40972 contains 6.

EXAMPLES

SETP,67

This will cause formatted listings to go to the screen (3+64=67). You may enter the number in this form or in equation form ("SETP,3+64"), whichever you find more convenient.

SETP,5+128+64
or
SETP,197

Either of these examples would set device 5 to be the output device (normally ASCII). This command turns both the FORMATTING and AUTO-PAGING features on.

COMMAND

TRACE

FUNCTION

SELECT TRACE/STEP MODE AND SPEED.

SYNTAX

TRACE
TRACE [P] [V] [<speed>]

With the options:

P Trace Program execution
V Trace Variable assignments

<speed> may be:

1-255 1 is fastest, 255 is slowest
0 Hold-off trace until flag is set

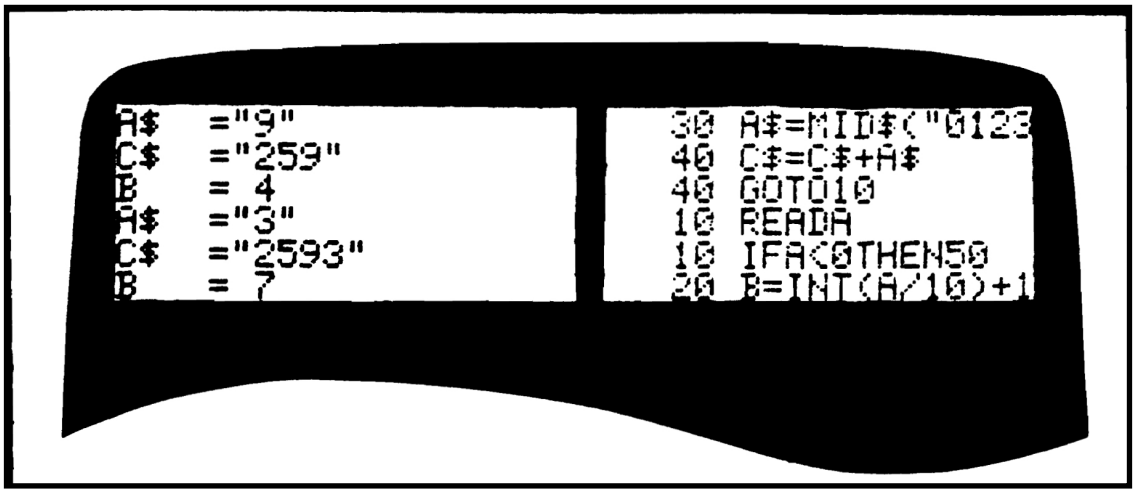
DESCRIPTION

The TRACE command sets a "trace-hook" which will display the progress of a BASIC program as it is running. There are three types of TRACE; they are "TRACEP", "TRACEV", and "TRACEPV" ("TRACEPV" is simply "TRACEP" and "TRACEV" at the same time). To trace program execution and variable assignments at a readable speed, an appropriate command would be:

```
TRACEPV 100
```

When the program is running, the "trace-hooks" each set-up a reverse field window at the top of the screen, displaying the current operation and the previous five operations in their windows. The window for "TRACEV" is at the top left corner of the screen, and the window for "TRACEP" is at the top right corner. When both trace windows are active, the display would look something like the example shown in figure 1 on the following page.

Figure 1. TRACEPV Screen Display



At any time, the output may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL], or stepped by holding down the SPACE BAR and tapping [LOGO] or ended by pressing [RUN/STOP].

At any time, the display (and the program) may be paused with the SPACE BAR, continued with [LOGO], slowed by pressing [CTRL), or stepped by holding down the SPACE BAR and tapping [LOGO], or stopped completely by pressing the [RUN/STOP] key. Pressing the [SHIFT] key will cause the program to executed at the fastest possible speed (this allows you to use the [SHIFT LOCK] key).

TRACEV

The "TRACEV" mode will display the new value of a variable each time it is assigned a new value with a "LET" command or in an equation (eg., A=3*X). It will also display the STARTING VALUE of a variable In a FOR-NEXT loop. It DOES NOT display values set by other commands such as GET, READ, INPUT, and NEXT unless you tell it to. For example, to track the variable received in an input statement, you would re-assign the variable such as:

```
10 INPUT#1,A$:A$=A$
```

The same technique applies to FOR-NEXT loops as in the line:

```
105 FOR I= TO E:I=I
```

This technique helps you to display variables only when they have meaningful values. It allows you to ignore things like delay loops (FOR I=1 TO 5000:NEXT) and "GET" loops (when the value is thrown away).

Due to limitations in the window size, the indices into array variables are not displayed (you could use the technique shown above). If a string variable is too long to fit into the window, it will be truncated. This is indicated by the fact that there will be no closing quote.

TRACEP

The "TRACEP" mode will display the line number and the command currently being executed. The line is displayed up to the next colon (or end of line). If necessary, the line may be truncated to fit in the window. Control codes within quotes (home, cursor-up, etc.) will not be differentiated from their alphabetic counterparts (a side-effect of the high-speed window).

You may occasionally see a line which looks something such as:

```
1020 1090
```

At first glance this would seem like an illegal statement. A number by itself is actually a valid command, but only after a "THEN" statement. The actual program line would have been a statement similar to:

```
1020 IF A =0 THEN 1090
```

TRACE HOLD-OFF

If you select a trace speed of zero (in any trace mode), the appropriate "trace-hooks" will be set, but tracing will not begin when the program is run. To enable tracing from within the program, use the command:

POKE 118,<speed>

Tracing will continue until the end of the program, or until hold-off is re.-activated with:

POKE 118,0

This process may be repeated as often as desired.

DISABLING TRACE

To turn the TRACE functions off, enter the command "TRACE" alone (without a line number).

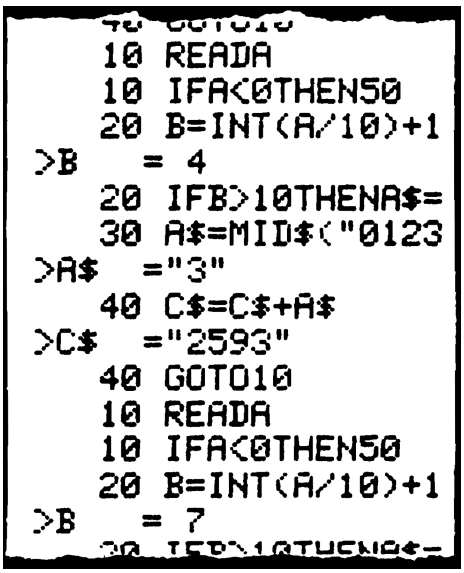
TRACE TO THE PRINTER

Trace output may be sent to the printer by running the program with the command:

∗RUN

When trace output Is directed to the printer, the normal trace windows will appear on the screen and a special output will be generated for the printer. Figure 2 shows the format of the printer output as seen on a Commodore-64™ printer.

Figure 2. TRACEPV Printer Output



If "TRACEV" is active, the **variable** trace lines are flagged with a ">" to make them easier to identify. The output is identical to the display in the trace windows.

NOTE: **The "CMD" command will not work when tracing a program to the printer.**

COMMAND

VERIFY

FUNCTION

COMPARE THE PROGRAM ON DISK (OR TAPE) WITH THE PROGRAM IN MEMORY.

SYNTAX

VERIFY
VERIFY ["<filename>"] [,<device #>]
VERIFY <directory>

DESCRIPTION

The VERIFY command, like all file commands under SYSRES™, may be used from the directory by typing the word "VERIFY" in the left margin beside the name of the desired program, then pressing [RETURN]. If the file is not a program ("PRG") then a "?SYNTAX ERROR" will be returned.

If the program on disk matches the program in memory, the message:

VERIFY OK: filename

will appear on the directory line. If the program on disk is different from the program in memory, the message:

VERIFY ERROR: filename

will appear (in reverse video) on the directory line.

When a filename is specified, VERIFY will default to the disk drive if no device number is specified. If just the word "VERIFY" alone is entered, the next program will be verified from the TAPE drive.

If you want to verify a program, with a specified name, from the tape drive, you must use the form:

VERIFY [<filename>],1

or you could force the normal BASIC interpretation of the VERIFY command by preceding the command with a colon, as in:

:VERIFY "<filename>

This will verify the specified program from the tape.

UNEXPECTED VERIFY ERRORS

You may occasionally get a "VERIFY ERROR" when you know that a program is saved on the disk. This should never happen if you always save programs with SYSRES™ since it is caused

by a minor bug in the disk drive (which SYSRES™ corrects) which may put extra characters onto the end of your program when it is saved using a "write-replace" ("@<drive>:<filename>"). To check if this is the case, use "@L" to list the program. If the message "DISK: 00, OK ,00,00" is printed at the end of the listing, it is an indication that the "write-replace" problem has occurred.

EXAMPLES

VERIFY"MYPROG

This will compare the program "MYPROG" on the disk with the program currently in memory.

VERIFY"REWRITER 3.0" PRG

This is an example of using the VERIFY command from the directory. The program "REWRITER 3.0" will be compared with the program in memory.

COMMAND

WHY
WHY?

FUNCTION

LIST LINE AND POSITION OF LAST ERROR.

SYNTAX

WHY
WHY?

DESCRIPTION

The "WHY" and "WHY?" commands will help to find the reason why the program stopped executing by displaying the line which the computer was executing when it stopped. Although the responses to "WHY" and "WHY?" are usually almost identical, they are arrived at by different means and actually give you different information. Both commands will continue to remember where the error was, even if you change or re-load the program!

The "WHY" command displays the **location that the computer was looking at** when the program stopped. In the case of **DEFined FuNctions**, the "WHY" command will usually display **the line which the FUNCTION is on**. If an error occurs in a **DATA statement** the "WHY" command will display **the line of DATA being READ**.

The "WHY" command will reverse the remainder of the line, starting at the location the computer last looked at before the error was discovered.

The command "WHY?" displays the **program line which the computer was executing** when operation was disrupted. This is the line given in the error message as in:

```
?ILLEGAL QUANTITY ERROR IN 20
```

EXAMPLES

To demonstrate the difference between the "WHY" and "WHY?" commands, enter the following program. User input is **BOLDFACE**. UNDERLINED text would be displayed in reverse-video on the screen.

```
10 DEF FNA(B)=A(B)+1000
20 PRINT FNA(-1)
RUN

?ILLEGAL QUANTITY ERROR IN 20
READY.
WHY

10 DEF FNA(B)=A(B)+1000
```

```
READY.
WHY?

20 PRINT FNA(-1)
```

COMMAND

*

FUNCTION

SEND OUTPUT TO THE PRINTER.

SYNTAX

*
*<any command>
*CMD224, <print expression>;

DESCRIPTION

Preceding any BASIC, SYSRES™, or machine-language-monitor command with as asterisk ("*") will send the output from that command (which would normally go to the screen) to the current printer device. Using the "SETP" command, the printer may be set to any device which does not require secondary addressing or a filename. This may include such devices as modems, ASCII printers, or even the screen (device #3).

The "*" command alone on a line will advance the printer by one line. A useful trick may be to enter the following "KEY" command (shown in lowercase mode):

key"[","<cursor-up>*Z"

When this function is entered, holding the shifted - "]" down will cause the printer to keep advancing until the key is released.

A special CMD command may be used with the "*" command, it is:

CMD224, <print expression>;

This will print the expression to the current printer device. You do not need to open file 224 as the "*" command does it for you. Although the output is not sent through the ASCII converter, the carriage-return/linefeed option will be handled (see "SETP").

After loading some programs which extend down into the cassette buffer area, the "*" command may appear to become inoperative. This is because the program has over-written the printer device number. You can correct the problem by simply entering a SETP command (usually "SETP,4").

The special command "*RUN" WILL NOT send all program output to the printer as might be expected. It is used to send the TRACE output to the printer (see the TRACE command). To send program output to the printer, use the "CMD" and "PRINT#" commands within your program.

AUTO PAGING

The auto-paging mode will cause the printer to automatically skip the page perforations on continuous-feed paper. When a printout is complete, the paper will be advanced to the top of

the next page. While the auto-paging mode is active, a "*" command alone on a line will feed one page. To activate the auto-paging mode, enter the command:

```
SETP, <device #>+128
```

FORMATTED LISTINGS

The formatting mode greatly increases the readability of BASIC program listings. When the formatting mode is active, program listings made with the a "*LIST" and "*@L" commands are printed to the following specifications:

LINE NUMBERS ARE RIGHT-JUSTIFIED
ONE COMMAND PER LINE
SPACES BETWEEN COMMANDS
FOR-NEXT LOOPS INDENTED BY LEVEL OF NESTING

Printouts on an ASCII printer have the additional features:

COMMANDS PRINTED IN CAPITALS
TEXT PRINTED IN LOWERCASE MODE

Formatted listings may be sent to the SCREEN by using a printer device number of 3. To set the formatting mode on, use the command:

```
SETP, <device #>+64
```

The formatting and auto-paging modes may be selected at the same time with the command:

```
SETP, <device #>+128+64
```
or
```
SETP, <device #>+192
```

SENDING OUTPUT TO A FILE

The "*" command can be used to send output to a file oriented device (such as the disk drive) or to a device requiring a special secondary address (like the printer formatting channel). In these cases use the special file number 224 in the example (may not work with the tape drive):

```
OPEN224,8,2,1,"1:FMLIST,S,W"
CMD224,"";
```

When the current CMD file is 224, you may use the "*" command to send output to that file. This only works once. The file Is automatically closed after a "*" command. Note: for formatting to take place, the actual printer device must be set to format mode (eg., SETP,4+64 has previously been done).

USING A NON-Commodore PRINTER

SYSRES™ allows you to use a non-Commodore (ASCII) printer to print program listings, or any other outputs produced by SYSRES™ commands. Normally, SYSRES™ will recognize device #5 as an ASCII printer and will overstrike symbols to represent special characters in the

Commodore-64™ character set. It will also overstrike the number zero with a slash to differentiate it from the letter "0". When text is within quotes, it is printed in lower case, with shifted letters in capitals. Shifted letters not represented in ASCII (such as shifted-number graphics) are printed as the unshifted character, underscored. Control characters are over-struck with a circumflex (or a "©" on QUME WP sequence printers). Outside of quotes, all text is printed as capitals, with shifted letters underlined. Back-arrows and up-arrows are also generated using overstrikes. This full-conversion mode works best with letter-quality printers such as NEC SPINWRITER, C.ITOH STARWRITER, TEC, QUME, and DIABLO, or any printer which accepts the back-space character (chr$(8)).

SYSRES™ also handles paging for non-Commodore-64™ printers, printing 60 lines per page, then skipping six lines for the page perforations (6 lines per inch on an 11 inch page is 66 lines per page). See the "SETP" command for further information and for instructions on how to change these specifications.

NOTE: Output from BASIC commands DOES NOT GO THROUGH THE ASCII CONVERTER. Also, BASIC commands have a propensity for sending extra garbage to files (such as linefeeds and "READY." messages).

To use SYSRES™ with a Commodore VIC-1525 serial graphic printer, enter the command:

```
POKE40970,52
```

before attempting to set the auto-paging mode (SETP,4+128 or SETP,4+128+64 with formatting). The 1525 printer does not accept to C8M printer standard codes for paging, so this poke tells SYSRES™ that you have an ASCII printer with built-in character-set conversion. You may now use commands such as "*LIST" to list to the printer with auto paging.

If you wish to list a program in lowercase mode on a 1525 printer, enter:

```
OPEN224,4,7
CMD224,"";
*LIST
```

This method will also work with all other print-outs such as *@L, *DUMP, FIND, etc. If you use this feature quite often, you might want to define a key such as:

```
key"l","open224,4,7Zcmd224,QQ;Z*listZ
```

COMMAND

# #

FUNCTION

DISPLAYS THE CURRENT VERSION OF SYSRES™.

SYNTAX

#

DESCRIPTION

The "#" command displays an identifying message in the format:

SYSRES™ 64 Vxxxxx (C)1987 Hands-on Software Inc.

The "#" command is a good way to re-connect SYSRES™ after running a program. It ignores any garbage on the line and tells that SYSRES™ is active. It produces a totally harmless error message if SYSRES™ is not in the computer.

XMON-64 INSTRUCTIONS

XMON is a machine-language monitor for the Commodore-64™. It has been converted to the Commodore-64™ with SYSRES™ in mind, so it is 100% compatible. See "MON" for loading instructions.

SIMPLE ASSEMBLER

```
.A 2000 A9 12         LDA #$12
.A 2002 9D 00 80      STA $8000,X
.A 2005 DEX:GARBAGE
```

In the above example the user started assembly at $2000. The first instruction was **L**oa**D** **A**ccumulator immediate with $12. In the second line the user did not need to type the A and address. The simple assembler retypes the last entered line and prompts with the next address. To exit the assembler press [RETURN] after the the address prompt. Syntax is the same as the disassembler output. A ":" can be use to terminate a line.

COMPARE MEMORY

```
.C 1000 2000 C000
```

Compares memory from $1000 to $2000 to memory beginning at $C000. Compare will print the locations of the unequal bytes.

DISASSEMBLER

```
.D 2000
```

```
., 2000 A912          LDA #$12
., 2002 9D 00 80      STA $8000,X
., 2005 AA            TAX
```

Disassembles to the end of memory starting at $2000. The three bytes following the address may be modified. Use the [CRSR] keys to move to and modify the bytes. Hit [RETURN] and the bytes in memory will be changed. XMON will then disassemble that line again.

```
.D 2000 3000
```

Disassembles from 2000 to 3000.

FILL MEMORY

```
.F 1000 1100 FF
```

Fills the memory from $1000 to $1100 with the byte $FF.

GO RUN

```
.G
```

Go to the address in the PC register display and execute code. All the registers will be replaced with the displayed values.

.G 1000

Go to address $1000 and begin running code.

HUNT MEMORY

.H C000 D000 'READ

Hunt through memory from $C000 to $D000 for the ASCII string "READ" and print the address where it is found. A maximum of 32 characters may be used.

.H C000 D000 20 D2 FF

Hunt memory from $C000 to $D000 for the sequence of bytes $20, $02, $FF and print any addresses where it is found. A maximum of 32 bytes may be used. Hunt can be stopped with the [RUN/STOP] key.

INTEGRATE MEMORY

.I F000

```
.I F000 54 4F 4F 20 40 41 4E 59 TOO MANY
.I F008 20 46 49 4C 45 D3 46 49  FILESFI
```

Displays hex and ASCII until the end of memory.

.I F000 F080

Displays hex and ASCII from $F000 to $F080.

LOAD FROM TAPE

.L

Load any program from cassette #1. Due to a bug in the ROM routines for the "L" command, this command may not always work from disk. Use the SYSRES™ "BLOAD" command (from BASIC) with the disk.

.L "RAM TEST"

Load from cassette #1 the program named "RAM TEST ".

Beware: Load from tape with a file name breaks the IRQ saved by the monitor. Do not use the "G" (go) command after a load or save. Exit to BASIC and re-enter monitor.

MEMORY DISPLAY

.M 0000 0008

```
.: 0000 00 01 A0 B3 4D 5C F6 E7
.: 0008 28 49 8A 4B 5C 7D 0E 3F
```

Display memory from $0000 to $0008. The bytes following the address may be modified by editing and then pressing [RETURN].

NEW LOCATOR

.N 7000 77FF 6000 0400 9000

.N 77CD 77FF 6000 0400 9000 W

The first line fixes all three-byte instructions in the range $7000 to $77FF by adding $6000 offset to the bytes following the instruction. New locator will not adjust any instruction outside of the $0400 to $C000 range. The second line adjusts .word values in the same range as the first line. New locator stops and disassembles on any bad op-code.

REGISTER DISPLAY

.R

```
   PC   IRQ  P# SR AC XR YR SP
.; 0000 EA31 07 51 DE 03 04 05
```

Displays the register values saved when XMON was entered. The values may be changed with the edit followed by [RETURN].

SAVE TO TAPE OR DISK

.S "PROGRAM NAME",01,0800,0C80

Save to cassette #1 memory from $0800 up to but not including $0C80 and name it "PROGRAM NAME".

.S "0:PROGRAM NAME",08,0800,0C80

Exactly the same as above except this saves to the disk (device #8).

Beware: Save to tape with a file name breaks the IRQ saved by the monitor. Do not use the "G" (go) command after a load or save. Exit to BASIC and re-enter monitor.

TRANSFER MEMORY

.T 1000 1100 5000

Transfer memory in the range $1000 to $1100 and start storing it at address $5000.

WALK CODE

.W

Single step starting at address in register PC.

.W 1000

Single step starting at address $1000. Walk will cause a single step to execute and will disassemble the next instruction. Step speed may be controlled with [<] for SINGLE STEP, [RVS] for SLOW, and [SPACE] for FAST.

EXIT TO BASIC

.X

Return to BASIC ready mode. The stack value saved when entered will be restored. Care should be taken that this value is the same as when the monitor was entered. A CLR or anything that would cause a SYNTAX ERROR in BASIC will fix any stack problems.

XMON-64 INSTRUCTIONS:

A	SIMPLE ASSEMBLER
C	COMPARE MEMORY
D	DISASSEMBLER
F	FILL MEMORY
G	GO RUN
H	HUNT MEMORY
I	INTEGRATE MEMORY
L	LOAD FROM TAPE
M	MEMORY DISPLAY
N	NEW LOCATOR
R	REGISTER DISPLAY
S	SAVE TO TAPE
T	TRANSFER MEMORY
W	WALK CODE
X	EXIT TO BASIC