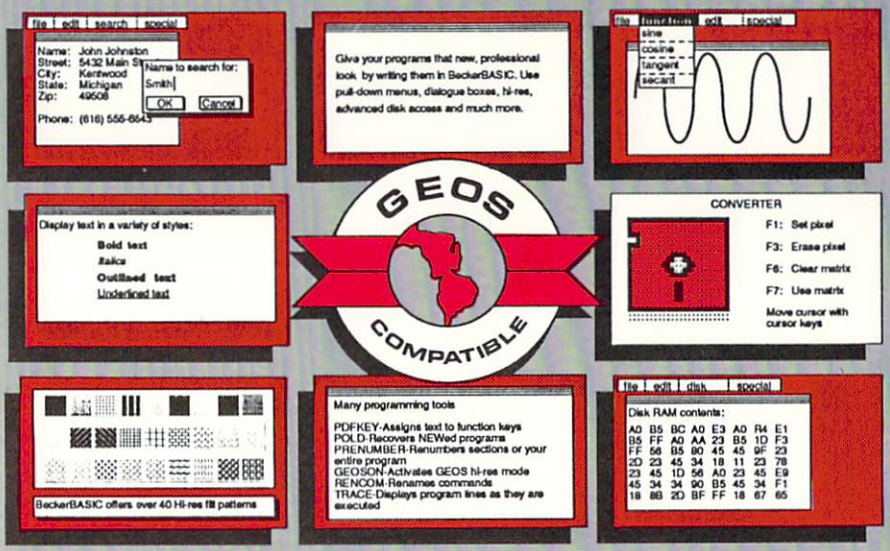
Becker BASIC

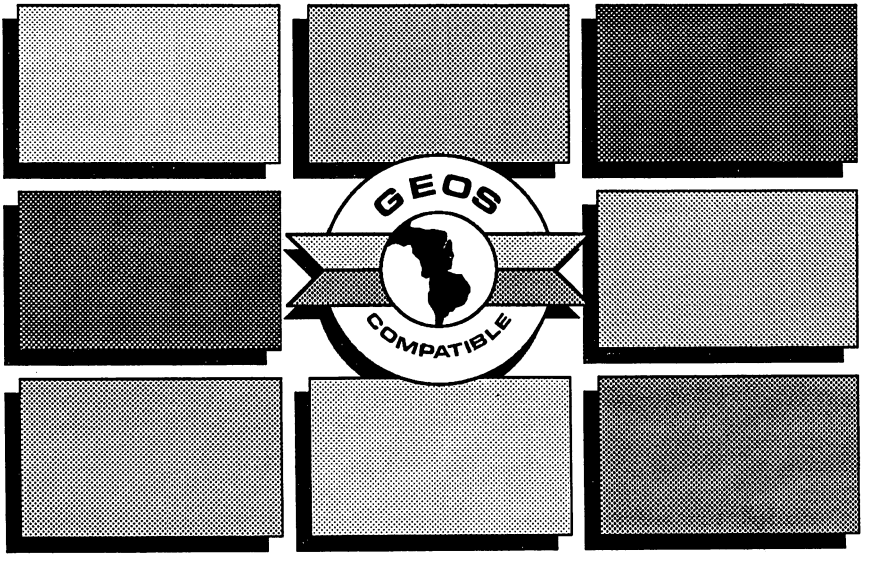
For programming applications under GEOS



Abacus
A Data Becker Product

Becker BASIC

For programming applications under GEOS



A Data Becker Product

Abacus

Copyright Notice

Warranty

Abacus Software makes no warranties, expressed or implied as to the fitness of this software package for any particular purpose. In no event will Abacus Software be liable for consequential damages. Abacus Software will replace any copy of this software which is unreadable if returned within 30 days of purchase. Thereafter, there will be a nominal charge for replacement.

First Printing February 1988
Printed in U.S.A.

Data Becker GmbH
MerowingerStr. 30
4000 Dusseldorf, W. Germany

Abacus, Inc.
5370 52nd Street
Grand Rapids, MI 49508

ISBN 1-55755-033-6

Table of Contents

Foreword

BeckerBASIC is an extension to BASIC 2.0 which is fully compatible with the GEOS user interface. BeckerBASIC includes programming tools, error handling commands, hi-res graphics, sound and GEOS support.

Unlike some language extensions, BeckerBASIC supports all aspects of programming. From programming tools and error handling to graphics and sound, BeckerBASIC is just what you need for efficient programming.

BeckerBASIC can be summed up in two words: Flexible and practical! BeckerBASIC even allows you to change the command and function names.

BeckerBASIC is one of the most complicated products ever released for the C64. The testing process at Data Becker and Abacus was an exhaustive one. However, it's almost impossible to test any product on every piece of hardware or software (e.g., disk drive enhancements, operating system extensions, countless application programs, etc.) available for the C64. This means that neither the author nor the publishers can be held responsible for support of programming or application problems (aside from errors within BeckerBASIC itself, of course).

To give you the complete picture of the program, BeckerBASIC comes with this thorough, easy to follow manual. I hope that this manual will serve you well.

Best of luck in your work with BeckerBASIC.

Martin Hecht
Stuttgart, West Germany
September 20, 1987

1. Introduction

BeckerBASIC consists of three interpreters, contains over 270 new commands, and runs under GEOS. BeckerBASIC is much more than a normal BASIC extension, however.

1.1 BeckerBASIC structure

You should learn the many commands and functions before you can program efficiently in BeckerBASIC. That is where this manual comes in. It's not absolutely necessary that you read the entire book to learn BeckerBASIC.

BeckerBASIC contains a total of 273 commands, and is made up of three interpreter systems: The Input-, the Testing- and the Run-Only-Systems.

If you look at the first directory page of your BeckerBASIC distribution diskette from the GEOS deskTop, you'll see three files named System 1, System 2 and System 3. These three programs are the three BeckerBASIC interpreters in VLIR format.

The three interpreters can be accessed directly from the deskTop by double-clicking the desired icon. You can toggle between the Input- and the Testing-Systems while a BeckerBASIC program resides in memory without losing the program.

Application programs written in BeckerBASIC can also be accessed from the deskTop by double-clicking their icons. The BeckerBASIC Run-Only-System loads, then the program loads and executes.

The CONVERTER program on the BeckerBASIC distribution diskette lets you define an icon for your BeckerBASIC programs. Normally, this routine assigns its own BASIC icons.

All three systems can be accessed at any time from the GEOS deskTop. They are loaded in from the diskette, then GEOS keeps them in memory.

Total loading time is between 10 and 15 seconds. Toggling between the Input- and Testing-Systems takes nine seconds to load and initialize.

All three systems give you 15,800 bytes of free memory. If you stop to think that GEOS and BeckerBASIC are in memory at the same time, this is a good amount of memory. There are ways around memory limitations: BeckerBASIC has overlay capabilities (loading multiple programs from diskette), and if you avoid high-res graphics, the unused bitmap starting at location 40960 gives you about 8K of additional memory.

The Input-System: The Input-System works much like the BASIC 2.0 editor—you type in and edit programs in this interpreter. Not all of BeckerBASIC's commands can be used here. Most of the available commands are programmer's tools.

All illegal commands produce an ILLEGAL COMMAND ERROR. The Testing-System is used for trying out your BeckerBASIC programs. You toggle to the Testing-System by pressing the key combination <CTRL><Commodore>.

The Testing-System: This interpreter lets you test run BeckerBASIC programs. The Testing-System uses all BeckerBASIC commands, as well as the editing and programmer's utility commands.

After starting the system, a normal text screen appears with a menu for controlling the entire operation. This menu is controlled by function keys.

Pressing the <F1> function key starts a BeckerBASIC program already in memory. After the program executes, the Testing-System returns you to the menu.

When an error occurs within the program, BeckerBASIC displays the prompt, "Error in Program! Load Input Interpreter (y/n)?" When you press the <y> key, BeckerBASIC loads the Input-System and displays the incorrect line number. If ERRSHOWON is in effect, BeckerBASIC also displays the error in reverse video. If you press the <n> key, the main menu of the Testing-System reappears.

Pressing <F3> asks for the name of a program you want loaded from diskette and run. Pressing <F7> or <CTRL><Commodore> returns you to the Input-System.

BeckerBASIC programs remain in memory when you switch from the Input-System to the Testing-System and back. <F8> exits BeckerBASIC and returns you to the deskTop.

The Run-Only-System: This interpreter allows royalty-free distribution of BeckerBASIC program code without distributing BeckerBASIC itself. There are no utilities or programming tools in this interpreter (no Input-System, no TRACE function, etc.).

When an error occurs, the message "Error in Program! Contact this program's author" appears on the screen, and BeckerBASIC displays a menu similar to that displayed from the Testing-System, except you cannot access the Input or Testing-Systems from the Run-Only system.

CONVERTER program: This program converts BeckerBASIC programs to GEOS format, for direct access by the Run-Only interpreter and the GEOS deskTop. By supplying a command at the end of a BeckerBASIC program to return you to the GEOS deskTop, the program runs as if it's an independent GEOS application.

The drop-down menus and dialogue boxes require the second hi-res bitmap starting at memory location 24576. This reduces the amount of available BASIC memory by eight kilobytes.

1.1.1 Starting BeckerBASIC

You start BeckerBASIC as you would start any GEOS application. Before you start BeckerBASIC, however, make one or more working copies of the BeckerBASIC distribution diskette. Use the BACKUP or DISKCOPY program from the GEOS system diskette to make backup BeckerBASIC diskettes. The procedure is exactly the same as making a backup of other GEOS applications (see your GEOS manual for information). When you've finished making backup copies, put the original diskette in a safe place.

Copy the GEOS deskTop to your backup diskettes. You could also copy over desk accessories such as the Notepad, but these accessories are inaccessible from BeckerBASIC.

You can now begin your tour through BeckerBASIC. All program sections mentioned in this chapter are described in detail later on in the book. If you don't understand what you read here, the later descriptions should clear things up.

You'll find all BeckerBASIC system files on the first page of the directory on the GEOS deskTop. Open the BeckerBASIC work diskette by selecting the `open` item from the `disk` menu. The upper row shows icons named System 1, System 2 and System 3. These are BeckerBASIC's three interpreters. The first icon is the Input-System, the second icon represents the Testing-System and the third is the Run-Only system. All these interpreters start when you double-click the icons.

Look for the BASIC icon named DEMO. This demonstration program is a BeckerBASIC program, handled as a GEOS application. DEMO displays just a few of BeckerBASIC's abilities. Start it by double-clicking the icon from the deskTop.

The CONVERTER program converts your BeckerBASIC application to GEOS format.

Most of the BeckerBASIC programs on the system diskette have a BASIC icon. This icon is generated by the CONVERTER. If you want, you can make your own icon using the CONVERTER.

The last three programs on the first page of the directory are discussed later in this book. Chapter 9 describes SPRITE-EDIT, while Chapter 7 tells about DDM.C.S (the Drop-Down Menu Construction Set) and D.C.S (the Dialogue Box Construction Set). The second directory page lists different BeckerBASIC utilities and sample programs.

Appendix E takes you through the steps in creating a BeckerBASIC program. The final program is called ADDRSAMPLE on your BeckerBASIC disk. The program makes use of drop-down menus, dialogue boxes and many of the added commands BeckerBASIC gives.

1.1.2 The Input-System and Testing-System

The first two BeckerBASIC interpreters, the Input-System and Testing-System, work together. You enter and edit your BeckerBASIC programs in the Input-System, and test the programs using the Testing-System.

BeckerBASIC was broken up into three separate interpreters to save memory on the C64. The available memory is already low because of GEOS residing in memory. If the entire BeckerBASIC system was put into memory, there wouldn't be any room left for program development.

BeckerBASIC gives you almost 16,000 bytes of free BASIC memory. This is more than most non-GEOS BASIC extensions offer. When you consider that you get both GEOS and BeckerBASIC in memory, 16000 bytes is plenty of memory.

Each system has a limited number of the 273 BeckerBASIC commands available.

The Input-System is the first interpreter that needs close examination. Double-click the System 1 icon on the GEOS deskTop. The Input-System loads into the computer.

At the end of the loading procedure, some graphic garbage prints on part of the deskTop, the screen turns black, and the starting screen of the Input-System appears.

You'll find a number of programming utilities in the Input-System, like PDUMP and PRENUMBER (see Chapter 2 for more information). To save a BASIC program to diskette, simply type in DSAVEB"name". DLOADB"name" loads a BeckerBASIC file from diskette (see Section 5.3 for more information).

In most cases, you must enter the Testing-System to test programs, especially for GEOS hi-res and sound commands. You can test some programs from the Input-System mode. Start the program by typing RUN and pressing the <RETURN> key. To load and start a program with one command, type DRLOADB"name" (see Section 5.3).

NOTE: You cannot run programs using GEOS hi-res commands from within the Input-System.

You can start the Input-System directly from Commodore BASIC 2.0 by typing LOAD"DBL",8,1 and pressing the <RETURN> key. This gives you about 24,000 bytes of BASIC memory for development, as well as 20,000 bytes of memory unoccupied by GEOS.

The Input-System works the same whether you run it with or without GEOS, with one exception: If you run the Input-System from BASIC 2.0, you can't access the Testing-System or the deskTop.

You can see a complete list of Input-System commands in Appendix A. All commands and functions marked with an asterisk (*) or number sign (#) can be used in the Input-System.

If you use commands the interpreter doesn't understand, the computer stops and displays the ILLEGAL COMMAND ERROR message. When this happens, you must switch to the Testing-System to test the program.

You can access the Testing-System from the Input-System in two ways. First, you can save the program to diskette and type in DESKTOP <RETURN> to return to the GEOS deskTop. When the deskTop finishes loading, double-click the System 2 icon to load the Testing-System. The other method is to press the key combination <CTRL><Commodore>. Pressing these two keys loads the Testing-System in about nine seconds, while retaining the BeckerBASIC program you were working on in memory.

NOTE: Make sure that a BeckerBASIC work diskette containing both the Input- and Testing-System is in the drive when you make this switch, and not just a diskette on which you store your programs. Otherwise, the computer may crash, and destroy the program in memory.

Along with the current BASIC program, you also have the complete set of debugging tools (e.g., TRACE, ERRSHOWON and ONERRORGO—see Section 2.2), as well as all your variables.

A menu screen appears after the loading procedure which lists four options. The Testing-System has no options for editing BASIC programs.

To start your BASIC program, press the <F1> key to "Start program." Press the <F3> key to load a program from diskette and run it. You can end program name input by pressing the <RETURN> key. Pressing <SHIFT><RETURN> returns you to the menu screen.

The use of drop-down menus, dialogue boxes and hi-res graphics have been avoided in the operation of these interpreters for a number of reasons. First, they take up too much memory, and second, function keys are faster. However, if you really want to make your programming user-friendly, you can add GEOS commands to your own programs (see Chapters 7 and 8).

If BeckerBASIC programs have errors, the screen displays the message "Error in program!" and asks, "Load Input-System? (Y/N)". If you press the <n> key (no), the system returns to the menu screen of the Testing-System. If you press the <y> key (yes), the computer returns to the Input-System and displays the error messages.

If you have the extended error display on using the ERRSHOWON command (see Section 2.2), the incorrect line is listed, and the error appears in reverse video.

NOTE: As long as ONERRORGO is active (see Section 2.2), the error handling follows this route only in the Testing-System.

When you toggle back to the Input-System, you have all variables available. For example, you can now check the values of individual variables or print out current variable contents with the PDUMP command (see Section 2.1).

You could do a lot of switching back and forth between the Input- and the Testing-Systems when in the development phases of a program. This takes time, but it's something like working with a compiled language. For example, when you work with a Pascal compiler, you have to enter the text in an editor, load the compiler and try compiling the program. If the compilation fails, you have to return to editing mode, fix the program and start over. Since BeckerBASIC's load times are so brief, this waiting time isn't a problem.

The BeckerBASIC system will help you to learn structured programming: After about the 15th or 20th error message, you'll learn to be much more careful in your program development.

The last two menu options are self-explanatory. <F7> performs the same function as <CTRL><Commodore>, returning you to the Input-System.

You have a total of three methods of returning to the Input-System. You can press the <F7> key or the <CTRL><Commodore> key combination. The latter is useful for going to the Input-System, fixing the program, and returning to the Testing-System to retry the program. The third method returns you to the deskTop. Pressing the <F8> key has the same effect as typing DESKTOP in the Input-System (see Section 1.5). There are a few exceptions to available commands in the Testing-System. The Appendices list all BeckerBASIC commands. Those commands unavailable to the Testing-System and the Run-Only-System are marked with a number sign, and result in an ILLEGAL COMMAND ERROR.

1.1.3 The Run-Only-System

The entire program development and testing phases are performed in the Input and Testing-Systems. The Run-Only-System is of interest when you want to distribute your own BeckerBASIC programs as GEOS applications. The CONVERTER routine adds icons and info data to BeckerBASIC programs. Converted programs access the Run-Only-System when you double-click the icons from the deskTop.

The Run-Only-System contains almost all the same coding as the Testing-System. The big difference between the two is in error handling. The only error you get in the Run-Only-System is the message, "Error in program! Contact this program's author!" The error then sends you to the menu screen.

The <F1> key starts a BASIC program already in memory. <F3> automatically loads and runs the program name you request. <F8> returns you to the deskTop.

You cannot call the Input-System from the Run-Only-System. However, you can set up an error trap with ONERRORGO (see Chapter 2) for eventually catching errors. You can put a message in listing your address ("Error in program: Please write me at the following address—...").

When the Run-Only-System finds ERRSHOWON, ERRSHOWOFF,TRON and TROFF commands in a program, it returns either ILLEGAL COMMAND ERROR or the "Error in Program" message.

Distribute the Run-Only-System ONLY when you want to distribute your BeckerBASIC programs to other GEOS users. Distributing copies of the BeckerBASIC system itself is illegal. The only other ground rule: The Run-Only-System must be unchanged (leave the Run-Only-System named System 3).

The Run-Only-System should only be copied from the system diskette. Use the GEOS deskTop to do this. If you can't remember how to copy files, check your GEOS manual or ***GEOS Inside and Out*** from Abacus for instructions.

1.1.4 The CONVERTER

The CONVERTER program is an application written in BeckerBASIC used to convert your BeckerBASIC programs to GEOS format. Double-click the CONVERTER icon to start the program.

The CONVERTER program serves two purposes: It converts a BeckerBASIC program so that you can open it by double-clicking on its icon; and it also contains an icon editor for creating your own BASIC program icons.

NOTE: When you wish to edit a program already converted with the CONVERTER, you must run it through the CONVERTER program after editing in the Input-System. Also, CONVERTER should be used to convert a completely tested and debugged program only.

Here's what the CONVERTER does:

First, the routine asks for the name of the file to be converted and its filetype. The CONVERTER can handle both programs and data files (**never** try to start data files direct from the deskTop). Be sure that the diskette containing the program you want converted is in the drive.

If the program has not been converted, the CONVERTER mentions this. The CONVERTER then asks whether you want this file converted to a BeckerBASIC program or a data file.

The CONVERTER then asks for the data you want placed in the Info screen. You can select the default values by pressing <RETURN> for each entry (if the program was converted before), or enter new values. NOTE: The year input must be two digits (e.g., "88").

The CONVERTER asks "Use standard icon (y/n)?" If you respond with <y><RETURN> (yes), the program assigns the standard BASIC icon to the program, identifying the code as a BeckerBASIC program. Data files have BASIC DATA icons.

If you answer the prompt with <n><RETURN> (no), the CONVERTER branches to an icon editor, in which you can create your own program icons. The following functions are available in the icon editor:

<F1> sets a pixel (turns it on)

<F3> unsets a pixel (turns it off)

<F6> clears the icon matrix

<F7> transmits the completed icon

Before saving the data to diskette, a confirmation prompt appears: "Save data (y/n)?" If you respond with <n><RETURN> (no), the data clears and the CONVERTER restarts. If you wish to convert several programs, answer the next prompt ("Another program?") <y><RETURN> to restart the CONVERTER.

NOTE: You cannot use commas or semicolons when entering your info text. However, the info text can be edited later from the Info screen on the deskTop.

Converted BeckerBASIC programs run when you double-click their icons from the deskTop. The Run-Only-System must be on the same diskette as the converted BeckerBASIC program.

NOTE: You can replace the END statement at the close of a program with the DESKTOP command. The program then automatically returns to the GEOS deskTop, making it look as if it's a real GEOS application. (BeckerBASIC programs only *look* like GEOS applications; they don't really run the same as GEOS applications).

A good example of BeckerBASIC programming is the DEMO program on the BeckerBASIC distribution diskette.

The deskTop can be on the same diskette, but it doesn't have to be on the same diskette. If the deskTop is unavailable, GEOS displays a dialogue box asking for a diskette containing the deskTop.

Chapter 2 contains detailed information about the individual commands. Chapters 2, 3, 5 and 6 are the minimum reading you should do before you start working with BeckerBASIC.

One important note when renaming files: GEOS uses a different character coding from BASIC. The uppercase lettering and the numbers 0 to 9 are identical to BASIC character codes. However, the lowercase lettering is different. When you rename a BeckerBASIC program from the deskTop, use uppercase letters only, or else you may not be able to load the file from the Input-System. The best examples are the BASIC programs stored on the system diskette. When you display the GEOS deskTop directory, you'll see that all program names appear in uppercase lettering. However, if you read the directory (see DIR, Chapter 5), the BeckerBASIC names appear in lowercase lettering.

1.2 Changing command names

The option of renaming commands may seem unusual to you, but it's more than just a plaything. It allows you to program efficiently.

With over 250 commands, it's hard to find command names that suit every user. You can change the command names available to you from the Input-System.

Take the TRANSFER command, for example (Section 4.1). Since this is a frequently used command, maybe the command name would work better for you as the abbreviation TR. Or you could change the name to MEMSHIFT, or even MOVE.

You can rename commands to whatever you want. The format (parameter layout) and function stay as they are.

The new commands retain compatibility with other BeckerBASIC programs, since the commands are coded independently of the commands in memory.

You can distribute a program written in your implementation of BeckerBASIC to another user, and he can use your program with his Run-Only-System.

The entry and output of BASIC lines when editing takes a bit longer than BASIC 2.0. The large command set in BeckerBASIC causes this drop in speed.

1.2.1 Setting up the command table

BeckerBASIC uses two command tables. The first table contains the original command names; the second contains the user-defined new names.

OLDCOMTAB	(021)	(c)
NEWCOMTAB	(020)	(c)

OLDCOMTAB lets you switch to the original command name table, which is in effect when BeckerBASIC initializes. During program input, OLDCOMTAB compares all command names with those stored in the original table, and interprets the commands.

NEWCOMTAB switches to the newly defined command names, whether you've redefined a new name or not. BeckerBASIC automatically assigns the original command names to the new command table when NEWCOMTAB is called. You can assign new command names after calling NEWCOMTAB.

Format: NEWCOMTAB: ... :OLDCOMTAB

COMTAB (250) (f)

As mentioned above, you can toggle back and forth between the two command tables, either in direct mode or program mode.

COMTAB determines which command table is currently active.

Format: CT = COMTAB

The original table returns a value of 0 to CT; a new command table gives CT a value of 1.

1.2.2 Handling command names and numbers

PHELP (019) (c)

The PHELP command gives you a general overview of BeckerBASIC commands. PHELP displays all the commands on the screen, including their numbers. The display appears in a format of 2 columns, each set containing 20 commands. This takes up seven screen of text, since the numbers must also be visible.

Format: PHELP NO

NO is the number of the output page. Every page contains 40 commands. Page 1 (NO=1) shows commands 1-40, page 2 (NO=2) commands 41-80, etc. The seventh and last page (NO=7) lists commands 241-273. NO can be a number between 1 and 7. The command name output follows in the format COMMAND NUMBER:COMMAND NAME (e.g., 1:GOTO, 2:GOSUB, etc.).

NOTE: When you select the original command table with OLDCOMTAB, the displayed command names come from this table. However, when the NEWCOMTAB command is used, the names come from the new table.

COMNUM (231) (f)

You'll frequently want information about a specific command or function in BeckerBASIC.

All BeckerBASIC commands and functions are in numerical order (see PHELP). COMNUM gives the number of any command.

Format: A = COMNUM (BF$)

A contains the number of the command word listed in BF$. Any string can go into BF$.

COMNUM does essentially the same thing as PHELP: If the original command table is active, COMNUM compares BF$ with the stored name, then checks the new table. If BF$ doesn't match the old or new table, then A is assigned the value 0. A numerical expression for BF$ results in a TYPE MISMATCH ERROR.

Examples:

A = COMNUM("GOTO") makes A=1.

B$ = "COMNUM":B = COMNUM(B$) makes B=231.

C = COMNUM("XYZ") results in C=0, since the command "XYZ" doesn't exist (unless you've created your own command named XYZ).

For example, you need a description of the HPRINT command. SCPRINT COMNUM ("HRPRINT") gives a result of 214.

COMNUM is helpful, when used in conjunction with the table in Appendix A, in figuring out renamed commands.

COMNAME (251) (f)

COMNAME does the opposite of COMNUM: A number returns the command corresponding to the number.

Format: NM$ = COMNAME (BN)

BeckerBASIC assigns the command name for BN to the variable NM$. Like COMNUM, COMNAME accesses either table through OLDCOMTAB or NEWCOMTAB.

Examples:

SCPRINT COMNAME (32) returns TRACE.

SCPRINT COMNAME (149) returns SDVOLUME, a sound command.

G$ = MID$(COMNAME(243),3,5) returns: G$="CHECK".

TR$ = COMNAME (400) gives an ILLEGAL QUANTITY ERROR, since no command exists with the number 400.

1.2.3 Renaming commands

RENCOM (022) (c)

RENCOM allows you to rename any BeckerBASIC command, including RENCOM itself. There are two ways to do this:

1. RENCOM (BN) = (NN$)

Command number BN receives the new name listed in NN$.

2. RENCOM (ON$) = (NN$)

RENCOM replaces the command name listed in ON$ (OldName) with the new name contained in NN$. ON$ is immediately compared with the newly defined command name, which goes to the second command table.

There are some rules you must remember when assigning new command names:

The new name must have a minimum of two characters, and a maximum of 15 characters. Going beyond these results in a COMMAND NAME ERROR.

BeckerBASIC provides 3000 bytes for newly defined command names, which assumes an average name length of 10 characters. When the command table goes past this 3000 byte limit, the result is a COMMAND NAME ERROR.

You cannot use quotation marks ("), apostrophes ('), or Commodore ASCII codes higher than 127 in your names (see your C64 manual or ***Programmer's Reference Guide*** for ASCII code information). These characters result in a COMMAND NAME ERROR.

Using a question mark (?), colon (:), semicolon (;), comma (,) space ( ) or a number from 0 to 9 at the beginning of a command name also results in a COMMAND NAME ERROR.

A new command name should not contain part of another command name. For example, say you had two commands named GOTHERE and GOTHERETOO. When the interpreter encounters GOTHERETOO, it will treat the command as GOTHERE. That is, it executes GOTHERE, and interprets TOO as a parameter or another command. This interpretation only occurs if both the GOTHERE and GOTHERETOO commands are in the command table.

Here's how BeckerBASIC interprets commands. If it recognizes a string as a command name, then it compares all the names in the command table with the string. When it finds a command name in the table whose name matches the string in question, the command executes and the program continues.

If the comparison ends without finding a match, an error occurs. At best, the interpreter could treat the extension of a command name as a parameter, as in the GOTHERETOO example described above.

As already mentioned, RENCOM checks the old command name against the new command table. Here's a little trick which allows you to use the original command name:

OLDCOMTAB:RENCOM(COMNUM(ON$) = (NN$)) changes to the original command table. COMNUM searches for ON$ in the original table. The intended command number transfers through RENCOM, and the program continues (1st command variant).

Examples:

RENCOM ("RENCOM") = ("COMCHGE") changes RENCOM to COMCHGE. All you have to do is remember to use COMCHGE for renaming commands, instead of RENCOM (e.g., COMCHGE ("GOTO")=("GOTHERE")).

RENCOM ("LIST") = (";LINLIST") results in a COMMANDNAME ERROR, since the command LINLIST begins with a semicolon.

RENCOM ("ONERRORGO") = ("WHENOOPSGETLINE") assigns the command ONERRORGO the name WHENOOPSGETLINE. This command has the maximum of 15 characters.

```
5'DISPLAY OLD COMMANDS AND ASK FOR NEW NAMES'
10 FOR BN=1 TO 273
20 SCPRINT COMNAME (BN)
30 NN$=""
40 INPUT "NEW NAME:";NN$
50 IF NN$=""THEN NN$=COMNAME(BN):ENDIF
60 RENCOM (BN) = (NN$)
70 NEXT BN
```

This short routine displays each command name and asks for a new command name. If you don't want the name changed, press the <RETURN> key.

1.2.4 Saving and loading command tables

DSCOMTAB	(023)	(c)
DLCOMTAB	(024)	(c)

DSCOMTAB saves the new command name table to diskette. DLCOMTAB loads a table into memory which was saved using DSCOMTAB.

Format: DSCOMTAB NA$: ... :DLCOMTAB NA$

NA$ is the name under which the table is or was stored to diskette. This string can be a maximum of 16 characters in length (a longer name causes a STRING TOO LONG ERROR).

Example:

DSCOMTAB "NEWTAB" saves a new command table to diskette under the name NEWTAB. DLCOMTAB "NEWTAB" loads the table into memory.

TABNAME (209) (c)

Another problem exists when you toggle from the Testing- to the Input-System: Calling the Input-System loads the command name tables and the program code from diskette.

If you're working with new names, then the corresponding name table must be reloaded, so that the system recognizes the command names. BeckerBASIC uses the TABNAME command to convey the name of the table stored on diskette.

Format: TABNAME NM$

NM$ is the name under which the table is stored on diskette. The name can have a maximum length of 16 characters.

You can also use this command for loading a name table for a program restart. Just put the necessary commands into a short program and save this under the name TABINT on your work diskette:

```
10 DLCOMTAB "NAME":NEWCOMTAB:TABNAME "NAME":END
```

When you start up the Input-System, type the following in direct mode:

```
DRLOADB"TABINT"
```

The program loads and automatically starts, and initializes the command table NAME (see Section 5.3 for more information).

1.3 BASIC 2.0 commands

All normal BASIC 2.0 commands function in all three BeckerBASIC interpreters. Some of these commands were included in the BeckerBASIC system (note the command numbers in parentheses): GOTO (001), RUN (005), IF (110), THEN (111), RESTORE (003), ON (174), LIST (004) and NEW (177). You can only change the names of these commands. The other BASIC 2.0 commands cannot be renamed for technical reasons.

BASIC 2.0 programs run under BeckerBASIC after you convert them to BeckerBASIC. List the program lines on the screen under BeckerBASIC and press the <RETURN> key on each line so the line is accepted. This way, you can set up the new BeckerBASIC coding in BASIC memory.

1.4 Adding commands & functions

DB	(173)	(c)
DF	(244)	(f)

Machine language programmers may add commands and functions to BeckerBasic. **NOTE:** This section assumes that you have some knowledge of machine language. If not, please go on to Section 1.5.

When BeckerBASIC finds a DB or DF, the program branches to memory address 25500 or 25000, respectively. You can define new commands or functions in these memory locations. See Appendix G (page 228) for examples of DB and DF.

One note about new commands and functions: The command subroutine should end with RTS, as you would with any machine language program.

The value of the function should be placed into the floating point accumulator 1. Corresponding routines are available in the C64 operating system. For example, a 1-byte value normally found in the Y-register can be placed in the floating point accumulator by JSR $B3A2, or accessed in a routine with JMP $B3A2.

1.5 Miscellaneous

LIST (004) (c)

The BeckerBASIC LIST command is basically the same as the BASIC 2.0 LIST. The big difference between the original LIST command and the BeckerBASIC LIST is that the BeckerBASIC LIST can run within a program, without stopping program execution.

Here is an example of in-program use of LIST:

```
100 SCPRINT "LINE 200:":LIST 200
200 SCPRINT "NEXT LINE:"LIST 300
300 SCPRINT"END"
```

The LIST parameters are as follows:

LIST 10 -100 lists program lines from 10 to 100.

LIST 10 - lists the program starting at line 10 to the end.

LIST - 100 lists from the start of the program up to and including line 100.

LIST lists the entire program.

NOTE: If you rename the LIST command (e.g., to PROGLIST), and you've switched to the new command table, don't use the LIST command!! You'll get a system crash.

Use the new name as soon as you start working with the new command table. The old table always has LIST on it for your use.

PRLIST (170) (c)

PRLIST has the same purpose as LIST, except that PRLIST sends the output to a printer.

The printer must have a device address of 4.

Examples:

PRLIST 10 - 25 lists lines from 10 to 25.

PRLIST 15:PRLIST 20:PRLIST 100 prints lines 15, 20 and 100.

PAUSE (007) (c)

This command inserts a pause in a program, to keep messages on the screen for a period of time.

Format: PAUSE SC

The variable SC equals the number of seconds you want the program to wait. SC=1 delays for about one second. Values for SC range from 0 to 255.

SWAP (071) (c)

The SWAP command swaps variable contents, and lets you avoid creating a third variable.

Format: SWAP V1,V2: ... :SWAP V1$,V2$

The contents of variables V1 and V2 are exchanged with each other, as are the contents of variables V1$ and V2$. Note that both variables should be of the same type (floating-point/ floating-point, integer/ integer or string/ string). SWAPping different variable types results in a TYPE MISMATCH ERROR.

Examples:

SWAP A,SD exchanges the contents of A and SD.

SWAP BF$(37),D$(2,3) exchanges the contents of the array elements BF$(37) and D$(2,3).

SWAP W%,IR causes a TYPE MISMATCH ERROR, since W% is an integer and IR is a floating-point variable.

NEW (177) (c)

NEW works the same as the BASIC 2.0 command of the same name: It clears BASIC memory of all program code and variables. BeckerBASIC's NEW clears stack memory, as well as initializing the stack pointer for the REPEAT, WHILE, LOOP and PROCEDURE commands (see Chapter 6 for more information).

RESET (175) (c)

RESET performs a partial reset of your computer. That is, it and BeckerBASIC return to start-up status. The video chip, as well as all pointers, (variable pointer, stack pointer, etc.) are reset. Also, all error traps such as ONKEYGO, ONERRORGO, STOPOFF, etc. are cleared.

A BASIC program deleted with this command can be restored with POLD (see Section 2.1.1). If GEOS is in memory, it is unaffected.

Format: RESET

DESKTOP (008) (c)

DESKTOP returns you to the GEOS deskTop from BeckerBASIC, provided the deskTop is on the diskette currently in the drive. BeckerBASIC and any program in memory are erased before the deskTop reloads.

Format: DESKTOP

2. Program development

2.1 Utilities

This section describes the programming utilities available in the Input-System. If you try using these utilities in any other system, you'll get an ILLEGAL COMMAND ERROR. The exceptions are PBCEND and GTBCEND, which can be used in either system.

2.1.1 Programming commands

Here are the commands you'll use most frequently in program development:

PAUTO **(009)** **(c)**

This enables automatic line numbering.

Format: PAUTO FL,LI

FL is the first line number you want given. Values for FL can range from 0 to 63999.

LI is the increment between line numbers. Values for LI can range from 1 to 255.

Here's how it works: After you type in a command and press the <RETURN> key, the next line number appears on the next line, followed by the cursor. Now you enter your program text. Press the <RETURN> key again to get a new line number and new program line. This next line will be LI higher than the earlier line number (e.g., if LI=10, then the line following 200 would be 210, etc.). Pressing <SHIFT><RETURN> disables auto line numbering.

Example:

PAUTO 100,5 makes the first program line 100, followed by 105, 110, etc.

PRENUMBER (010) (c)

This command renumbers program lines. All branch commands like GOTO and GOSUB are unchanged, however. The reason is that BeckerBASIC allows you to jump to labels and calculated line numbers. Changing line numbers is unnecessary with labels, and calculated line numbers are self-adjusting (e.g.,GOTO A*2+10).

Why have a RENUMBER command? When you run short of program lines (e.g., when you want to insert a line between lines 10 and 11), PRENUMBER can make room between line numbers.

Format: PRENUMBER NL,LI[,[SL][- EL]]

NL is the first new line number of the program or program range being renumbered. Values for NL can range from 0 to 63999.

LI is the increment between lines (see PAUTO) once they are renumbered. Values for LI can range from 1 and 255.

If you don't want to renumber the entire program, you can add the additional parameters to limit the procedure to a selected range of lines.

SL,EL SL is the first line and EL is the last line of the range to be renumbered. The parameters can be stated in the same way as the LIST command: SL, SL- or -EL.

Examples:

PRENUMBER 1000,10 numbers the entire program in steps of 10. The first new line is 1000.

PRENUMBER 100,5,-200 numbers the program from the start to line 200 in steps of 5. The first new line=100.

PRENUMBER 5000,2,4500-5000 renumbers lines 4500-5000 in increments of 2, starting at line 5000.

PMERGE (012) (c)

PMERGE allows the merging of BASIC programs on diskette. The line numbers of the programs make no difference, since PMERGE can merge any program. The program to be merged sorts with the program in memory line for line (old lines are deleted if line numbers match).

Format: PMERGE MN$

MN$ is the name of the merged program. PMERGE deletes all variables, so you may want to use DOVERLAYK and DOVERLAYW (see Section 5.3.3), which do not delete variables.

NOTE: To avoid syntax errors in the program, make sure that no lines in the program being loaded are overwritten by PMERGE. Merged program lines with smaller line numbers than the current program will usually result in a program stopping. PMERGE should only be used in direct mode.

PDEL (013) (c)

PDEL deletes a single line or a series of lines from a program. Like PMERGE, PDEL deletes all variables:

Format: PDEL [[L1]-[L2]] [[L3]-[L4]]][,...]

L1-L4 are the line numbers or the starting and ending line numbers of the range(s) to be deleted. To delete several lines or a range, you can use - to connect ranges and commas to separate each range.

NOTE: If you use this command in program mode, do not delete the program lines preceding or containing this command.

Examples:

PDEL 10,20,30 deletes program lines 10, 20 and 30.

PDEL 10-20,30- deletes program lines from 10 to 20, then lines 30 to the end.

PDEL 10,1000-1040 deletes line 10, as well as lines 1000 to 1040.

POLD (011) (c)

POLD restores a BASIC program just deleted with NEW, RESET or PDEL (variable contents are unrestored).

It's important that you type in this command immediately after typing NEW, RESET or PDEL. If you type in a new program line, you won't be able to restore your program. This command works only in direct mode:

Format: POLD

PBCEND (014) (c)
GTBCEND (249) (f)

PBCEND changes the top of memory for BASIC programming. The default value for this top of memory is around 32575. PBCEND is commonly used in dialogue box and drop-down menu creation (see Chapter 7). GTBCEND returns the current top of BASIC memory.

Format: PBCEND EN: ... :EN = GTBCEND

EN is the desired or the current top of BASIC memory. EN should be no higher than 32575.

PMEM (015) (c)

PMEM displays the current BASIC memory layout. After you type in PMEM, the output appears in the format:

```
PROGRAM:       00000
VARIABLES:     00000
ARRAYS:        00000
BYTES FREE:    00000
```

The current values appear instead of these zeroes. All values represent bytes.

PDUMP (203) (c)

PDUMP list the currently defined variables, their names and current values.

Format: PDUMP

Example:

```
AD      =       123.45
F%      =       -14562
GT      =       V$="TEXT"
W       =       -3
BN$     =       "EXAMPLE"
```

2.1.2 Function keys

You can program function keys to print frequently used commands or strings.

PDFKEY (016) (c)

PDFKEY assigns a text to a function key. This text can be up to ten characters long.

Format: PDFKEY (NR) = (TX$)

NR is the number of the function key to be pressed. This number corresponds to the keyboard layout of the Commodore 64. NR can be a value from 1 to 8. Values above or below this range result in an ILLEGAL QUANTITY ERROR.

TX$ contains a text used by the function key. Strings longer than 10 characters produce a STRING TOO LONG ERROR. Commands can be abbreviated to three or four characters, so this is not a big disadvantage.

NOTE: To set up a function key so that it does nothing, you must include CHR$(0). You can do this either with PDFKEY(NR) = (CHR$(0)) or PDFKEY (NR)=("").

PKEY (017) (c)

PKEY lists the current function key setup on the screen.

Format: PKEY

Here's a typical display:

```
F1:     RUN
F2:     PMEM
F3:     PDUMP
F4:     LIST
F5:     POLD
F6:     TRON
F7:     TROFF
F8:     COLORS
```

PFKEYON (179) (c)
PFKEYOFF (180) (c)

PFKEYON turns the function key setup on, and PFKEYOFF turns the setup off.

Before you use PFKEYOFF for the first time, the function key setup must already be turned on with PFKEYON. Each function key contains CHR$(0) when turned off (see the NOTE under the entry for PDFKEY above).

Function key assignments are active in direct mode only. Program mode can use function keys also, without turning off the setup with PFKEYOFF.

NOTE: To execute a command assigned to a function key without pressing the <RETURN> key, add a CHR$(13) to the end of the assignment for that function key. For example, assign this command to the <F1> key:

```
PDFKEY (1) = ("RUN"+CHR$(13))
```

Now when you want to run a BASIC program in memory, just press the <F1> key.

When you have the ability to turn the function key layout on or off, is a question of keyboard priority. Since interrupts control the keyboard reading system, the function key layout set by PDFKEY has highest priority.

The function key layout turned on by PFKEYON has higher priority than all other function key settings. Other setups are assigned CHR$(0), so they cannot execute.

2.2 Error handling

This section describes the commands available in both the Input-System and the Testing-System for testing programs and handling errors. The TRACE command is one of these, and can help you understand the workings of a very complex program.

The BeckerBASIC error handling system operates on three levels: The lowest level corresponds to the standard BASIC 2.0 error display; when an error occurs, BASIC displays a message on the screen.

ERRSHOWON	(030)	(c)
ERRSHOWOFF	(031)	(c)

The second level of error handling displays the incorrect syntax in reverse video. You can turn on this second level of error handling with ERRSHOWON and off with ERRSHOWOFF.

NOTE: Remember three points about the ERRSHOWON command:

1) You cannot have ONERRORGO (see Section 2.2.2) and ERRSHOWON on at the same time.

2) If the incorrect line appears in the last two lines of the screen, the reverse video display may appear in the wrong area.

3) If you scroll the incorrect line up when listing, the reverse video display may appear in the correct column, but a line or two too low. If the error is at the end of a program line, it may be impossible to display the bad area in reverse video.

These last two items can be bypassed if you remember the following rule: If an incorrect line isn't in reverse video, then look at the end of the line for the incorrect command.

ONERRORGO (025) (c)
ONERROROFF (026) (c)

The third and most user-friendly level is the ONERRORGO command. It is the only error tool which can be used in the Run-Only-System. This lets you branch to a program line, and assign a variaable for holding the error message, as well as the incorrect line's number.

Format: ONERRORGO LN, FN, FT$, FZ

LN is the line number to which the program should branch on an error.

FN is the variable containing the error number. See Appendix B for a list of all error messages.

FT$ is the string variable in which the error text is stored. Error texts are similar to texts normally displayed on the screen (e.g., SYNTAX ERROR, ILLEGAL QUANTITY ERROR, etc.).

FZ is the variable containing the line number of the incorrect line. An error in direct mode assigns FZ a value of 0.

ONERRORGO can be placed anywhere within a program, but you can also define it in direct mode as well. Also, any number of ONERRORGO commands can exist in a program.

When an error occurs, BeckerBASIC displays the last command executed. Like ERRSHOWON, ONERRORGO has an off switch - ONERROROFF (026).

RESUMECUR (027) (c)

RESUMECUR continues program execution after error handling at the current command.

Format: RESUMECUR

RESUMENEXT (028) (c)

RESUMENEXT continues program execution from the command following the command that caused the error.

Format: RESUMENEXT

RESUME (029) (c)

RESUME continues program execution at any point in the program.

Format: RESUME LN:...: RESUME LN$

LN is the line to which the program should jump; LN$ is the label of the
LN$ line to which the program should jump. When the third RESUME command is used without having first run into an error, and without a program jump (e.g., ONERRORGO), the system displays a RESUME WITHOUT ONERRORGO ERROR. The RESUME command can only be used for ending an error handling routine. If you compare ONERRORGO with GOSUB, then RESUME is comparable to the RETURN statement.

NOTE: When you encounter an error in direct mode, do not use the RESUME command.

Error handling with ONERRORGO is complicated, but easy to work with once you learn its essentials.

When you use only one error handling routine within a program, then the ONERRORGO command should be at the beginning of the program. This traps all errors within a program. First you must supply the line number to which the error should branch, followed by the variable names for the error number, error text and incorrect line.

Examples:

ONERRORGO 1000,A,B$,C places the error number in A, the error text in B$ and the error line in C. The program branches to line 1000 when the error occurs.

PZ=3700:ONERRORGO PZ+ER,ER,ER$,EL puts the error number in ER, the error text in ER$ and the error line in EL. The program branches to line 3700+ER, set according to the error number in ER.

Errors can be easily identified by their error numbers, as you saw from ER in the last example. The given error text (ER$ in the last example) can be used to display user information on the screen.

In most cases, the error handling ends with a program break, since it hardly makes sense to continue a program that has errors. Then why is there a RESUME command? This command can be very useful in many cases. Take RESUMECUR, for example. If a program using disk access finds that either the disk drive is turned off or that there is no diskette in the drive, you usually get a DEVICE NOT PRESENT ERROR. ONERROR and RESUMECUR solve these problems:

```
5 'DEMO OF ONERRORGO'
10 ONERRORGO 1010,A,B$,C
100 DLOADM "PRG"
1000 'ERROR HANDLING'
1005 'DEVICE NOT PRESENT ERROR'
1010 IF NOT(A=5) THEN POPIF:GOTO 1500:ENDIF
1020 :
1030 SCPRINT;">>TURN DISK DRIVE ON<<" :SCPRINT
1040 SCPRINT">>INSERT A DISK, AND<<" :SCPRINT">>PRESS A KEY<<"
1050 :
1060 KEYDEL:WAITKEYA:'WAIT FOR A KEYPRESS'
1070 :
1080 RESUMECUR:'GO TO INCORRECT LINE'
1090 '...'
1100 '...'
1110 '...'
1500 'OTHER ERRORS HERE'
```

Line 10 establishes the ONERRORGO parameters. If the LOAD command in line 100 finds that the disk drive is off, then it branches to the error handling routine at line 1000. Line 1010 checks to see if it is actually a DEVICE NOT PRESENT ERROR. If so, line 1030 tells you to turn the disk drive on, insert a diskette and press a key to execute the command (see Section 3.1.1 for more information on KEYDEL and WAITKEYA).

Finally, RESUMECUR executes the normal LOAD command. ONERRORGO and the RESUME command offer interesting and elegant programming options.

2.2.3 The TRACE commands

TRACE displays the program line number currently executing. This is useful for testing program flow and getting a better understanding of program structure.

Of particular interest is single-step mode, which lets you single-step through a program (command by command). Pressing a key (the <CTRL> key in BeckerBASIC) moves the program from one command to the next. Single-step mode is the best method of seeing what a program does and when. BeckerBASIC's TRACE command does still more.

The program being edited can be displayed in any area of the screen. The beginning of the next command to be executed appears in reverse video. The <F1> and <F3> keys turn the screen display on or off during program execution.

The biggest disadvantage of the TRACE commands in program mode is setting up the TRACE parameters and switching on the TRACE mode with a command.

If you exit a program in normal mode, you must first turn off all TRACE commands. You have to start the TRACE mode from the beginning of the program.

BeckerBASIC gets around this disadvantage by splitting the mode into three commands.

TRACE (032) (c)

TRACE assigns the necessary parameters to trace mode. You can use as many TRACE commands as you wish within a program. TRACE should be the last command in that mode.

Format: TRACE LN, VW, AF

LN is the screen line number at which the program line to be traced should appear. Values for LN range from 1 (topmost line) to 25 (bottom line). The bottom two lines (lines 24 and 25) do the same as in ERRSHOWON: If the current line scrolls up during output, the reverse video could end up one or two lines too low. Therefore, try to stay away from the last two screen lines.

VW is the value assigned to the delay loop. This loop sets the time delay between commands. Values for VW can range from 0 to 255. The longest possible delay occurs when VW=1; the shortest possible delay results when VW=255. VW=0 turns on single-step mode.

AF determines whether or not the program line in process should be displayed on the screen or not. If AF=0, output is suppressed. If AF=1, the current line set in LN appears on the screen. If you want to turn on the screen output only in selected places, set AF to 0 and input the desired line in LN. <F1> and <F3> turns the output on and off.

As mentioned above, pressing the <CTRL> key executes the next command in single-step mode. This also applies to direct mode (when you start a program with RUN, you must press the <CTRL> key as well as the <RETURN> key).

The remaining functions of all the TRACE modes work in both direct mode and program mode. NOTE: The current command display is unavailable in direct mode.

When reading program lines on the screen, the TRACE routines use the available command name tables. These tables are in the hi-res graphic bitmap memory (see Appendix C). When you use hi-res graphics in a program, these tables are overwritten. Therefore, you should switch into hi-res graphics for program output after you turn off TRACE (setting AF to zero), and leave the TRACE mode off. Otherwise, you could get a system error. Besides that, to use program line output, you should first load the Input-System into the computer, then toggle over to the Testing-System, so that the name tables load over from the Input-System.

TRON (006) (c)

TRON turns TRACE on. All commands following this (up to the last TRACE command) run under a time delay.

Format: TRON

TROFF (167) (c)

TROFF turns TRACE off, returning the computer to normal mode.

Format: TROFF

Both the TRON and TROFF commands can be used within a program as many times as you wish. TROFF has no effect in normal mode.

Examples:

TRACE 5,100,1 sets screen line 5 as the output line. A delay value of 100 is given. This display follows immediately after the TRON command (AF=1).

ZE=1:EM=0:TRACE ZE,EM,0 makes the topmost line the display line. EM=0 turns on single-step mode. The 0 suppresses the output.

3. Input and output

The most important areas of programming are the lines of communication between user and computer, and computer and peripherals. This chapter is devoted to input and output, paying particular attention to screen and cursor control.

3.1 Data input

3.1.1 Keyboard input

The most important input device is the keyboard. BeckerBASIC has numerous commands for making keyboard input easier and more comfortable.

KEYREPEATON (033) (c)

KEYREPEATON switches on the keyboard repeat function. Note that the cursor is turned on with the repeat function.

Format: KEYREPEATON

The speed at which the key repeats is adjusted by the CRFREQ command (see Section 3.4 for more information).

KEYREPEATOFF (034) (c)

KEYREPEATOFF turns the keyboard repeat function off.

Format: KEYREPEATOFF

STOPOFF (036) (c)
STOPON (035) (c)

The keyboard of the C64 has one key that can be a nuisance, the <STOP> key. If a user presses this key at the wrong time (e.g., during diskette access), serious problems could result. BeckerBASIC offers the STOPOFF and STOPON commands.

STOPOFF disables the <STOP> key. A running program cannot be stopped by pressing the <STOP> key.

Format: STOPOFF

STOPON has the opposite effect of STOPOFF: The <STOP> key is enabled - a running program can now be stopped by pressing the <STOP> key.

Format: STOPON

The next command extends BASIC's ability to read the keyboard.

KEYDEL (176) (c)

The C64 has a keyboard buffer into which up to 10 characters (keypresses) are stored. The computer reads the keypresses from this buffer. Since the keypresses register through an interrupt, the buffer may already be full after every keypress.

The buffer may read a previous keypress instead of the input you want it to read, resulting in an error. The keyboard buffer can be deleted with the KEYDEL command.

Format: KEYDEL

WAITKEYA (037) (c)

WAITKEYA waits for any keypress. The keyboard buffer is deleted before reading, so KEYDEL is unnecessary in this case. This command can be used in connection with GET.

Example:

```
10 PRINT"PLEASE PRESS THE <A> KEY."
20 WAITKEYA:'WAIT FOR A KEYPRESS'
30 :
40 GET EG$:'READ KEYPRESS'
50 :
60 'IF IT IS NOT <A>, WAIT UNTIL IT IS'
70 IF NOT(EG$="A") THEN POPIF:GOTO 20:ENDIF
```

WAITKEYS (038) (c)

WAITKEYS waits for a specific keypress, assigned with the ASCII code of the desired key (see your C64 manual for ASCII codes).

Format: WAITKEYS TE

TE is the ASCII code of the desired key. Values for TE can range from 0 to 255. WAITKEY 65 waits for the <A> key.

KGETV (039) (c)

This command is similar to the BASIC 2.0 GET command: It reads data from the keyboard. However, it is much more flexible than GET.

Format: KGETV VR$,LE [,K$]

VR$ is a string variable which recieves the input.

LE sets the input length. Values for LE can range from 1 to 255.

K$ limits the amount of input allowed. All keys you want included must be in K$. For example, if you want only the numbers from 0 to 9 read as legal input, K$ would equal "0123456789". The K$ parameters are optional; if parameters are included, they must be enclosed in quotation marks.

KGETV reads data only as a string. Input can be changed to numeric input using the BASIC 2.0 VAL command.

Examples:

```
10 SCPRINT "ENTER A NUMBER BETWEEN 1 AND 5."
20 KGETV EG$,1,"12345":'READ NUMBER
30 EG = VAL(EG$):'CONVERT TO NUMBER
40 SCPRINT EG" IS THE NUMBER YOU SELECTED."

10 ZL$ = "+-0123456789":'LEGAL CHARACTERS'
20 'READ 4-DIGIT NUMBER WITH LEADING CHAR'
30 KGETV EG$,5,ZL$:A = VAL(EG$):PRINT A

10 SCPRINT "MENU":'DISPLAY MENU'
20 SCPRINT "MODULE A: A"
30 SCPRINT "MODULE B: B"
40 SCPRINT "YOUR CHOICE (A OR B)?"
50 KGETV MN$,1,"AB":'MODULE CHARACTERS'
60 GOTO MN$:'JUMP TO DESIRED MODULE'
70 "A":SCPRINT"MODULE A":END
80 "B":SCPRINT"MODULE B":END
```

KBGETV (040) (c)

KBGETV is similar in format to the KGETV command. However, this command displays the character at the current cursor position, which can be useful for longer inputs.

Format: KBGETV VR$,LE [,K$]

See KGETV above for these parameters.

Example:

```
10 CRSET 5,1:'SET CURSOR'
20 PRINT "YOUR INPUT:";
30 CRON:'CURSOR ON"
40 KBGETV D$,10:'10-CHARACTER INPUT"
50 CROFF:'CURSOR OFF'
```

As you can see in the example, the input goes where the cursor is assigned (see Section 3.4 for more information on cursor commands).

Cursor control is used here for controlling the reading of the string during input. If you'd prefer to avoid this command, there are alternatives in Section 3.1.2.

KGETV and KBGETV are intended for shorter input. However, there are other commands in this chapter which can handle input on an entire screen page.

The next two commands can read the <SHIFT>, <CTRL>, <Commodore> keys and others.

STTEST (232) (f)

STTEST checks for input from one of these alternate keys.

Format: WT = STTEST

WT can also test for the <SHIFT>, <Commodore> and <CTRL> keys. WT can be assigned the following values to show one or more of these keys pressed:

```
0       NONE OF THESE THREE KEYS
1       <SHIFT> KEY
2       <COMMODORE> KEY
3       <SHIFT> AND <COMMODORE> KEYS
4       <CTRL> KEY
5       <SHIFT> AND <CTRL> KEYS
6       <COMMODORE> AND <CTRL> KEYS
7       <SHIFT>, <CTRL> AND <COMMODORE> KEYS
```

Example:

IF STTEST=5 THEN POPIF:GOTO 1000:ENDIF branches to 1000 if the <SHIFT> and <CTRL> keys are pressed during the current keyboard reading.

WAITST (178) (c)

WAITST waits for one or more alternate keys to be pressed, then immediately continues on with the next command in the program.

Format: WAITST GT

GT determines which key or key combination to expect. The list of values is the same as for STTEST (i.e., GT=1 means the <SHIFT> key, etc.). Values for GT range from 0 to 7. Any numbers outside this range result in an ILLEGAL QUANTITY ERROR.

Example:

WAITST 3:SCPRINT "SHIFT+COMMODORE" waits until the <SHIFT> and <Commodore> keys are pressed simultaneously.

When these comands are used in conjunction with the ONKEYGO command (see below), a whole new set of programming possibilities opens. You can even jump to a predetermined program routine while editing a program in direct mode.

ONKEYGO (041) (c)

The ONKEYGO command allows you to interrupt a program with a keypress, branch to a subroutine and continue the main program at the point at which the ONKEYGO occurred.

Format: ONKEYGO CR,LN

CR contains the ASCII code of the key pressed. Values for CR theoretically range from 0 to 255.

LN is the line to which the program should jump on this keypress. Values for LN range from 0 to 63999. Numbers outside of this range result in an ILLEGAL QUANTITY ERROR.

You can have as many ONKEYGO commands in a program as you wish, however only the last ONKEYGO command is active. NOTE: Multiple definitions are not allowed (e.g., branching to line 1000 when the <A> key is pressed, line 2000 when the <B> key is pressed, etc.).

Examples:

ONKEYGO 65,5000 branches to line 5000 when the user presses the <A> key.

ONKEYGO ASC("A"),5000 performs the same function. When you don't know the ASCII code for a character, you can use the ASC function.

ONKEYGO 137,61000 branches to line 61000 when the user presses the <F2> key.

RETKEY (042) (c)

RETKEY acts as the close of a subroutine branched to by ONKEYGO.

Format: RETKEY

If a program encounters a RETKEY without having first executed an ONKEYGO, the result is a RETKEY WITHOUT ONKEYGO ERROR.

To get a better grasp of what happens, here are descriptions of what occurs after ONKEYGO:

Direct mode: Direct mode branches direct to the given program line and runs the program code to the next RETKEY command. After RETKEY, the computer returns to direct mode.

Program mode: The program executes to the end of the current line, the next line number is stored in a buffer. The program then branches to the line number specified in the ONKEYGO command. When RETKEY is encountered the line number in the buffer is used to return to the main program, and executes the next command in the main program.

ONKEYOFF (166) (c)

There are two options for cancelling an ONKEYGO definition: Either you set a new definition, or you invoke ONKEYOFF.

Format: ONKEYOFF

The ONKEYGO command should be turned off at the end of a program with ONKEYOFF. The reason is that the ONKEYGO may accidentally branch to a program line when in direct mode.

3.1.2 Screen input

The screen is not an input device, and doesn't directly provide data input. Still, "screen input" describes the process of displaying keyboard input on the screen, and BeckerBASIC has numerous commands for this type of programming.

To write data easily on the screen, there are many commands. One small example is the WINPROC procedure at the end of Chapter 6. With this program, you can define input windows of any size and type on the text screen, store screen contents in a buffer and restore these contents on the screen.

As long as the data only shows on the screen, it is not very useful. BeckerBASIC has two commands to transfer screen data into computer memory or a string variable.

SGETV (043) (c)

SGETV converts screen data into a string variable.

Format: SGETV VR$, LE, RO, CO

VR$ is the name of the string variable to which the data is assigned.

LE gives the number of characters to be read, based on RO and CO's screen position. Values for LE can range from 1 to 255.

RO are the row (RO) and column (CO) of the screen position from which
CO the data is read. After command execution, the cursor returns to the home position of the screen. Values for RO range from 1 to 25, while values for CO range from 1 to 40.

Examples:

SGETV EG$,10,17,5 puts 10 characters from row 17, column 5 into the variable EG$.

CRHOME:SCPRINT"HELLO":SGETV T$,5,1,1 puts the text HELLO into T$. CRHOME places the cursor in the home position of the screen (see Section 3.4).

SGETM (044) (c)

SGETM reads screen data and stores it in a preassigned area of memory.

Format: SGETM SA, LE, RO, CO

SA gives the starting address of the memory range into which the data is stored. Values for SA can range from 0 to 65535.

LE gives the length of the data being read, based upon RO and CO as the starting point. Values for LE range from 1 to 255.

RO are the row (RO) and column (CO) of the screen position from which
CO the data is read. After command execution, the cursor returns to the home position of the screen. Values for RO range from 1 to 25, while values for CO range from 1 to 40.

Unlike the TRANSFER command described in Chapter 4, SGETM prepares the memory for storage and processes the data into a variable, which it converts from BSC (true ASCII) to Commodore ASCII code.

There are two good places to store data:

The hi-res bitmap (40960 to 48960) gives you 8000 bytes. Naturally, you can only use this range if you aren't using hi-res graphics.

Smaller quantities of data can be stored in the cassette buffer from memory locations 828 to 1023.

Both areas of memory have the advantage that they lie outside of BASIC memory, and thus won't disturb that memory.

Examples:

SGETM 41000,22,3,7 reads 22 characters from row 3, column 7 and puts these characters into memory starting at memory location 41000.

SGETM 828,50,1,10:MGETV EG$,10,828 reads 50 characters starting at row 1, column 10, and stores the characters starting at memory location 828. The MGETV (see Section 4.3) places the first ten characters into the variable EG$.

3.2 Data output

3.2.1 Screen output

To make all screen output easier, BeckerBASIC includes the commands SCPRINT and AT.

SCPRINT (047) (c)

SCPRINT is much the same as the BASIC 2.0 PRINT statement. You can position the text when you add AT to SCPRINT (see below).

AT (048) (c)

AT puts the cursor at a specified screen position. This command can be used only in connection with the SCPRINT command.

Format: SCPRINT [AT RO,CO;]"EXPRESSION"

RO RO is the row position and CO is the column position at which the text
CO appears. Values for RO range from 1 to 25, while values for CO range from 1 to 40. The EXPRESSION follows CO, separated by a semicolon. The expression between quotation marks appears at the cursor position marked by RO and CO. You can omit the expression between quotation marks just to position the cursor without text. The semicolon cannot be omitted.

Remember that the AT must immediately follow SCPRINT. Syntax like SCPRINT "TEXT ",AT is not allowed. Another consideration in SCPRINT is the status of the BeckerBASIC RVS flags (see RVSON and RVSOFF below).

You may find it easier to set cursor positioning with the CRSET command (see Section 3.4).

RVSON	(049)	(c)
RVSOFF	(050)	(c)

A disadvantage to the PRINT command in BASIC 2.0 is the fact that the computer changes reverse video to normal video when the end of a PRINT statement is reached.

If you wish to display longer PRINT statements in reverse video, you must end each PRINT statement with a semicolon. This makes it much more difficult to plan screen format. BeckerBASIC solves this problem with the RVSON and RVSOFF commands.

RVSON turns reverse video on. All output in a SCPRINT command appears in reverse video. RVSOFF turns the text back to normal mode.

Example:

```
10 RVSON:'REVERSE MODE ON'
20 SCPRINT AT 1,1;"HERE"
30 SCPRINT AT 2,5;"IS"
40 SCPRINT AT 3,7;"AN"
50 SCPRINT AT 4,9;"EXAMPLE"
60 RVSOFF:'REVERSE MODE OFF'
```

LETTERON	(133)	(c)
LETTEROFF	(134)	(c)

LETTERON turns on lowercase lettering, while LETTEROFF returns the system to uppercase lettering. You can switch ba1ck and forth between modes in direct mode by pressing <SHIFT><Commodore>. LETTERON and LETTEROFF were intended for use in program mode.

LOCKON	(135)	(c)
LOCKOFF	(136)	(c)

LOCKON disables the uppercase/lowercase toggling. LOCKOFF enables the toggling. LOCKON is useful when you want to keep the user from switching between character sets.

3.2.2 Printer output

BeckerBASIC provides two easy methods for printing data.

PRPRINT (171) (c)

PRPRINT sends any alphanumeric data to the printer, much like the SCPRINT command. The rules for PRPRINT are identical to those used in SCPRINT and the BASIC 2.0 PRINT statement.

Format: PRPRINT "EXPRESSION"

PRCOM (172) (c)

PRCOM sends individual printer codes to the printer. This is especially useful for sending control codes such as bold, expanded print, etc.

When used within a program line, PRCOM can have as many control codes as you like, as long as each code is separated from the next by a comma. PRCOM is the same as the BASIC 2.0 sequence:

OPEN 14,4:PRINT#14, CHR$(C1):CLOSE14

Format: PRCOM C1[,C2,C3...]

C1,C2 is the code normally sent in the form of a CHR$(..) code. PRCOM can handle individual printable characters. The ASCII code must be concluded by a <RETURN> (ASCII code 13). PRCOM ASC("A"),13 sends an A.

NOTE: All BeckerBASIC printer commands correspond to the BASIC 2.0 sequence:

```
OPEN 14,4:PRINT#14,...:CLOSE14
```

Never use any other file commands with a logical file number of 14! This number was assigned to the printer since it's an unusual logical number.

3.3 Screen management

This section includes commands for clearing and changing the screen, as well as loading and saving areas of the screen.

PCOLORS (018) (c)

PCOLORS states the list of colors available to the user, and their respective color numbers.

When you have trouble remembering the correct color and number, just enter PCOLORS and press the <RETURN> key to display the following table:

0	black	8	orange
1	white	9	brown
2	red	10	lt.red
3	turquoise	11	grey 1
4	purple	12	grey 2
5	green	13	lt.green
6	blue	14	lt.blue
7	yellow	15	grey 3

BORDER (051) (c)
CLBORDER (246) (f)

BORDER changes the screen border color. CLBORDER reads the current border color.

Format: BORDER FN: ... :FN = CLBORDER

FN is the color code corresponding to the output from PCOLORS. Values for FN can range from 0 to 15.

Values for BORDER can theoretically range from 0 to 255, although once you pass 15, the color numbers just repeat (16=0, 17=1, etc.).

GROUND (052) (c)
CLGROUND (245) (f)

GROUND changes the screen background color. CLGROUND reads the current background.

Format: GROUND FN: ... :FN = CLGROUND

FN represents the background color. Values for FN can range from 0 to 15.

Values for GROUND can theoretically range from 0 to 255, although once you pass 15, the color numbers repeat (16=0, 17=1, etc.).

CLS (053) (c)

CLS clears the text screen, and corresponds to the BASIC 2.0 statement PRINT CHR$(147). The cursor moves to the home position after the screen clears. To move the cursor to the home position without clearing the screen, use the CRHOME command (see Section 3.4).

Format: CLS

SCRON (054) (c)
SCROFF (055) (c)

These commands turn the screen on (SCRON) and off (SCROFF) through software. These don't literally turn the screen power on or off; they blank out the screen.

Format: SCRON: ... :SCROFF

This is useful for quickly blanking and retrieving screen masks.

SCRDSAVE	(056)	(c)
SCRDLOAD	(057)	(c)

SCRDSAVE stores the current screen to diskette. SCRDLOAD loads a stored screen file.

Format: SCRDSAVE NA$: ... :SCRDLOAD NA$

NA$ is the name under which the screen is stored. This name can have a maximum of 16 characters.

SCRDLOAD loads direct into the current screen and overwrites the old screen. SCRDSAVE is used to save a screen mask setup for later recall.

3.4 Cursor control

CRHOME	(058)	(c)

CRHOME moves the cursor to its home position (the upper left corner of the screen).

Format: CRHOME

CRSET	(059)	(c)
CRPOSL	(233)	(f)
CRPOSC	(234)	(f)

CRSET sets the cursor at any location on the screen. CRPOSL (line) and CRPOSC (column) read the current cursor position.

Format: CRSET RO,CO: ... :RO = CRPOSL: ... :CO = CRPOSC

RO RO is the current row position and CO is the current column position
CO returned by CRPOSL and CRPOSC. Values for RO range from 1 to 25, while values for CO range from 1 to 40.

CRCOL (060) (c)
CLCURSOR (247) (f)

CRCOL changes the cursor color and the text color. CLCURSOR reads the current cursor color.

Format: CRCOL FN: ... :FN = CLCURSOR

FN is the color code corresponding to the output from PCOLORS. Values for FN can theoretically range from 0 to 255, although once you pass 15, the color numbers repeat (16=0, 17=1, etc.).

CRON (061) (c)
CROFF (168) (c)

CRON turns on the cursor at the current cursor position. This is useful when you want the user to make an important input. CROFF turns the cursor off again.

Format: CRON: ... :CROFF

CRREPEATON (062) (c)
CRREPEATOFF (063) (c)

CRREPEATON turns on the cursor repeat function, i.e., the repeated movement of the cursor as you hold down one of the cursor keys (this function is built into the operating system). This command acts much the same as the KEYREPEATON command (see Section 3.1.1). CRREPEATOFF turns off cursor and keyboard repeat.

Format: CRREPEATON: ... :CRREPEATOFF

CRFREQ (064) (c)

CRFREQ changes the cursor and keyboard reading frequency. That is, it changes the speed at which the cursor movement and keyboard output occur (faster or slower).

Format: CRFREQ NR

NR is the rate at which the system reads the cursor and keyboard. Values for NR range from 0 to 255. It is best to use values between 25 and 125. Smaller numbers cause faster movement, while larger values slow the movement.

Another effect can be put to good use with BASIC programs that depend on time: The slower the cursor movement (i.e., the larger the NR value), the faster a BASIC program executes, in cases of NR values larger than 125.

4. Memory access commands

This chapter describes the most important memory commands. The most vital commands are MYFILL, which fills a memory range with the user's choice of characters; and TRANSFER, which moves any area of memory to another area of memory (e.g., the character generator in ROM). See how the original memory range, the destination and end range can overlap in Section 4.3, using the VGETM, MGETV and VARADR commands.

4.1 Working with memory ranges

The following four commands are intended specifically for handling memory, from a large range of memory down to a single memory location.

TRANSFER (065) (c)

TRANSFER moves a designated memory range to a free area of memory. The memory range to be moved can be in ROM as well as RAM. The destination range can only be in RAM, since you can't write to ROM.

Format: TRANSFER BA, BE, NA [,KN]

BA BE — BA is the first memory location of the range. BE is the ending location of the range. Values for BA and BE can range from 0 to 65535.

NA — contains the starting address of the destination range to which the memory range is transferred. Values for NA can range from 0 to 65535.

KN — states the type of memory to which BA and BE are being moved. KN=1 is the character generator; KN=3 is RAM, and KN=5 is the ROM. The default value for KN is 3 (RAM).

Examples:

TRANSFER 1024,1062,1025 transfers the contents of the topmost screen line one location to the right.

TRANSFER 1025,2023,1024 moves the entire screen one character to the left.

TRANSFER 1024,2023,42000 puts the current screen contents at memory location 42000.

TRANSFER 42000,42999,1024 wipes out the screen.

TRANSFER 48000,48100,48000,5 moves the ROM from 48000 to 48100, and transfers it to the RAM below it.

```
10 AD = 1024+40*(ZE-1)+(SP-1)
20 TRANSFER AD,AD+(LE-1),ZL
```

This short program takes the contents or the screen at row ZE, column SP and a length of LE, and puts it at destination range ZL.

MYFILL (067) (c)

MYFILL fills the specified memory range with a given value or character.

Format: MYFILL BA, BE, WT

BA BA gives the first memory location to be filled, BE is the last memory
BE location to be filled and WT the fill value. Values for WT can range
WT from 0 to 255.

Examples:

MYFILL 1024,1103,32 clears the topmost screen lines.

MYFILL 55296,55495,1 colors the first five screen lines white.

```
10 CLS:SCPRINT AT 14,5;"MYFILL  DEMO"
20 PAUSE 2
30 FOR WT=0 TO 255
40 MYFILL 1024,2023,WT:PAUSE .75
50 NEXT WT
```

This program displays each character on the screen, 1000 characters at a time.

BSCASCW	**(070)**	**(c)**
ASCBSCW	**(069)**	**(c)**

When you set data into RAM from the screen (e.g., with TRANSFER), a problem can occur: Screen memory data is in BSC format (Berkeley Softworks Code - true ASCII), while the strings must be in Commodore ASCII format for editing.

BSCASCW converts these strings from BSC to ASCII format. When you need to transfer string data to a memory range on screen, you need to convert it from ASCII to BSC format. The ASCBSCW performs this conversion.

Format: BSCASCW BA, BE: ... :ASCBSCW BA, BE

BA BA give the starting address and BE the ending address of the memory
BE range to be converted.

Examples:

BSCASCW 41000,41500 converts the RAM area from 41000 to 41500 from BSC code to ASCII code.

ASCBSCW 47000,48000 converts the RAM area from 47000 to 48000 from ASCII to BSC code.

CRHOME:SCPRINT "PICT":BSCASCW 1024,1027 converts the text "PICT" (visible on the screen) from BSC to ASCII code.

4.2 Accessing individual memory locations

In addition to the BASIC 2.0 POKE and PEEK commands, BeckerBASIC offers the following memory access commands.

DOKE (066) (c)

DOKE places a value in two consecutive memory locations, unlike POKE, which accesses just one memory location. The value is divided into low byte/high byte format.

Format: DOKE AD,WT

AD is the starting memory location. Values for AD can range from 0 to 65535.

WT is the value inserted into memory locations AD and AD+1. Values for WT can range from 0 to 65535. DOKE can be assigned parameters for machine language programs.

Examples:

DOKE 48000,35000 puts the value 35000 into memory locations 48000 and 48001. After execution, memory location 48000 contains 184 and 48001 contains 136.

DOKE 828,VR(10) puts the contents of the array element VR(10) into locations 828 and 829.

DEEK (235) (c)

DEEK reads the contents of two consecutive memory locations and gives the total value as a variable. The first memory location is read as the low byte, while the second location is read as the high byte.

Format: VR = DEEK (AD)

AD is the starting address of the two memory locations read. Values for AD range from 0 to 65535.

Examples:

WT = DEEK(50000) gives the contents of locations 50000 and 50001 in WT. If location 50000 contains 10 and location 50001 contains 120, then WT returns the value 30730.

SCPRINT DEEK(43) displays the start of BASIC pointer on the screen.

TEEK (236) (f)

Like PEEK, TEEK reads individual memory contents. In addition, it determines whether the character generated is in ROM or RAM.

Format: VR = TEEK (AD [,KN])

AD is the desired memory address. Values for AD can range from 0 to 65535.

KN states the type of memory accessed. KN=1 is the character generator; KN=3 is RAM, and KN=5 is the ROM. The default value for KN is 3 (RAM).

Examples:

W = TEEK (56325,1) reads the contents of memory location 56325 in the character generator.

W = TEEK (56325) reads RAM location 56325 (from the CIA register).

4.3 Exchanging memory and variable contents

The memory range from 40969 to 48960 is hi-res memory, an ideal area for storing data of all kinds (provided you aren't using the hi-res memory for anything else). These three commands are designed for storing different data.

VGETM (181) (c)

VGETM puts the contents of a specific string variable or alphanumeric expression into RAM memory.

Format: VGETM BA, VR$

BA is the first memory location at which the string data is placed. Values for BA range from 0 to 65535.

VR$ is the given string expression.

Examples:

VGETM 830,"TEXT" puts the string "TEXT" starting at memory location 830.

T$="EXAMPLE":VGETM 42000,"AN "+T$ stores the string "AN EXAMPLE" starting at location 42000.

MGETV (068) (c)

MGETV reads the conttents of a memory range into any string variable.

Format: MGETV VR$, LE, BA

VR$ is the name of the string variable into which the memory contents are loaded.

LE,BA LE is the length of the memory range, BA is the first memory location to be placed in the string variable.

Examples:

MGETV T$,10,890 reads the contents of locations 890 to 899 into variable T$.

```
10 TRANSFER 1024,1028,41500:BSCASCW 41500,41504
20 MGETV EG$,5,41500
```

TRANSFER takes the first five characters of the screen starting at memory location 41500, converts the result from BSC to ASCII code, and puts the result into the variable EG$.

NOTE: When you move data directly from the screen to RAM (e.g., with TRANSFER), you should convert the memory area from BSC code into ASCII code using the BSCASCW command (see Section 4.2).

VARADR (237) (f)

VARADR conveys the starting address of a variable into BASIC variable memory.

Format: V1 = VARADR (VR): ... :V2 = VARADR (VR$)

VR is the starting address of the variable.

VR$ is the name of the variable.

Aside from easy memory access, there are very few uses for VARADR. One possibility of this function lies in the buffer storage of larger variable arrays, or sections of variable arrays. You can compute the first and last array elements and then move the array with TRANSFER (see Section 4.1).

Examples:

A = VARADR(ZT) computes the address of variable ZT and stores it in A.

```
5 DIM A%(55)
10 W1=VARADR(A%(1)):W2=VARADR(A%(52)):'CONVEY ADDRESS
20 W2=W2+1:'ENDADR.+1, AN INTEGER MADE UP OF TWO BYTES'
30 TRANSFER W1,W2,43000:'TRANSFER CONTENTS'
```

This short program transfers the contents of the array elements A%(1)-A%(52) to memory starting at 43000.

The following program lets you put the values in any integer array:

```
10 W1=VARADR(A%(1)):'TRANSMIT STARTING ADDRESS'
20 'STORE VALUE IN INTEGER ARRAY'
30 TRANSFER 43000,43103,W1:'2 BYTES PER VARIABLE * 52'
```

5. Disk commands

The 1541 disk drive is an extremely versatile storage device. It performs simple loading and saving, as well as allowing user-created data access.

The most interesting capabilities of the 1541 can only be achieved by complex programming. And even the simplest tasks, such as deleting a file, involves a bit of program code.

BeckerBASIC has many commands to make your diskette programming easier. For example, deleting a file in BASIC 2.0 required the command OPEN 15,8,15,"S:NAME":CLOSE15. In BeckerBASIC, all you do is type in DSCRATCH"NAME".

Please bear the following rules in mind when using BeckerBASIC diskette commands:

Never use BeckerBASIC diskette commands together with BASIC 2.0 diskette commands, since conflicts with secondary addresses could occur.

Syntax of command parameters is most important at the disk drive level. This is vital when you're uncertain about sending commands on the disk channel (see DSTATUS, Section 5.1).

Three diskette commands can cause trouble when used in conjunction with GEOS disk management:

1) The DSENDCOM "V" command validates a diskette (organizes space). Never use this command on a GEOS diskette, since it could destroy important program information (e.g., the info block) and actual program data.

2) DHEADER formats a diskette. Use this command in GEOS with caution, since DHEADER creates a diskette in normal DOS format. You can correct this by converting the formatted diskette to GEOS format from the deskTop.

3) DRENAME (rename a disk file) should not be used in conjunction with completed BeckerBASIC programs, since a program run through the CONVERTER program could be destroyed when you try to change the directory entry (name) of the program. If you must rename a completed BeckerBASIC program, do it from the GEOS deskTop with the `rename` menu option.

5.1 Common commands

Here are the diskette commands which you'll use most often.

DIR (072) (c)

DIR displays a diskette directory on the screen without disturbing the program in memory.

Format: DIR [SL$]

If you type in DIR without any parameters, the entire directory appears. Pressing the <STOP> key halts the directory display.

SL$ selects certain parts of the directory for display. DIR "$*=P" displays program files (PRG) only; DIR "$*=S" displays sequential files (SEQ); DIR "$*=R" displays relative files (REL); and DIR "$*=U" displays user files (USR).

Along with filetypes, you can use the wildcards * and ? for selecting individual filenames.

The asterisk (*) replaces all characters following it. "$FD*" selects all files starting with the characters FD. "$DIR*" gives all files starting with DIR (e.g., DIRECTORY, DIRTY, etc.).

The question mark (?) can represent any character in a filename. DIR "$AD??CF" lists all six-character filenames starting with AD and ending with CF. The two characters in between can be any letter or number. DIR "$???TT?FP?1" selects all ten-character files containing T as the fourth and fifth characters, F as the seventh character, P as the eighth character and 1 as the tenth character.

The asterisk and question mark can be used together. For example, DIR "$C?T*" reads all files starting with C and containing a T as its third character (e.g., COT, CAT, CITIES, etc.).

The wildcards can also be used in conjunction with the filetype selection. DIR "$OUT*=S" selects all sequential files beginning with OUT.

DSENDCOM (074) (c)

DSENDCOM sends any commands to the disk drive. It is the equivalent of the BASIC 2.0 OPEN 1,8,15, "COMMAND":CLOSE1.

Format: DSENDCOM KN$

KN$ contains the disk command. DSENDCOM "S:NAME" deletes the file NAME. KN$ is a string up to 40 characters in length. Longer strings result in a STRING TOO LONG ERROR.

You'll find other commands in this section that are more convenient to use than DSENDCOM.

DSTATUS (073) (c)

DSTATUS reads the disk error channel.

Format: DSTATUS [FM$]

FM$ is the name of the string variable in which the error message should be placed. If you omit FM$, then the message appears on the screen at the current cursor position. The message appears in the format:

```
ERROR_NUMBER, ERROR_TEXT, TRACK, SECTOR
```

ERROR_NUMBER lists the number on which ERROR_TEXT is based. You can isolate ERROR_NUMBER from the message with FN=VAL(FM$). TRACK and SECTOR indicate the specific data block at which the error occurred. See your 1541 owner's manual for a complete list of error messages.

If the disk status is okay, then the result is 00,OK,00,00. The obvious signal for a disk error is the flashing status light on the disk drive. When that occurs, read the error channel to find out the problem.

DSCRATCH (079) (c)

DSCRATCH deletes files from diskette.

Format: KN$="NAME1[,NAME2,...]":DSCRATCH KN$

KN$ contains the name of the file to be deleted. Additional files can be added to KN$, each separated by commas. The string within KN$ can be a maximum of 38 characters. The wildcards * and ? can be used here, just as in the DIR command. For example, DSCRATCH "N?M*" deletes all files containing N as the first character and M as the third character.

DRENAME (078) (c)

DRENAME renames files stored on diskette.

Format: KN$="NEW NAME=OLD NAME":DRENAME KN$

KN$ contains the new and current filenames. These filenames can be up to 16 characters long.

```
Example:
```

DRENAME "COMPUTATION=TEST.FILE" renames the file TEST.FILE to COMPUTATION.

DHEADER (075) (c)

Before using a new diskette, it must be formatted. The BeckerBASIC formatting command is DHEADER.

Format: KN$="DISKNAME [,ID]":DHEADER KN$

KN$ contains the diskette name and the identification characters (ID). If you omit ID, an already formatted diskette can be cleared and renamed. A new, unformatted disk must have an ID assigned to it the first time you format it. The formatting process takes about 80 seconds.

NOTE: Formatting an already formatted diskette destroys all the data currently on that diskette.

Examples:

DHEADER "TEST,TT" formats a new diskette and assigns it the name TEST and the id TT.

DHEADER "DATA" deletes the directory of an already formatted diskette and names the diskette DATA.

DINIT (076) (c)

DINIT loads the BAM (Block Availability Map) into disk memory. The BAM shows how data is organized on diskette. Normally the BAM automatically loads into disk drive memory when you change a diskette.

There are occasions when the disk drive can confuse two diskettes. This happens when the id characters are the same when you switch from one diskette to another. If this happens, the disk drive assumes that the newly inserted diskette is the same diskette as the old one.

When this happens, and you know that the diskette ids are the same, you can initialize the diskette (load the BAM) with the DINIT command.

Format: DINIT

DRESET (077) (c)

DRESET sets the disk drive into the power-up state, something like resetting the computer, without the disadvantages.

Format: DRESET

5.2 Changing disk drive addresses

The default address of the disk drive is 8. If you work with two disk drives (the C64 allows up to 5 disk drives), the addresses must be different from one another.

The following three commands allow address changes and multiple disk drive operation.

DADRCHANGE (194) (c)

DADRCHANGE allows the change of a disk drive's device number through software (see your 1541 manual for hardware address changes):

Format: DADRCHANGE DN

DN is the new disk drive device number. Values for DN can range from 4 to 15. Other values result in an ILLEGAL QUANTITY ERROR.

The disk drive not planned for an address change must be switched off.

DKDEVNB (195) (c)
DDEVADR (253) (c)

DKDEVNB determines which disk drive should be assigned the following commands. DEDEVNB is followed by the address of the desired device. DDEVADR gives the address of the disk drive.

Format: DKDEVNB DN: ... :DN = DDEVADR

DN is the new disk drive device number. Values for DN can range from 4 to 15. Other values result in an ILLEGAL QUANTITY ERROR.

Example:

This example is in two parts. Type the first program in and save it with DSAVEB"TEST" on drive 8. Do not RUN this program.

```
25 'TYPE THIS PROGRAM IN FIRST AND SAVE IT AS "TEST"'
30 DKDEVNB 9
40 DSAVEB "TEST"
50 SCPRINT DDEVADR
```

Clear your memory with NEW, then type in the next program listing. After you save it, RUN it.

```
5 'TYPE IN, SAVE AND RUN THIS PROGRAM.'
10 DADRCHANGE 9:WAITKEYA
20 DLOADB"TEST"
```

Turn off the disk drive you want kept as device 8. Line 10 changes the device number of the currently switched on disk drive to 9. Turn on the other drive (device 8) and press a key (WAITKEYA waits for a keypress). Line 20 loads the program "TEST" into memory, overwriting the first program. The program now in memory saves itself as "TEST" to device number 9 and the SCPRINT command displays the current device number (9).

NOTE: You only need to change disk drive addresses once with the extra drive turned off. From then on, you can change addresses within the program while the power is on.

DKDEVNB 8:DADRCHANGE 11 changes disk drive 8 to device 11.

5.3 Program mode commands

The commands described in this section work best with BASIC and machine language programs. The first topic is the saving and loading of programs, including machine language. Screen memory, hi-res bitmaps and other data have their own commands for dealing with data.

NOTE: As already explained in Chapter 1, some commands can also access ROM under RAM. This category includes loading and saving machine language programs.

Disk files can be handled by their filetypes (see Section 5.4.4 below).

5.3.1 Saving and verifying programs

DSAVEB	(082)	(c)
DCSAVEB	(084)	(c)

DSAVEB saves a BASIC program from memory to diskette. DCSAVEB deletes a program of the same name from diskette, then saves the program in memory to diskette under that name.

Format: DSAVEB PR$ ... :DCSAVEB PR$

PR$ is the name under which the program is saved. PR$ can be a maximum of 16 characters in length.

DCSAVEB is the equivalent of BASIC 2.0's SAVE"@:NAME". This command has two advantages: First, DCSAVEB deletes and replaces programs with up to 16 characters in the filename (SAVE"@:NAME" allows only 14 characters). Second, DCSAVEB avoids most file errors or data loss.

Examples:

DSAVEB "UTILITY" saves the BASIC program in memory to diskette under the name UTILITY.

DCSAVEB "UTILITY" deletes a file named UTILITY from diskette and saves the program currently in memory to diskette under the name UTILITY.

DSAVEL (197) (c)
DCSAVEL (198) (c)

DSAVEL and DCSAVEL let you save selected program lines to diskette. DCSAVEL deletes the program of the same name from diskette, then saves the program lines in memory to diskette under that name.

Format: DSAVEL PR$ [,[EL]-[LL]]: ... :DCSAVEL PR$ [,[EL]-[LL]]

PR$ is the name under which the BASIC program is saved. PR$ can be up to 16 characters in length.

EL is the first program line to be saved.

LL is the last program line to be saved.

Examples:

DSAVEL "NAME1",10-30 saves program lines 10 to 30 as the file NAME1.

DCSAVEL "NAME2",125 deletes the old file NAME2 from the diskette and saves line 125 to diskette as NAME2.

DSAVEL "NAME3",-10:DSAVEL "NAME4",25- saves the program from the beginning to line 10 to diskette as NAME3. Then lines 25 to the end of the program are saved to diskette as the file NAME4.

NOTE: If you attempt to DSAVEL a line number larger than the highest program number, BeckerBASIC returns an ILLEGAL QUANTITY ERROR.

For example, take a program that has lines numbered 10, 12, 17, 20, 21 and 49:

DSAVEL"NAME",10-30 saves lines 10 to 21 correctly, but no line 30 exists.

DSAVEL"NAME",-60 causes an error, since the number 60 is larger than the maximum line number (49).

DSAVEM (083) (c)
DCSAVEM (085) (c)

DSAVEM and DCSAVEM save machine language programs and all kinds of data to diskette. DCSAVEM deletes a machine language program of the same name from diskette, then saves the program currently in memory to diskette under that name.

Format: DSAVEM PR$, BA, BE: ... :DCSAVEM PR$, BA, BE

PR$ is the filename under which the program in memory is saved. PR$ can be up to 16 characters long.

BA BA is the starting and BE the ending memory locations of the program.
BE Values for these two addresses can range from 0 to 65535.

Examples:

DSAVEM "FILE1",41000,42000 saves the RAM area between locations 41000 and 42000 as the file FILE1.

DCSAVEM "MP1",828,850 deletes the file already on diskette under the name MP1, and saves the memory range from location 828 to location 850 under the same name.

DSAVEM"SCREEN",1024,2023 saves the current screen contents to diskette under the name SCREEN. For better screen saving and loading commands, see SCRDSAVE and SCRDLOAD (Section 3.3).

DVERIFYB (086) (c)

DVERIFYB compares the BASIC program currently in memory with a program stored on diskette. If both programs are identical, the computer responds with OK, otherwise the result is a VERIFY ERROR.

Format: DVERIFYB PR$

PR$ is the name of the program on diskette that you want compared to the program in memory.

Example:

DSAVEB"NAME":DVERIFYB"NAME" saves the program in memory to diskette as NAME, then compares the program in memory with the program NAME on diskette.

DVERIFYM (087) (c)
DVERIFYAM (199) (c)

DVERIFYM compares a machine language program or other data file on diskette with an equivalent program in memory. The starting address of the program in memory is taken as the starting address of the program on diskette.

DVERIFYAM compares a machine language or other program on diskette with a program in memory. The starting address of the program in memory can be assigned.

Both commands result in either OK (both programs are identical) or VERIFY ERROR.

Format: DVERIFYM PR$: ... :DVERIFYAM PR$,BA

PR$ is the name of the program to be compared with the program currently in memory.

BA is the starting memory address at which the machine language program begins in memory.

Examples:

DSAVEM"NAME",48000,48020:DVERIFYM "NAME" saves the memory range from 48000 to 48020 to diskette as NAME, then compares the program on diskette with the code in memory.

DSAVEM "NAME",830,950:DVERIFYAM "NAME",47500 saves the memory range from 830 to 950, and compares it with the memory range starting at address 47500.

5.3.2 Loading programs

DLOADB	(088)	(c)
DRLOADB	(091)	(c)

DLOADB and DRLOADB loads a BASIC program from diskette into memory. DRLOADB automatically starts the program after loading it, so you don't have to type RUN.

Format: DLOADB PR$: ... DRLOADB PR$

PR$ is the name of the file to be loaded from diskette. PR$ can be up to 16 characters in length.

DLOADM	(089)	(c)
DLOADAM	(090)	(c)

DLOADM and DLOADAM load machine language programs or other data files. DLOADM loads the program at the memory address at which it was saved. DLOADAM lets you load the program at any address. Neither command affects the BASIC pointer. The OUT OF MEMORY ERROR you could get by loading machine language in BASIC 2.0 (LOAD "NAME",8,1) doesn't occur with DLOADM and DLOADAM.

Format: DLOADM PR$: ... :DLOADAM PR$,BA

PR$ is the name of the file to be loaded from diskette. PR$ can be a maximum of 16 characters long.

BA gives the load address of the program. Values for BA range from 0 to 65535.

Examples:

DLOADM"NAME1" loads the program NAME1 into memory.

DLOADAM "NAME2",42000 loads the program NAME2 into memory starting at address 42000.

5.3.3 Overlays

When you write larger programs, it may be necessary to break the program up into smaller programs and load the sections as the program executes. The biggest problem here is retaining variable contents, since BASIC normally destroys variables when a new program loads. Overlay commands solve this problem!

DOVERLAYK (092) (c)

DOVERLAYK loads a specified BASIC program into memory at the start of BASIC. All variables from the previous BASIC program are retained. When the new BASIC program finishes loading, it executes immediately. The previous program is deleted from memory when the new program loads.

Format: DOVERLAYK PR$

PR$ is the name of the program to be loaded. PR$ can be a maximum of 16 characters in length.

DOVERLAYW (093) (c)

DOVERLAYW lets you load line numbers into a program already in memory. Identical line numbers in memory are deleted.

DOVERLAYW has a similar function to the PMERGE command (Section 2.1.1). The exception: DOVERLAYW keeps the variables in the original program intact.

What applies to PMERGE also applies to DOVERLAYW: When you use this command within a program, the program being loaded in cannot have line numbers smaller than or equal to the number of the current BASIC line (in which the DOVERLAYW command stands). In such a case, the program may stop with a SYNTAX ERROR message.

Format: DOVERLAYW PR$

PR$ is the name of the program you want loaded. This name is a string up to 16 characters long. NOTE: Strings normally written in the form VR$="TEXT" must be written as VR$="TEXT"+"" so that the string is handled correctly in the loading process. The added +""ensures that the string is copied into the top of string memory. You can also perform this in DATA statements: READ VR$:VR$=VR$+V$+"".

Example:

• First program in memory (P1)

```
10 M$(1) = "HERE'S ":'1ST PART OF MSG'
20 DOVERLAYW "P2":'LOAD IN 2ND PROGRAM NAMED P2'
30 M$(3) = "EXAMPLE OF "+"":'3RD PART OF MESSAGE'
40 :
50 FOR I=1 TO 4:SCPRINT M$(I):NEXT I:'DISPLAY MESSAGE'
```

• Program loaded by P1 (P2)

```
25 M$(2) = "AN ":'2ND PART OF MESSAGE'
35 M$(4) = "DOVERLAYW.":'4TH PART OF MESSAGE'
```

Running P1 results in this message on the screen:

```
HERE'S
AN
EXAMPLE OF
DOVERLAYW.
```

List the program when it's done running. It will look like this:

```
10 M$(1) = "HERE'S ":'1ST PART OF MSG'
20 DOVERLAYW "P2":'LOAD IN 2ND PROGRAM NAMED P2'
25 M$(2) = "AN ":'2ND PART OF MESSAGE'
30 M$(3) = "EXAMPLE OF "+"":'3RD PART OF MESSAGE'
35 M$(4) = "DOVERLAYW.":'4TH PART OF MESSAGE'
40 :
50 FOR I=1 TO 4:SCPRINT M$(I):NEXT I:'DISPLAY MESSAGE'
```

NOTE: When you can't arrange the program so that line numbers don't conflict, then you should use the DLOADPROC command (see Chapter 6). This sets up *procedures* independent of programs whose line numbers will not conflict with the main program.

LDEL (132) (c)

LDEL has a similar function to the PDEL command (Chapter 2). It deletes individual lines or sets of lines from a program. Unlike PDEL, variable contents remain intact.

Format: LDEL Z1 [, Z2-Z3,...]

Z1 is the line to be deleted.

Z2,Z3 is the optional set of lines to be deleted.

You can put as many parameters into LDEL as you can fit into a program line.

Example:

```
10 ...
20 LDEL 50,72-79,100
30 ...
```

The LDEL in line 20 deletes program lines 50,72 to 79 and 100.

5.4 Logical files

The logical file is an efficient way to handle data of all kinds on diskette. Every logical file has a name under which it is stored on diskette. Every logical file has a *logical file number*. This number easily lets you see whether the file is set for reading or writing.

This section lists the essential commands needed for logical file access. They follow the same principles as stated earlier.

5.4.1 Logical file commands

Logical file access consists of three basic actions:

- Open the file
- Read /write the file
- Close the file

When you open a file, the filename and logical file number state the necessary parameters. BeckerBASIC's DGETV and DGETM replace the BASIC 2.0 commands GET# and INPUT# for reading file data. Writing data is performed by the BASIC 2.0 PRINT# command. All read and write errors are signalled according to the logical file number.

DCLOSE closes the file and ends the access. If you wish to re-access the file, it must again be opened by the DOPEN command.

NOTE: You must use the DCLOSE command to close the file; you can't just leave the file open. Also, remember to use the proper secondary addresses when closing and opening files with DCLOSE and DOPEN.

The disk drive system allows a maximum of three open files at one time. If you open a fourth file, a TOO MANY FILES ERROR results. You should also keep in mind that one relative file is equal to two normal files. If you have a relative file open, you can only have one sequential file open as well.

DOPEN (080) (c)

DOPEN opens a file of any type for reading or writing. All filetypes have their own special open commands (more on this below).

Format: KN$="FILENAME,FILETYPE,MF":DOPEN LF,KN$

LF is the logical file number of the file. Values for LF can range from 1 to 127 (you can theoretically use values higher than 127, but it doesn't usually make sense for disk access). The logical file number identifies the file, and has nothing to with the type of file access itself.

KN$ contains the filename. Filenames can be a maximum of 16 characters. It is separated from the mode flag (MF) by a comma.

There are four filetypes available:

- S sequential (SEQ)
- P program file (PRG)
- U user file (USR)
- R relative file (REL)

MF is the mode flag, which states whether the file is open for reading or writing. You have a choice of two letters for MF:

- R Read data
- W Write data

One exception exists when opening a relative file with DOPEN: You omit the mode flag and replace it with the record length in character code form. Another peculiarity stands in opening sequential files: Using A for MF lets you append an existing sequential file to an open file.

Examples:

DOPEN 1,"EX1,S,W" opens EX1 as a sequential file for writing.

DOPEN 2,"EX2,P,R" opens program file EX2 for reading.

DOPEN 4,"EX4,L"+CHR$(82) opens EX4 as a relative file with a record length of 82 characters. You can perform both read and write access on this file.

DOPEN 5,"EX5,S,A" opens sequential file EX5 for appending data to the file previously opened by DOPEN.

FILENUM (252) (f)

FILENUM lists the number of files currently open. Checking this occasionally helps you avoid having more than three files open at a time.

After opening the desired file with the DOPEN command, you can write or read any data in the file, depending upon which mode is active when the file opens. The next two commands are used for reading data.

DGETV (045) (c)

DGETV reads data from any disk file and puts this data into a string variable.

Format: DGETV LF, VR$, LE

LF is the logical file number.

VR$ is the name of the string variable into which the data goes.

LE is the number of characters that should be read from the file. Values for LE can range from 1 to 255.

DGETV has the advantage over BASIC 2.0's INPUT# in that it can handle up to 255 characters at a time.

Examples:

DGETV 7,EG$,23 reads 23 characters from logical file 7, and places these characters into string variable EG$.

FOR I=1 TO 3:DGETV 2,A$(I),12:NEXT I reads 12 bytes three times from logical file 2 and inserts the contents into variable array A$(1), A$(2) and A$(3).

DGETM (046) (c)

DGETM reads data from any disk file, and places this data in any area of memory.

Format: DGETM LF, SA, LE

LF is the logical file number of the corresponding file.

SA is the address of the first memory location of the data read. Values for SA range from 0 to 65535.

LE sets the number of bytes to be read from the file. Values for LE range from 1 to 255.

The use of DGETM instead of DGETV is useful when the data must be transferred directly to the screen, and variable contents must stay free (see Chapter 4 for memory access commands).

Examples:

DGETM 2,42000,52 reads 52 characters from logical file 2, and places the data in the computer starting at memory location 42000.

```
...
110 DGETM 5,48000,120:'READ DATA'
120 MGETV A$,50,48000:
130 MGETV B$,20,48050:
140 MGETV C$,50,48070:
...
```

Line 110 reads the data and places it in memory starting at address 48000. Lines 120 to 140 put the data in memory location 48000 and place it in string variables A$, B$ and C$.

EOF (238) (f)

EOF helps you determine the end of the current disk file.

Format: FL = EOF

When the end of file is reached, FL = -1 (logical true); otherwise FL is equal to 0 (logical false).

Example:

```
...
110 Z=0:REPEAT:DGETM 3,42000+Z,1:Z=Z+1:UNTIL EOF
```

This short routine reads the data from logical file 3 until it reaches the end of the file. The file goes into memory starting at memory address 42000 (more on the REPEAT/UNTIL construct in Chapter 6).

DCLOSE (081) (c)

DCLOSE closes a logical file, signalling the computer and disk drive that the file access is finished.

Format: DCLOSE LF

LF is the logical file number of the file accessed.

Example:

```
10 OPEN 6,"DATA,S,R":'OPEN SEQ. FILE FOR READING'
20 DGETV 6,E$,21:'READ DATA'
30 DCLOSE 6:'CLOSE FILE'
```

This routine reads the data from a sequential file and places it in E$.

5.4.2 Sequential file commands

The following three commands simplify sequential file access.

DSQOPEN (094) (c)

DSQOPEN is designed for opening sequential files. The simplest form uses the logical file number and the corresponding filename.

Format: KN$="FILENAME [,LS]":DSQOPEN LF,KN$

LF is the desired logical file number. Legal values for LF range from 1 to 127.

KN$ is the filename. KN$ is a string containing up to 16 characters.

LS is the mode flag. This flag can be one of 4 characters:

- R read data
- W write data
- A append data to existing file
- M open a file not previously closed by DCLOSE (merge)

Omitting the mode flag defaults the file to read status (R).

Two mode flags are new: A and M.

A If you try writing to an existing sequential file with new data using the W mode flag, the error message FILE EXISTS results. You can add to this file by opening it with the mode flag A. All data sent through PRINT# is appended to the existing file.

M As already mentioned, every logical file must be closed with DCLOSE after a session. If you forget to close a file, the next time you try to read it, you'll get a WRITE FILE OPEN error. One possibility for opening a saved file is with the M mode flag. Once the file is open, read the entire file, write the data into a new file (remember to close it) and delete the old file with DSCRATCH.

Examples:

DSQOPEN 5,"DATA" opens the sequential file DATA for reading. The logical file number is 5.

DSQOPEN 12,"DAT2,W" opens the sequential file DAT2 for writing.

DSQOPEN 1,"DATF,M" opens the improperly closed sequential file DATF for reading.

NOTE: Close sequential files with DCLOSE.

DSQCONCAT (095) (c)

DSQCONCAT allows multiple sequential files to be added to a new file (maximum 4 files).

Format: KN$="NF=F1,F2,...":DSQCONCAT KN$

F1,... are the names of sequential files added to the new file.

NF is the name of the new file.

KN$ is the string containing the data about NF, F1, etc. This string can be up to 38 characters in length.

Examples:

If you want to add a file to an existing file, you can do the following:

```
10 DSQCONCAT "ZW=F1,F2"
20 DSCRATCH"F1"
30 DRENAME "F1=ZW"
```

This program appends file F2 to file F1.

DSQCONCAT "DATG=DAT1,DAT2,DAT3,DAT4" combines files DAT1 through DAT4 into the new file DATG. DAT1 through DAT4 remain as separate files, as well as the combined file DATG.

5.4.3 Relative file commands

With the help of these three commands, you can easily handle relative files.

DRLOPEN (096) (c)

DRLOPEN is for opening relative files for writing or reading. There is no differentiation between reading and writing with relative files.

Format: DRLOPEN LF, FN$, RL

LF is the desired logical file number (1-127).

FN$ is the desired or already existing filename (maximum of 16 characters).

RL is the record length. Relative files are divided into records, and all records have the same length. This parameter must be given on every file opening, regardless of whether the file is new or existing. Values for RL can range between 1 and 254 bytes.

NOTE: Once you set a record length on initially opening a file, the record length cannot be changed. Trying to re-open a file using a different record length results in a RECORD NOT PRESENT ERROR.

When the input is sent with PRINT#1 and concluded with <RETURN> (e.g., PRINT#1,A$), you must allow 1 byte for the CHR$(13) (<RETURN> key) within each record. A 50-byte record can only contain 49 characters plus <RETURN>.

Thanks to the special BeckerBASIC reading commands, the record length doesn't include the <RETURN> key. This is something like adding a semicolon to the end of the PRINT# command (e.g., PRINT#1, A$;).

Examples:

DRLOPEN 2,"DATA",70 opens a relative file named DATA with a record length of 70 bytes and a logical file number of 2.

DRLOPEN 7,"LAYOUT",254 opens a relative file named LAYOUT with the maximum record length of 254 bytes.

DRLCLOSE (200) (c)

DRLCLOSE is the close command for relative files.

Format: DRLCLOSE LF

LF is the logical file number used with DRLOPEN.

Example:

DRLCLOSE 7 closes the relative file assigned logical file number 7. One similarity between DCLOSE and DRLCLOSE: When a disk error occurs during the time a file is open, you must close the corresponding file.

DRLRECORD (097) (c)

All data records in a relative file are accessed by record numbers, with values from 1 to 65535. Tò access a record (i.e., read from it or write to it), you must set the computer to the record's position.

Format: DRLRECORD LF, RN, RP

LF is the logical file number of the file currently being accessed.

RN is the record number you want. Values for RN can range from 1 to 65535.

RP allows you to move to a position within the record. Legal values for this can range from 1 (first byte of the record) to 254 (last byte of the record).

There are two things to keep in mind about positioning:

1) When writing a record, RP must start out set to 1. Data records are sent from that point in one group through the PRINT# command. When a position is found that is larger than the last data record of the corresponding file, the result is a RECORD NOT PRESENT ERROR. However, the next write access to the record with PRINT# executes correctly. The message RECORD NOT PRESENT signals that you have gone past the previous end of the file.

2) A write access to a record fills all data records with lower numbers that haven't been written to yet with CHR$(255). For example, you define a new relative file with DRLOPEN 7,"DATA",50. Using DRLRECORD 3,70,1:PRINT#3,RD$ writes to record 70. Records 1 to 69 are written with CHR$(255), and can be written to later on.

To avoid unnecessary waiting time during file access, if you know the length of the file, you can move to the last record position and fill in the entire file with CHR$(255). For instance, a program to fill in a relative file containing 200 records and record length of 72 bytes can look like this:

```
5 'THIS PROGRAM WRITES DATA TO THE 200TH RECORD OF A REL FILE'
10 DRLOPEN 1,"DATA",72:'OPEN FILE'
20 DRLRECORD 1,200,1:'POSITION TO 200TH RECORD'
30 PRINT#1,CHR$(255):'WRITE RECORD'
40 DRLCLOSE 1:'CLOSE FILE'
```

Now for a complete example of relative file handling using the simple file handling commands included in BeckerBASIC:

```
5 'RELFILE MGR.BECKERBAS'
10 LF=1:'LOGICAL FILE NUMBER'
20 DN$="DATA":'FILENAME'
30 RL=20:'RECORD LENGTH'
40 DRLOPEN LF,DN$,RL:'OPEN FILE'
50 :
90 CLS
100 SCPRINT "READ OR WRITE RECORD (R/W)?";
110 KBGETV WL$,1,"WR":'SELECT W OR R'
120 GOSUB WL$:'AND CALL SUBROUTINE'
130 :
140 SCPRINT:INPUT "MORE? (Y/N)";W$:'CONTINUE?'
150 IF W$="Y" THEN POPIF:GOTO100:ENDIF
160 DRLCLOSE LF:'NO, CLOSE FILE'
170 END:'END PROGRAM'
180 :
190 :
500 "R":'READ RECORD'
510 SCPRINT:INPUT"RECORD NUMBER";RN
520 DRLRECORD LF,RN,1:'POSITION TO RECORD'
530 DGETV LF,EG$,RL:'READ RECORD'
540 SCPRINT EG$:'DISPLAY ON THE SCREEN'
550 RETURN
560 :
570 :
600 "W":'WRITE RECORD'
610 SCPRINT:INPUT"RECORD NUMBER: ";RN
620 DRLRECORD LF,RN,1:'MOVE TO RECORD'
630 SCPRINT"YOUR INPUT:";
640 KBGETV EG$,RL:'GET DATA FROM KYBD'
650 PRINT#LF,EG$;:'AND SEND IT'
660 RETURN
```

5.4.4 Opening user and program files

DUSOPEN (098) (c)

DUSOPEN opens a user file (files containing a USR identifier in their directory listings).

Format: KN$="FILENAME [,LS]":DUSOPEN LF,KN$

LF is the desired logical file number. Legal values for LF can range from 1 to 127.

KN$ contains the 16-character filename, as well as LS.

LS is the optional mode flag. If LS is W, then the file opens for writing; if LS is R, then the file opens for reading. If you omit the mode flag, the file opens for reading (R).

Examples:

DUSOPEN 3,"NAM1" opens the user file NAM1 for reading data.

DUSOPEN 5,"NAM2,W" opens user file NAM2 for writing.

DPGOPEN (196) (c)

DPGOPEN opens program files. This lets you load and edit a program byte for byte.

Format: KN$="FILENAME [,LS]":DPGOPEN LF,KN$

LF is the logical file number. Legal values for LF range from 1 to 127.

KN$ contains the 16-character filename, as well as LS.

LS is the optional mode flag. If LS is W, then the file opens for writing; if LS is R, then the file opens for reading. If you omit the mode flag, the file opens for reading (R).

Examples:

DPGOPEN 1,"PRG1" opens the program file PRG1 for reading.

DPGOPEN 7,"PRG2,W" opens the program file PRG2 for writing.

Files opened by DUSOPEN or DPGOPEN may be closed using DCLOSE.

5.5 Direct diskette access

Diskettes store data in blocks of 256 bytes each. A direct access file allows you to access (read or write) individual blocks of data. This means that you can easily create your own data structures based upon program, sequential and relative files. These commands allow simple manipulation of available files, or even the directory.

CAUTION: Even though direct access gives great flexibility in disk access, remember that direct access can also turn little errors into big ones! For example, one badly written data block can destroy an entire sequential file.

If you want to design your own file structures, you should use a newly formatted diskette which contains no programs, relative or sequential files. Or at the very least, use a backup copy of the diskette you want to read from or write to.

DDAOPEN (099) (c)

DDAOPEN is designed for opening a direct access file. Before describing the format of this command, you need some general background about the organization of a direct access file.

All data read from a disk data block is first stored in buffer memory within the disk drive's memory. From there you read the data with the commands DGETV, DGETM, etc.

By the same token, data written to a data block is stored in this buffer, then transferred to diskette using a special command.

The disk drive has a total of five buffers available, each identified by the numbers 0 to 4:

```
NUMBER  CORRESPONDING MEMORY RANGE
0               768-1023
1               1024-1279
2               1280-1535
3               1536-1791
4               1792-2047
```

DDAOPEN opens the specified buffer for file access.

Format: DDAOPEN LF [,PN]

LF is the logical file number. Legal values for LF can range from 1 to 127.

PN is the buffer number, chosen from the list above. Buffer selection has nothing to do with the later transfer of the file, so you can omit the PN parameter from DDAOPEN if you don't care which buffer is used.

The number of the buffer selected can be read after using the DDAOPEN command (e.g., with DGETV LF,P$,1:PN=ASC(P$)). If you give an illegal number for PN, or the corresponding buffer is being used, the DOS responds with a NO CHANNEL error.

In most cases, all you need to do is open a direct access file. You can have a maximum of four of these files open at a time. Watch out for opening different filetypes at once (SEQ, PRG, etc.). You can have a maximum of two relative files open at a time. If you overstep the maximum allowable number of files, you'll get a TOO MANY FILES ERROR error messages.

Examples:

DDAOPEN 2 opens a direct access file with a logical file number of 2.

DDAOPEN 5,3 sets up a direct access file with a logical file number of 5 in disk buffer 3.

DDAOPEN 1,0:DDAOPEN 2,2:DDAOPEN 3,0 leads to a NO CHANNEL ERROR after the third DDAOPEN command, since buffer 0 is already open.

DDAREADBL (101) (c)

DDAREADBL reads the desired track and sector (data block) from diskette into the direct access file's buffer.

Format: DDAREADBL LF, TR, SC

LF is the logical file number. Legal values for LF can range from 1 to 127.

TR,SC are the track (TR) and sector (SC) of the desired data block. Use the values for track and sector in the table below:

TRACK	SECTOR
0 - 17	00 - 20
18 - 24	00 - 18
25 - 30	00 - 17
31 - 35	00 - 16

All other values or combinations of values result in an ILLEGAL TRACK OR SECTOR ERROR.

Examples:

DDAREADBL 3,18,0 loads the data block at track 18, sector 0 (the first directory block) into the buffer.

DDAREADBL 2,20,20 causes an ILLEGAL TRACK OR SECTOR ERROR, since track 20 has no sector 20.

DDAREADBL 5,1,7:DDAREADBL 2,35,3 gets the data block in track 1, sector 7 and places it in the buffer assigned to the direct access file with logical file number 5. Then data from track 35, sector 3 loads into the buffer assigned logical file number 2.

After the data block loads into the buffer, you can read the data with the resident commands (DGETV, DGETM, etc.). In addition, you can set a buffer pointer to a memory location within the buffer, for reading or writing the data.

DDAPOINT (100) (c)

DDAPOINT sets the buffer pointer to a memory location within the buffer.

Format: DDAPOINT LF,PS

LF is the logical file number. Legal values for LF can range from 1 to 127.

PS is the desired position at which the pointer should be set. Legal values for PS can range from 0 (the first byte of the buffer) to 255 (the last byte of the buffer).

Examples:

DDAPOINT 3,27 positions the pointer to the 28th byte of memory assigned by logical file number 3.

DDAPOINT 7,255 puts the pointer on the last byte of the buffer controlled by logical file number 7.

DDAREADBL 2,18,7:DDAPOINT 2,20:DGETV 2,G$,45 reads bytes 21 to 66 of track 18, sector 7 and places the bytes into the variable G$.

DDAWRITEBL (102) (c)

DDAWRITEBL writes the data block in the specified buffer to diskette.

Format: DDAWRITEBL LF, TR, SC

LF is the logical file number. Legal values for LF can range from 1 to 127.

TR,SC are the track (TR) and sector (SC) of the desired data block. Use the values for track and sector in the table below:

TRACK	SECTOR
0 - 17	00 - 20
18 - 24	00 - 18
25 - 30	00 - 17
31 - 35	00 - 16

All other values or combinations of values result in an ILLEGAL TRACK OR SECTOR ERROR.

NOTE: To store the current data block, you can send it to diskette with PRINT#.

Examples:

DDAWRITEBL 2,1,0 writes the contents of the buffer assigned logical file number 2 to the data block starting at track 1, sector 0.

DDAPOINT 5,122:PRINT#5,"DATA";:DDAWRITEBL 5, 25, 7 sets up the direct file access to the data buffer assigned logical file number 5, starting at the 123rd byte. Then the buffer contents are saved to track 25, sector 7.

```
5 'DISK RENAMER CHANGES YOUR DISK NAME THROUGH DIRECT ACCESS'
10 INPUT"NEW  DISKETTE NAME:";DN$
20 DN$=LEFT$(DN$,16):'KEEP NAME DOWN TO SIXTEEN CHARACTERS'
30 :
40 DDAOPEN 1:'OPEN DIRECT ACCESS FILE'
50 DDAREADBL 1,18,0:'LOAD FIRST DIRECTORY BLOCK'
60 DDAPOINT 1,144:'POSITION TO DISK NAME'
70 PRINT#1,DN$;:'WRITE NEW NAME TO BUFFER'
80 DDAWRITEBL 1,18,0:'WRITE DATA BACK TO DIRECTORY BLOCK'
90 DCLOSE 1:'CLOSE FILE'
```

Normally, you name a diskette once--when you format a diskette. The program above lets you change the name of your diskette, without any loss of data, anytime you want. NOTE: *Type this program in carefully.*

DDABLALLOC (103) (c)
DDABLFREE (104) (c)

Now that you know the essentials of direct access and data control, BeckerBASIC has two DOS commands which allocate and free up diskette memory.

DDABLALLOC allocates data in a specific track and sector on the BAM (Block Availability Map). DDABLFREE frees memory in a specific track and sector.

Format: DDABLALLOC TR, SC: ... :DDABLFREE TR, SC

TR,SC are the track (TR) and sector (SC) of the desired data block. Use the values for track and sector in the table below:

TRACK	SECTOR
0 - 17	00 - 20
18 - 24	00 - 18
25 - 30	00 - 17
31 - 35	00 - 16

All other values or combinations of values result in an ILLEGAL TRACK OR SECTOR ERROR.

Attempts to re-allocate diskette memory already allocated result in the error message NO BLOCK. The third and fourth parameters of this error list the next available data block (the next highest track and sector).

Examples:

DDABLALLOC 12,19 allocates the data block at track 12, sector 19 in the BAM.

DDABLFREE 2,7 frees up memory in the block at track 2, sector 7.

DDABLALLOC 18,0 causes the error message 65,NO BLOCK,18,10. This means that the data block at track 18, sector 0 are already allocated with the first directory entry. The next free block is in track 18, sector 10.

The following command sequence below lets you isolate the track and sector number of the next available data block from the error message. The error message is stored in the variable DS$ for later retrieval (e.g., with DSTATUS DS$):

```
10 IF VAL(DS$)=65 THEN SP=VAL(MID$(DS$,13,2))
20 SK=VAL(MID$(DS$,16,2)):ENDIF
30 ...
```

If no free data block exists (i.e., there is no next available block), TR and SC are both set to zero.

5.6 Disk memory access

The 1541 disk drive has its own disk operating system (DOS); which means it can perform its disk operations without the computer's support. In addition to the DOS, which is stored in ROM, there are two kilobytes of RAM allocated for working memory and buffer memory. The following commands let you read disk drive memory (both RAM and ROM), write to disk drive RAM, and execute your own machine language programs from within disk drive RAM.

DMYPEEK (239) (f)

DMYPEEK reads individual bytes from disk memory.

Format: VL = DMYPEEK (AD)

AD is the memory address whose contents are placed in the variable VL. Values for AD can range from 0 to 65535.

Example:

BL=DMYPEEK(762)+256*DMYPEEK(764) assigns the number of free blocks on the diskette currently in the disk drive to the variable BL.

DMYREADV (202) (f)

DMYREADV reads up to 255 bytes of a memory segment from disk memory into a string variable.

Format: DMYREADV VR$, LE, BA

VR$ is the name of the string variable assigned to the data.

LE,BA are the address of the first memory location to be read and the length of the data being read, computed from BA. Values for BA range from 0 to 65535; values for LE range from 0 to 255.

Example:

DMYREADV K$,16,1936 reads the name of the disk currently in the drive and places it in the variable K$. Characters following the filename are filled in with <SHIFT><SPACE>. This can be used to check if the correct diskette is currently in the drive.

DMYREADM (201) (c)

DMYREADM reads up to 255 bytes of a memory segment selected from disk memory and places it in a range of memory in the computer.

Format: DMYREADM RA, LE, BA

RA is the address of the computer's memory at which the data should be placed. Values for RA range from 0 to 65535.

LE,BA are the address of the first memory location to be read and the length of the data being read, computed from BA. Values for BA range from 0 to 65535; values for LE range from 0 to 255.

Example:

DMYREADM 42000,37,725 reads the last error message sent by the disk drive from address 725 to 761 (error message buffer memory) and places this segment in the computer's memory starting at memory location 42000.

DMYPOKE (106) (c)

DMYPOKE writes individual values to disk drive RAM.

Format: DMYPOKE BA, WT

BA BA is the address and WT is the value placed into the address. Values
WT for WT range from 0 to 255; values for ba range from 0 to 65535 (NOTE: Not all values for BA are effective, see your C64 Programmer's Reference Guide, or The Anatomy of the 1541 Disk Drive from Abacus for memory locations).

Example:

DMYPOKE 106,10 changes the number of disk retries before it displays an error message to 10 accesses. The normal number of tries before an error message appears is 5.

DMYWRITEV (108) (c)

DMYWRITE writes a string up to 34 characters long to the given disk memory range.

Format: DMYWRITEV FA, SD$

FA is the address at which the data is written in disk memory. Values for FA range from 0 to 65535.

SD$ is the string to be sent to disk memory. Values for SD$ range from 1 to 34 characters. Strings longer than 34 characters result in a STRING TOO LONG ERROR.

Example:

DMYWRITE 1024,S$:DMYWRITEV 1024+LEN(S$),D$ places the combined contents of S$ and D$ to disk buffer 1, starting at memory address 1024.

DMYWRITEM (107) (c)

DMYWRITEM writes up to 34 bytes of consecutive computer memory into a given area of disk memory.

Format: DMYWRITEM FA, RA, LE

FA is the address of disk memory at which the data starts. Values for FA range from 0 to 65535.

RA,LE are the starting address of computer memory of the data sent to disk memory (RA) and the length of the data (LE) starting at RA. Values for RA range from 0 to 65535. Values for LE range from 1 to 34.

Example:

DMYWRITEM 1536,45000,20 places the memory from location 45000 to location 45019 into disk data buffer 3 (location 1536).

When you want to execute machine language commands stored in disk memory or on diskette, you can start these from the computer using the following commands.

DMYEXEC (109) (c)

DMYEXEC runs a machine language program found in disk RAM or ROM starting at the specified memory address. The machine language program must be ended with an RTS (Return from Subroutine).

Format: DMYEXEC SA

SA is the starting address of the machine program set for execution. Values for SA can range from 0 to 65535.

Example:

DMYEXEC 49597 branches to disk memory and deletes the command string buffer in the disk drive.

DDABLEXEC (105) (c)

DDABLEXEC loads the contents of the given data block into the predetermined direct access file disk memory (see Section 5.5). The contents are then executed as a machine language program. Like DMYEXEC, the machine language program must be concluded with an RTS (Return from Subroutine).

Format: DDABLEXEC LF, TR, SC

LF is the logical file number set in DDAOPEN (see Section 5.5). Legal values for LF can range from 1 to 127.

TR,SC are the track (TR) and sector (SC) of the desired data block. Use the values in the table below for track and sector:

TRACK	SECTOR
0 - 17	00 - 20
18 - 24	00 - 18
25 -30	00 - 17
31 -35	00 - 16

All other values or combinations of values result in an ILLEGAL TRACK OR SECTOR ERROR.

Example:

DDABLEXEC 3,14,19 loads the machine language program at track 14, sector 9 to the data buffer assigned logical file number 3, and starts the program.

6. Structured programming

Most large BASIC programs are unreadable. That is, their listings are difficult to read for style or program flow. This chapter discusses the structured programming commands of BeckerBASIC.

Along with an extended IF command and a special command for multiple-choice (SELECT), BeckerBASIC offers you three new loop types which allow more flexible programming than FOR/NEXT: WHILE/DO/ENDDO, REPEAT/UNTIL and LOOP/LPEXITIF/ENDLOOP.

With the exception of SELECT/ENDSEL, this chapter lists six preset constructs for simpler nested loop programming. For an introduction to nesting, here's an example written in BASIC 2.0.

```
10 FOR Z1=1 TO 10
20 : FOR Z2=1 TO 10
30 :   FOR Z3=1 TO 10
40 : ...
50 : ...
60 :   NEXT Z3
70 : NEXT Z2
80 NEXT Z1
```

This routine consists of three nested FOR/NEXT loops. The innermost loop executes lines 30 to 60. The second nested level runs lines 20 to 70, and the topmost level from line 10 to 80.

As you can see from the above example, the program becomes much more readable when you indent each loop level. The colons at the beginning of lines 20 to 70 are necessary so the interpreter ignores the spaces following them.

This chapter introduces new programming techniques and commands in BeckerBASIC, as well as demonstration programs.

6.1 Comments

BeckerBASIC has two extended versions of the BASIC 2.0 REM command. These versions use the apostrophe (') and quotation mark (").

You must place commentary between the ' or ". There must be a colon before or after the commentary. Also, you cannot mix the two characters as comment markers (" and ', or ' and ").

The major advantage of these comment markers over REM is the flexibility of comments: You can place comments between commands, instead of at the end of a command line, or on a separate line.

Examples:

```
RIGHT: 100 A=1:B=3:'DISPLAY A':PRINT A:'DISPLAY B':PRINT B:'READY'

WRONG: 100 A=1:B=3:'DISPLAY A'PRINT A 'DISPLAY B' PRINT B
'READY':'NO COLONS'
WRONG: 100 A=1:B=3:'DISPLAY A :PRINT A 'DISPLAY B' PRINT B
'READY':'FIRST COMMENT NOT ENCLOSED'
WRONG: 100 A=1:B=3:"DISPLAY A':PRINT A 'DISPLAY B' PRINT B
'READY':'FIRST COMMENT OPENS WITH A QUOTE, CLOSES WITH AN
APOSTROPHE'
```

6.2 Labels and calculated line numbers

GOTO	(001)	(c)
GOSUB	(002)	(c)
RUN	(005)	(c)
RESTORE	(003)	(c)
ON	(174)	(c)

Throughout the BeckerBASIC program disk you'll find most of the comments typed in between apostrophes. The quotation mark can be used as a comment marker, but it is also used for defining labels. The jump commands GOTO, GOSUB, etc. only function in BASIC 2.0 through the use of constants (e.g., GOTO 100, GOSUB 350). Commands like GOTO 2*A+B or GOTO "OUTPUT" don't run in BASIC 2.0. The first of these two (GOTO 2*A+B) handles the branch to a *calculated line number*. The second (GOTO "OUTPUT") looks for a *label*. Both these items are executable in BeckerBASIC thanks to the GOTO, GOSUB, RUN, RESTORE and ON commands.

Note that RESTORE also sets positions for DATA lines. While the DATA pointer of BASIC 2.0 moves only to the first DATA statement, BeckerBASIC lets you position the DATA pointer to any DATA statement, and any section of a program. This allows the use of calculated line numbers and labels (e.g., RESTORE 12+A or RESTORE "BLOCK").

The ON command is an extended version of BASIC 2.0's ON command (ON GOTO/ON GOSUB). Constants, calculated line numbers and labels can be combined here (e.g., ON A GOTO 100, "MARK1",2*CR+7).

Calculated line numbers may use any mathematical expressions. You can even use GOTO SIN(A) or GOSUB SQR(COS(B)). The only limit is that you stay within the legal values (from 0 to 63999). If you go beyond these values, the computer returns an ILLEGAL QUANTITY ERROR (values lower than 0) or a SYNTAX ERROR (values higher than 63999). Results containing decimal numbers automatically round off to integers (BeckerBASIC removes the decimal places).

Any alphanumeric expression can be used as a label (e.g., GOTO MID$(A$,1,2), GOSUB "MARKIT" or RESTORE A$+B$.

Three conditions are required for labels:

1) The label must be enclosed in quotation marks (").

2) The label must begin a program line.

3) A colon separates the label from the rest of the program line.

Examples:

```
5 'LABEL DEMO'
10 A=10*B+7:GOSUB "OUTPUT"
50 PRINT"THIS IS THE MAIN PROGRAM, AND SHOULD APPEAR AFTER THE";
60 PRINT"OUTPUT  SUBROUTINE.":PRINT"       -----"
70 END
100 "OUTPUT":PRINT"THIS IS THE ";CHR$(34);"OUTPUT";CHR$(34);
110 PRINT"SUBROUTINE AND SHOULD APPEAR FIRST. A=";A:PRINT:RETURN

5 'RESTORE DEMO'
10 A$(1)="BLOCK1":A$(2)="BLOCK2":A$(3)="BLOCK3"
20 INPUT"PLEASE SELECT A BLOCK NUMBER (1-3) AND  PRESS <RETURN>";BN
30 RESTORE A$(BN):'MOVE TO DESIRED BLOCK'
40 :
50 READ DA$
60 PRINT DA$
1000 "BLOCK1":DATA "DOG"
1010 DATA
1100 "BLOCK2":DATA "CAT"
1110 DATA
1200 "BLOCK3":DATA "MOUSE"
1210 DATA

1 'CALCULATED LINE DEMO'
5 PRINT"MAIN MENU"
10 PRINT"MODULE 1: 1"
20 PRINT"MODULE 2: 2"
30 :
40 INPUT"PLEASE SELECT 1 OR 2,PRESS <RETURN>:";YC
50 IF YC<1 OR YC>2 THEN END
60 'JUMP TO LINE 1000 (YC=1) OR LINE 2000 (YC=2)'
70 GOTO YC*1000
1000 PRINT"YOU CHOSE OPTION 1."
2000 PRINT"YOU CHOSE OPTION 2."
```

6.3 Branch structures

IF	(110)	(c)
THEN	(111)	(c)
ELSE	(112)	(c)
ENDIF	(113)	(c)

These control structures are extensions of BASIC 2.0's IF/THEN.

Format:
```
10 IF [CONDITION] THEN 'DO THIS'
20 [ELSE]'OTHERWISE, TRY ALTERNATE'
30 ENDIF
```

If the condition following the IF is fulfilled, the program executes the THEN. If the condition is unfulfilled at IF, the program looks for the ELSE and executes the section stated at ELSE. When ELSE is omitted, the program continues after the ENDIF.

As you can see from the format, the IF/THEN/ELSE/ENDIF can be used over several program lines.

There are a few points to keep in mind when working with the BeckerBASIC version of IF/THEN :

1) BeckerBASIC requires the ENDIF (i.e., it must be placed at the end of every IF sequence). If you leave out ENDIF, the interpreter usually responds with a CONSTRUCT NOT CLOSED ERROR.

2) A colon must precede the ELSE nad ENDIF instructions, unless one of these instructions is at the beginning of a program line.

3) There should be no line number immediately after THEN or ELSE in the same line. If you must do this, the POPIF command must be used. For example, the BeckerBASIC equivalent of IF A=1 THEN 1000 is:

```
IF A=1 THEN POPIF:GOTO 1000:ENDIF
```

If the interpreter finds an ELSE of ENDIF without a corresponding IF, the result is an ELSE/ENDIF WITHOUT IF ERROR. A THEN without an IF returns a SYNTAX ERROR.

Examples:

IF A=1 THEN B=0:ELSE B=1:ENDIF makes variable B equal to 0 if variable A equals 1; otherwise, variable B equals one.

IF WT=TZ THEN WB=7:ENDIF exits through ENDIF in either case, since there is no ELSE.

```
100 IF W$=MID$(AB$,4,2) THEN SCPRINT W$
110 ELSE W$="":ENDIF
```

If the condition is fulfilled, then the string W$ appears on the screen, otherwise the variable W$ becomes a null string.

```
100 IF AF$="I" THEN
110 GOSUB "INPUT"
120 ELSE GOSUB "OUTPUT"
130 ENDIF
```

This can improve the readability of a program (note the GOSUB in line 110).

IF/ENDIF constructs can be nested. The maximum nesting depth can theoretically be 255 levels.

LEVELIF (264) (c)

LEVELIF returns the current nesting depth of IF/THEN commands.

Format: VT=LEVELIF

VT can have values ranging from 0 (no nesting) to 255 (maximum nesting depth).

POPIF (208) (c)

You can exit a loop level at any time using the POPIF command. POPIF simply resets the pointer to the next nesting level up. Before or after POPIF, there must be a loop jump (e.g., a GOTO).

Format: POPIF

SELECT (122) (c)
CASE (123) (c)
OTHER (124) (c)
ENDSEL (125) (c)

SELECT/ENDSEL is basically an extended and easily modified version of the IF/ENDIF structure.

Format:
```
10 SELECT AW
20 CASE W1, ...:
30 CASE W2, W3, ...:
40 ...
50 OTHER ...
60 ENDSEL
```

AW is the numerical expression used by the SELECT command in line 10. Values for AW range from 0 to 255. Values beyond this range result in an ILLEGAL QUANTITY ERROR.

The value in AW determines the CASE command branched to by SELECT. Individual CASE statements can theoretically contain as many values as you can fit on one program line (the total number of CASE commands is limitless).

If one of the compared values goes over AW, the program executes the line following the highest CASE command. If the interpreter finds a new CASE, the program looks for ENDSEL before it continues on. If no CASE value matches AW, the command(s) listed following OTHER executes.

ENDSEL must conclude the SELECT area. OTHER is an optional command.

NOTE: CASE, OTHER and ENDSEL must be found by the interpreter at the beginning of a program line. Indentation and leading colons are not allowed.

Examples:

```
5 INPUT"TYPE A NUMBER - 1,4,7 OR 19";BE
10 SELECT BE
20 CASE 1:BE=BE*2
30 CASE 7:BE=BE-3
40 CASE 19:BE=9/BE
50 CASE 4:BE=BE+21
60 ENDSEL
70 PRINT BE
```

If BE is equal to 1,7,19 or 4, then the program branches to the appropriate CASE command's equation. The program ends with the ENDSEL command (line 60).

```
5 A=1:INPUT"NUMBER";CW
10 SELECT A*CW+7
20 CASE 2,4,7,9,117:GOSUB"SUBROUTINE1"
30 CASE 1,18,22:GOSUB"SUBROUTINE2"
40 OTHER GOSUB"SUBROUTINE3"
50 ENDSEL
60 END
70 "SUBROUTINE1":PRINT"THIS IS SUBROUTINE1":RETURN
80 "SUBROUTINE2":PRINT"THIS IS SUBROUTINE2":RETURN
90 "SUBROUTINE3":PRINT"THIS IS SUBROUTINE3":RETURN
```

The result of the equation A*CW+7 moves the program to the different subroutines. A result of 2,4,7,9 or 117 branches to SUBROUTINE1. A result of 1,18 or 22 branches to SUBROUTINE2. Any other result branches to SUBROUTINE3.

```
5 INPUT"NUMBER";WB
10 SELECT WB
20 CASE 1,3,5:A=1
30 CASE 2,4,6:A=2
40 ENDSEL
50 PRINT"A= ";A
```

This program can be simulated with an IF/ENDIF construct:

```
10 IF (WB=1 OR WB=2) OR (WB=5) THEN A=1:ENDIF
```

```
20 IF (WB=2 OR WB=4) OR (WB=6) THEN A=2:ENDIF
```

As you can see, this version is much harder to follow than the CASE/SELECT version. SELECT may not necessarily be the most useful construct when working with multiple conditions.

```
10 INPUT"NUMBER";KN
20 :
30 SELECT KN
40 CASE 1:WT$="SUNDAY"
50 CASE 2:WT$="MONDAY"
60 CASE 3:WT$="TUESDAY"
70 CASE 4:WT$="WEDNESDAY"
80 CASE 5:WT$="THURSDAY"
90 CASE 6:WT$="FRIDAY"
100 CASE 7:WT$="SATURDAY"
110 OTHER SCPRINT"BAD NUMBER. TRY AGAIN."
120 ENDSEL
130 SCPRINT" "WT$
```

This routine reads the number you input and puts the weekday into the variable WT$.

6.4 Loop structures

BeckerBASIC offers three loop types in addition to the BASIC 2.0 FOR/NEXT loop: WHILE/DO/ENDDO, REPEAT/UNTIL and LOOP/LPEXITIF/ ENDLOOP. All three types differ from each other in the time at which conditions execute. WHILE takes control at the beginning of the loop; REPEAT waits until the end of the loop. LOOP works at any point in the loop.

WHILE	(114)	(c)
DO	(115)	(c)
ENDDO	(116)	(c)

A WHILE loop performs its task as long as a condition remains true and the commands within the loop do not change (a FOR/NEXT construction runs only once in any case).

Format:
```
10 WHILE ... [CONDITION] DO
20 ...
30 ENDDO
```

Like IF/THEN, WHILE/DO operates with any condition. As long as this condition is true, the program commands between DO and ENDDO are executed.

When the program encounters an ENDDO, it checks the current loop condition between WHILE and DO. If this is still true, the commands between WHILE and DO continue execution. On false conditions, the program continues at the point following ENDDO.

If the WHILE condition is false after the first run, the program continues immediately after ENDDO.

The DO command must immediately follow the WHILE, similar to IF/THEN. ENDDO can occur at any point after DO and WHILE. If the interpreter finds a DO without a WHILE, the result is a SYNTAX ERROR. An ENDDO without a WHILE causes an ENDDO WITHOUT WHILE ERROR.

WHILE/ENDDO loops can be nested up to 15 levels. Once nesting goes past the fifteenth level, the interpreter responds with an OUT OF MEMORY ERROR.

Example:

```
10 INPUT"NUMBER (0-50)";AZ
20 ZP=0
30 WHILE NOT(ZP=AZ) DO
40 SCPRINT 2^ZP:ZP=ZP+1
50 ENDDO
```

This routine displays exponents of 2 from 2^0 to 2^49. The WHILE loop runs until AZ equals to ZP.

LEVELWHL (266) (f)

LEVELWHL returns the current nesting depth of WHILE/ENDDO loops.

Format: VT = LEVELWHL

VT can range from 0 (no WHILE/ENDDO loop structures) to 15 (maximum nesting level).

POPWHL (205) (c)

You can exit any loop level at any time with the POPWHL command. Directly after POPWHL, GOTO can be used to exit the loop. POPWHL clears the WHILE stack of the currently stored loop value.

Format: POPWHL

REPEAT (117) (c)
UNTIL (118) (c)

Unlike the WHILE/ENDDO command, the REPEAT/UNTIL loop tests for the end of the loop. REPEAT/UNTIL always executes at least once.

Format:
```
10 REPEAT ...
20 ...
30 UNTIL [CONDITION]
```

The program commands found in between REPEAT and UNTIL execute until the condition following UNTIL is true. As soon as this condition is met, the program continues after the UNTIL. REPEAT and UNTIL can be placed on different lines of the program.

If the interpreter finds an UNTIL without a previous REPEAT, the result is an UNTIL WITHOUT REPEAT ERROR.

REPEAT/UNTIL can be nested in up to 15 levels. Going past 15 levels causes an OUT OF MEMORY ERROR.

Example:

```
100 REPEAT
110 : B=0
120 : REPEAT
130 :   C=0
140 :   REPEAT
150 :     D=0
160 :     REPEAT
170 :       E=0
180 :       REPEAT
190 :         SCPRINT A+B+C+D+E
200 :         E=E+1
210 :       UNTIL E=1
220 :       D=D+1
230 :     UNTIL D=2
240 :     C=C+1
250 :   UNTIL E=3
260 : B=B+1
270 : UNTIL B=4
280 A=A+1
290 UNTIL A=5
```

This program contains five separate REPEAT/UNTIL constructs, and displays the number of executions on the screen.

LEVELREP (265) (f)

LEVELREP lists the current REPEAT/UNTIL loop's nesting level.

Format: VT = LEVELREP

VT can range from 0 (no REPEAT/UNTIL loop) to 15 (maximum number of loops).

POPREP (204) (c)

You can exit a REPEAT/UNTIL loop at any time using the POPREP command. Directly after POPREP, GOTO can be used to exit the loop. POPREP clears the REPEAT stack of the currently stored loop value.

LOOP (119) (c)
LPEXITIF (120) (c)
ENDLOOP (121) (c)

LOOP/ENDLOOP lets you set up common types of loops. The branch can be designated at any time, which makes LOOP/ENDLOOP extremely flexible, and useful when no other loop type will do the job.

Format:
```
10 LOOP ...
20 ...
30 [LPEXITIF ... 'CONDITION']
40 ...
50 ENDLOOP
```

The program data between the LOOP and ENDLOOP executes until the condition following LPEXITIF is fulfilled.

If LPEXITIF or ENDLOOP is used without a previous LOOP, the result is an LPEXITIF/ENDLOOP WITHOUT LOOP ERROR.

LOOP/ENDLOOP can be nested up to 15 levels. Going past 15 levels causes an OUT OF MEMORY ERROR.

Example:

```
10 LOOP
20 INPUT"STRING:";ZK$
30 LPEXITIF ZK$="END"
40 SCPRINT "LENGTH:";LEN(ZK$)
50 ENDLOOP
60 ...
```

This example lists the number of characters you type in at the prompt. You could have done this with an IF/THEN/POPIF/ENDIF sequence, but this program code performs the same job with a little more elegant style.

LEVELLP (267) (f)

LEVELLP lists the current LOOP/ENDLOOP loop's nesting level.

Format: VT = LEVELLP

VT can range from 0 (no LOOP/ENDLOOP loop) to 15 (maximum number of loops).

POPLP (207) (c)

POPLP lets you exit a loop at any time. Like the other POP commands, POPLP takes the stored value from the LOOP stack. Once the loop exits, you must branch with another command, like GOTO.

6.5 Procedures

The name *procedure* is taken from the Pascal programming language. A procedure is nothing more than a special subroutine. Unlike GOSUB/RETURN, all the variables defined within a procedure are *local* (i.e., only the procedure can use these variables). By the same token, procedures cannot access variables within the main program. Procedures allow you to create individual program sections that can be used with other programs without a conflict of variables.

Labels are a basic method of getting into a procedure (see Section 6.2).

The limits of extra variable ranges mean more work in some respects. The procedure may require one or more variable values from the program calling it (this is almost always the case), so you have to add these variables to the procedure.

This means that you have to pick and choose which variable the procedure needs from the main program. As you'll see from the descriptions below, these definitions are simpler than they might sound here.

NOTE: Writing procedures can become very complicated, especially when manipulating or setting stacks and vectors. If your program stops in the middle of a procedure with an error, before making any program changes, RESET or NEW the computer, then retrieve the program with POLD (this resets pointers correctly).

PROCEDURE	(126)	(c)
PROCEND	(127)	(c)
CALL	(128)	(c)

Every procedure definition begins with the PROCEDURE command and ends with PROCEND. Both commands must appear at the beginning of a program line.

The name of the procedure follows the PROCEDURE command. You can give it any name you wish. The only stipulations are that you place the procedure name in quotation marks, and that the procedure name occupies less than a program line in length.

The variables follow the name, each separated by commas and all variables placed in parentheses.

Next comes the procedure itself -- the commands you want executed by the procedure.

The procedure concludes with the PROCEND command. Like PROCEDURE, this must start at the beginning of a program line.

A typical procedure looks like this:

```
100 PROCEDURE...(HEADER)
110 ...
120 ... (PROCEDURE COMMANDS)
130 ...
140 PROCEND (END MARKER)
```

When the interpreter finds a PROCEDURE command within a running program, those program lines execute up until the PROCEND command.

Procedures can be inserted at any point within a program (except between control structures). A PROCEND without a preceding PROCEDURE causes a PROCEND WITHOUT PROCEDURE ERROR.

Procedures are called using the CALL command. If a procedure doesn't exist, the computer responds with an UNDEFINED PROCEDURE ERROR.

Format: CALL "NAME", (VAR_LIST)

"NAME" can be any string. It is the name of the procedure you want called.

VAR_LIST is the set of variables you want used. A variable list is necessary when variables must be used by both the main program and the procedure. i.e. (a$,b$,x;z$,q)

Variables within the PROCEDURE list can be broken into two categories, separated by semicolons. The contents of the variables to the left of the semicolon are received from the CALL. The variables to the right contain the values to be returned.

In CALL's list the values to the left of the semicolon are the values to be passed to the procedure; the right hand variables will contain the values to return.

```
100 PROCEDURE "TEST",(RW,CL,LE;EG$)
110 CRSET ZE,SP:CRON:KBGETV EG$,LE:CROFF
120 PROCEND
```

This procedure takes an input LE characters long from column CL and row RW, and places it in the variable EG$. A CALL for this procedure can look like this:

```
CALL "TEST",(10,2,17;W$)
```

A string 17 characters long is taken from column 2, row 10. This is placed in the variable W$ in the main program.

The common format looks something like this:

```
10 CALL "...",(W1,W2%,W3$,...;A1,A2%,A3%,...)
100 PROCEDURE "...",(B1,B2%,B3$,...;T1,T2%,T3$,...)
```

NOTE: The variables set for access by CALL must be defined BEFORE the first procedure call. Even if you just assign values of zero, that will work fine.

The number of variables or number of values is limited to the maximum length of the program line. The result of missing parameters can be inconvenient at best. In any case, the semicolon must remain in PROCEDURE, even if the variables themselves are missing. e.g., PROCEDURE "ABC",(A,B;), PROCEDURE "CBA",(;D,E), or PROCEDURE "BAC",(;).

It is also important that variables and values shared by CALL and PROCEDURE be in the same order and be the same type (e.g., don't try passing a string variable to a number). Errors of this type result in a PROCEDURE-PARAMETER ERROR.

There are some things about variable arrays and array elements to keep in mind. These explanations are for parameter passing to a procedure only, not for the return to the main program:

1) Individual array elements can be treated like normal variables:

```
100 CALL "NAME",(A(10),B$(3,7),...)
...
500 PROCEDURE "NAME",(BC,G$,...)
```

2) Larger numbers of arrays present problems. Here the first and last element of the desired array range, separated by a minus sign (-), must be given. If, for example, you want elements 7 through 19 of the array FT passed, then the code would look like this:

```
CALL "NAME",(FT(7)-FT(19),...)
```

The example below gives all 28 elements of the string array AG$:

```
CALL "NAME",(AG$(0)-AG$(27),...)
```

NOTE: An array passed to a procedure must be defined before passing to the procedure using the DIM statement. If you don't do this, the program may crash.

The corresponding variable array in the procedure must be predefined in the procedure header according to the last array element (again, using DIM). The procedures for the above two examples might look something like this:

```
PROCEDURE "NAME",(DA(12),...)
```

```
PROCEDURE "NAME",(SR$(27),...)
```

A little trick makes variable dimensioning feasible:

```
PROCEDURE "NAME",(DM,FL%(DM),...)
```

This assigns the integer array FL% a size of variable DM. When the procedure is called by the command CALL "NAME",(9,DW%(0)-DW%(9),...), FL% is assigned ten elements and is filled with the elements of the array DW%(..).

Naturally, you can assign the array larger dimensions than needed; the rest of the array fills with null elements.

BeckerBASIC uses only two controlling factors in array passing, type control and length control. Type control requires only that you make sure that arrays are of the same type when setting them up for passing. In other words, putting real into integer and vice versa is illegal. This results in a PROCEDURE-PARAMETER ERROR.

The length control simply compares the array lengths, making sure that the one has sufficient room to take on the other. Dimension control is impossible. One thing you can do here is convert multidimensional arrays to smaller dimensions, and vice versa.

Example:

```
10 CALL "NAME",(W(0,0)-W(3,3),...)
...
100 PROCEDURE "NAME",(FA(15),...)
```

The two-dimensional array W(..,..) transfers its contents into the single-dimensional array FA(...). You can also do the reverse:

```
10 CALL "NAME",(FA(15),...)
...
100 PROCEDURE "NAME",(W(3,3),...)
```

The one-dimensional array FA(..) transfers to the two-dimensional array W(..).

The first line of the note about passing arrays also applies to passing arrays from procedure to the main program. The procedure return to the main program is almost the same, with one exception: All arrays and simple variables returned must be predefined before the passing takes place. After CALL and the semicolon you give the first element of the array whose values you want passed (default is element 0). See the example below.

The example at the end of this chapter, WINPROC, is a complete demonstration of structured programming using procedures. Below are two shorter examples of procedures:

```
10 DIM TC$(15),ZN%(25):AK=0
20...
100 CALL "NAME",(12*4+CG,3,TC$(9)-TC$(11);AK,ZN%(6))
...
500 PROCEDURE "NAME",(AN,HV,AM$(HV);Z%,D%(3)-D%(17))
```

NOTE: The term 12*4+CG had to be predefined before CALL (e.g., with CG=0). The string array AM$ is dimensioned with a size of HV. The variable returned is given to the real variable AK by the integer variable Z%. This is handled as a simple variable. The element of the integer array D% must be converted into an integer array (ZN%; this fills in the seventh element).

Remember: The variable AK and the array ZN% must be defined / dimensioned before the procedure call (see program line 10 above):

```
10 DV(2,0)=7:DV(1,2)=138
100 CALL"NAME",(DV(0,0)-DV(2,2);)
110 ...
120 PROCEDURE "NAME",(MB(8);)
```

This example converts the two-dimensional array DV to the one-dimensional array MB. MB(2) contains the value 7, MB(7) the value 138.

Remember: The semicolon in the variable list must be given, even if no values appear to the right of the semicolon.

Procedures can be nested like loops. Nesting means in this case that a procedure can be called from within another procedure during execution of that procedure. This is particularly interesting when you want to create a self-calling, or recursive procedure.

You can have a maximum nesting level of 15. If you go past this level, the computer responds with an OUT OF MEMORY ERROR.

NOTE: On every procedure call, a new area of variable memory must be set aside, so nesting procedures can make great demands on the memory. Keep this in mind when writing recursive procedures. If, for example, you have defined three real variables within a procedure, and you plan on making procedures self-calling down to the seventh level, necessary variable memory for three variables is 147 bytes.

LEVELPROC (268) (f)

LEVELPROC returns the current nesting depth of procedures:

Format: VT = LEVELPROC

VT can range from 0 (no procedure nesting) to 15 (maximum nesting level).

POPPROC (206) (c)

POPPROC lets you exit a procedure before it's done executing. POPPROC clears the variable range of the procedure in BASIC memory and retains the last called procedure from the procedure stack. A jump command lets you go to any point in the program.

Format: POPPROC

The formats of the following three commands are the same as those in Section 5.3.3 (Overlays). Procedures can also be loaded, saved, etc. without loss of data.

DSAVEPROC (129) (c)

DSAVEPROC saves a procedure to diskette under the assigned name.

Format: DSAVEPROC NA$

NA$ is the name of the procedure to be saved. This string can be a maximum length of 16 characters (the disk drive cannot handle longer names). Although you can save procedures with DSAVEL (see Section 5.3.1), DSAVEPROC is easier to use.

Example:

DSAVEPROC "TEST" saves the procedure TEST to diskette.

DLOADPROC (130) (c)

DLOADPROC loads a saved procedure into memory without destroying any data. The difference between this command and DOVERLAY (Section 5.3.3) is that the procedure is attached to the program, rather than interspersed with the running program.

All procedures containing the same line numbers as the running program can be easily loaded into the running program. Once it happens, you cannot edit the two programs in memory; but you can remove the procedure with the DELPROC command.

DLOADPROC loads like any normal program or program section.

Format: DLOADPROC NA$

NA$ is the name of the procedure to be loaded. This name can be a maximum length of 16 characters.

Example:

DLOADPROC"OUTPUT" loads the procedure OUTPUT into memory and appends the procedure to the currently running program.

DELPROC (131) (c)

DELPROC deletes a procedure found within a program, without loss of variables.

Format: DELPROC NA$

NA$ is the name of the procedure you wish deleted. To delete individual program sections instead, use the PDEL (with variable loss; see Section 2.1.1) or LDEL (no variable loss; see Section 5.3.3).

Example:

DELPROC "OUTPUT" deletes the OUTPUT procedure from the running program.

If you want to use a procedure in an extreme case during editing, use the following:

DLOADPROC "NAME":CALL "NAME",(...)DELPROC "NAME"

Now to demonstrate procedures, here is a practical example: Window design for the Commodore 64.

Windows are normally rectangular screen areas into which information is displayed or data is entered. One particular problem with windows is the temporary storage of screen information. In addition, different window parameters such as window size, position, etc. must be retained. Variables work well for this, except that when you leave a procedure, the variables are lost.

The solution to this problem lies in the memory range from 40960 to 48960 (free RAM).

The maximum number of simultaneously active windows is five. You can change this parameter, like all other aspects of the program. The individual open windows can overlap each other.

However, the nesting principle holds true here, i.e., before you can deactivate a window part way down the nesting levels, the last active window(s) must be closed as well.

The program at the end of this section is on your distribution diskette under the name WINPROC. It contains a total of five procedures:

WINOPEN	Buffer storage of a window area
WINDEL	Clear a window
WINPRINT	Data output in a window
WININPUT	Data input in a window
WINCLOSE	Store a window area

NOTE: The window routines must be initialized at the beginning of a program with POKE 45535,0.

The first step to creating a window is calling the WINOPEN procedure (program lines 1070-1280). WINOPEN stores the necessary screen area in the back of RAM memory. You must assign the starting position (upper left hand corner of the window) and the horizontal and vertical orientation of the window. The sequence for this orientation is starting line (1-25), starting column (1-40), line length (1-40), column length (1-25).

If you no longer need the window, you can close the window you opened with the WINCLOSE procedure (program lines 1320-1490).

The WINDEL procedure (program lines 1530-1610) prepares the screen area for data input by clearing the area. WININPUT (program lines 1760-2100) lets you enter any data into the window. The cursor can be moved anywhere in the window with the cursor keys. The other key functions (colors, for example) are also accessible. WININPUT treats the window area as a miniature "screen."

Pressing the <RETURN> key places the window contents in a one-dimensional string array (one element per window line), and exits the procedure. To exit the procedure and have the procedure ignore the data, press <SHIFT><RETURN>.

Remember: The input array must be dimensioned in the main program BEFORE the procedure executes.

The WINPRINT procedure (program lines 1650-1720) writes the data in a one-dimensional string array (one array per window line) to a window. The parameters applying to WINDEL, WININPUT and WINPRINT also apply to WINOPEN.

You now have a general background of window techniques. There are other procedures in this program, though. You can, for example, remove the border from a window, or move different windows around the screen.

WININPUT can be extended to perform other functions. For example, you could make it scroll up or down, or include a special <cursor home>.

WARNING: These procedures contain no check for parameters. Adding this kind of error checking should pose little problem for you, if you want it.

```
10 'WINPROC'
100 'DELETE THESE LINES AFTER APPENDING TO ANOTHER PROGRAM'
110 :
115 CLS:LETTERON:CRCOL 1
120 SCPRINT AT 3,1;"THIS PROGRAM CANNOT START ON ITS OWN!"
130 SCPRINT AT 6,1;"IT IS INTENDED TO BE INTEGRATED"
140 SCPRINT AT 7,1;"WITH OTHER PROGRAMS YOU"
150 SCPRINT AT 8,1;"HAVE WRITTEN."
160 SCPRINT AT 17,6;"PLEASE PRESS A KEY."
165 WAITKEYA:END
170 :
180 :
190 :
1000 'WINDOW PROCEDURE'
1010 '(C) 1987 BY MARTIN HECHT'
1020 :
1030 'SAVED TO DISKETTE UNDER THE NAME WINPROC'
1040 :
1050 :
1060 :
1070 PROCEDURE "WINOPEN",(WZ,WS,ZL,SL;)
1080 :
1090 WA=TEEK(45535):'CURRENT NUMBER OF WINDOWS'
1100 IF WA=5 THEN SCPRINT "TOO MANY WINDOWS!":POPIF:PROCEND
1110 ELSE WA=WA+1:POKE 45535,WA
1120 ENDIF
1130 :
1140 'SCREEN MEMORY BUFFER'
1150 IF WA=1 THEN BA=45505
1160 ELSE BA=DEEK(45535-WA*2+2)
```

```
1170 ENDIF
1180 BD=PEEK(648)*256
1190 FOR AZ=WZ TO WZ+SL-1
1200 ZW=BD+(AZ-1)*40:TRANSFER ZW+WS-1,ZW+WS+ZL-2,BA-(ZL-1):BA=BA-ZL
1210 NEXT AZ
1220 :
1230 'PARAMETER STORAGE'
1240 DOKE 45535-WA*2,BA:'STARTING ADDRESS OF SCREEN BUFFER MEMORY'
1250 POKE 45525-WA,WZ:POKE 45520-WA,WS:'WINDOW STARTING POSITION'
1260 POKE 45515-WA,SL:POKE 45510-WA,ZL:'COLUMN /LINE LENGTHS'
1270 :
1280 PROCEND
1290 :
1300 :
1310 :
1320 PROCEDURE "WINCLOSE",(;)
1330 :
1340 WA=TEEK(45535):'CURRENT NUMBER OF WINDOWS'
1350 IF WA=0 THEN POPIF:PROCEND:ENDIF
1360 :
1370 'GET PARAMETERS'
1380 BA=DEEK(45535-WA*2):'BUFFER STARTING ADDRESS'
1390 WZ=TEEK(45525-WA):WS=TEEK(45520-WA):'WINDOW STARTING POS.'
1400 SL=TEEK(45515-WA):ZL=TEEK(45510-WA):'COLUMN / LINE LENGTH'
1410 :
1420 'RESTORE SCREEN AREA'
1430 BD=PEEK(648)*256:BA=BA+SL*ZL
1440 FOR AZ=WZ TO WZ+SL-1
1450 TRANSFER BA-(ZL-1),BA,BD+(AZ-1)*40+WS-1:BA=BA-ZL
1460 NEXT AZ
1470 :
1480 WA=WA-1:POKE 45535,WA:'NUMBER OF OPEN WINDOWS'
1490 PROCEND
1500 :
1510 :
1520 :
1530 PROCEDURE "WINDEL",(WZ,WS,ZL,SL;)
1540 :
1550 'CLEAR SCREEN AREA'
1560 BD=PEEK(648)*256
1570 FOR AZ=WZ TO WZ+SL-1
1580 ZW=BD+(AZ-1)*40:MYFILL ZW+WS-1,ZW+WS+ZL-2,32
1590 NEXT AZ
1600 :
1610 PROCEND
1620 :
1630 :
1640 :
```

```
1650 PROCEDURE "WINPRINT",(WZ,WS,ZL,SL,AG$(SL-1);)
1660 :
1670 'DISPLAY DATA IN WINDOW'
1680 FOR AZ=0 TO SL-1
1690 CRSET WZ+AZ,WS:SCPRINT AG$(AZ)
1700 NEXT AZ
1710 :
1720 PROCEND
1730 :
1740 :
1750 :
1760 PROCEDURE "WININPUT",(WZ,WS,ZL,SL;EG$(0)-EG$(SL-1))
1770 :
1780 KEYREPEATON
1790 ZE=WZ:SP=WS:'CURSOR IN WINDOW STARTING POS.'
1800 LOOP
1810 CRSET ZE,SP:CRON:WAITKEYA:GET TD$:CROFF
1820 SELECT ASC(TD$):'CURRENT INPUT'
1830 CASE 29:'CURSOR RIGHT'
1840 "CR":IF NOT(SP=WS+ZL-1) THEN SP=SP+1
1850 ELSE IF NOT(ZE=WZ+SL-1) THEN SP=WS:ZE=ZE+1:ENDIF
1860 ENDIF
1870 CASE 157:'CURSOR LEFT'
1880 IF NOT(SP=WS) THEN SP=SP-1
1890 ELSE IF NOT(ZE=WZ) THEN SP=WS+ZL-1:ZE=ZE-1:ENDIF
1900 ENDIF
1910 CASE 17:'CURSOR DOWN'
1920 IF NOT(ZE=WZ+SL-1) THEN ZE=ZE+1:ENDIF
1930 CASE 145:'CURSOR UP'
1940 IF NOT(ZE=WZ) THEN ZE=ZE-1:ENDIF
1950 CASE 13:'ACCEPT DATA'
1960 GOTO "WUEB"
1970 CASE 141:'CANCEL'
1980 GOTO "WEND"
1990 OTHER SCPRINT TD$;:GOTO "CR"
2000 ENDSEL
2010 ENDLOOP
2020 :
2030 "WUEB":'ACCEPT DATA'
2040 DIM EG$(SL-1)
2050 FOR AZ=0 TO SL-1
2060 SGETV EG$(AZ),ZL,WZ+AZ,WS
2070 NEXT AZ
2080 :
2090 "WEND":POPLP
2100 PROCEND
```

Here is a practical application of this program (delete lines 100-160 from WINPROC and add these lines). WZ/WS = starting line/starting column of window; ZL/SL = line/column length:

```
100 POKE 45535,0:'DO NOT FORGET TO INCLUDE THIS'
110 WZ=5:WS=5:ZL=10:SL=10:DIM W$(SL-1)
120 CALL "WINOPEN",(WZ,WS,ZL,SL;)
130 CALL "WINDEL",(WZ,WS,ZL,SL;)
140 CALL "WININPUT",(WZ,WS,ZL,SL;W$(0))
150 CALL "WINCLOSE",(;)
```

By changing the parameters in line 110, you can make your window any size, and enter any data in your window.

7. GEOS

GEOS operates in hi-res mode on the Commodore 64. The bitmap for GEOS lies between memory locations 40960 and 48960. This memory range uses the hi-res commands discussed in Chapter 8.

GEOS and hi-res commands can be used together. Both programming areas have some similarities, but these two subjects require two separate chapters. This chapter describes the creation of dialogue boxes and drop-down menus.

There is one thing you should remember when working with GEOS from BeckerBASIC: All tables and strings sent to GEOS must end with 0 (CHR$(0)). Not doing this can lead to a system error. Also note that the commands and functions listed in this chapter are not accessible in the Input System. You will have to switch to the Testing System with <CTRL><Commodore> to test your program.

Since GEOS uses a different text coding from BASIC, you must convert any text from Commodore ASCII to GEOS ASCII with the ASCGEOSW command (see Section 7.3). By the same token, any text transferred from GEOS to BeckerBASIC must be converted by the GEOSASCW command.

Not everything can be produced by the GEOS commands, even though it might seem as if you can program some things at first glance. This can occur because of memory layout, or for other technical reasons.

The ability or inability to program is connected with the differences between BASIC program structures (especially BeckerBASIC) and GEOS program structures. For example, it's technically impossible to edit icons from geoPaint. This is because the icon control runs in "multitasking" mode, and the mouse pointer must be freely movable during the entire program run, so the two items interfere. You might be able to jump to another program using a keypress in ONKEYGO (see Chapter 3).

With the help of a few programming tricks, you can simulate such important GEOS functions as drop-down menus and dialogue boxes from BASIC. In both cases, command control is given to GEOS then returned to BeckerBASIC.

Drop-down menus and dialogue boxes present another problem. These both need the second hi-res bitmap range starting at memory location 24576. This area of memory needs to be protected. Use the PBCEND command to do this. This reduces the available BASIC memory by almost eight kilobytes.

Both dialogue boxes and drop-down menus are best used with programs that load other programs through overlay commands from diskette (see Section 5.3).

GEOSON (211) (c)
GEOSOFF (212) (c)

GEOSON switches on high-resolution graphics. It is identical to the HRON function (see Chapter 8). GEOSOFF (identical to HROFF) turns off hi-res graphics, and returns program control to the normal text screen.

Format: GEOSON: ... :GEOSOFF

IMPORTANT: Before you use any GEOS commands, you must call the GEOSON (or HRON) command. Otherwise, the result will be a system error. Hi-res graphic drawing is impossible without these commands (the picture is drawn first, then the graphic screen switches on).

The same rule applies to going from hi-res mode to the normal text screen. If you want a text screen command (e.g., SCPRINT), you must switch off the hi-res screen with GEOSOFF (or HROFF). Again, failure to do this leads to a system crash.

Remember these rules, you'll save yourself a lot of time, trouble and system errors.

HRDEL (220) (c)

Once hi-res graphics are active, you'll want to clear the graphic bitmap. The HRDEL command performs this function.

Format: HRDEL

HRGDCOL	(218)	(c)
HRPTCOL	(219)	(c)
HRGTCOL	(272)	(f)

These three commands handle colors in the hi-res bitmap. HRGDCOL sets the background color, HRPTCOL sets the point color and HRGTCOL returns the current hi-res colors.

Format: HRGDCOL F1: ... :HRPTCOL F2: ... :FB = HRGTCOL (CD)

F1 is the color code you wish assigned to the unset points of the graphic. Values for F2 range from 0 to 15 (see the PCOLOR command, Section 3.3).

F2 is the color code you wish assigned to the set points of the graphic. Values for F2 range from 0 to 15 (see the PCOLOR command, Section 3.3).

CD sets the color status you want. CD=0 returns the current point color; CD=1 returns the current background color.

The border color can be changed with the BORDER command (see Section 3.3).

Here is a demonstration of hi-res graphic initialization:

```
100 GEOSON:'GRAPHICS ON'
110 HRDEL:'CLEAR BITMAP'
120 HRGDCOL 1:'BACKGROUND COLOR'
130 HRPTCOL 2:'POINT COLOR'
135 HRPLOT 160,100:HRPLOT 161,100:HRPLOT 162,100
140 WAITKEYA:GEOSOFF
```

Remember that HRON can be used instead of GEOSON to produce the same result. See Chapter 8 for more commands in hi-res mode.

7.1 Drop-down menus

GEOS puts its drop-down menu data into a table in memory. BeckerBASIC uses this table. This code table can be created using the `BeckerBASIC` Drop-down Menu Construction Set.

The Drop-down Menu Construction Set is an application program written in BeckerBASIC. This program lets you set up your data in the correct form, and place it at the correct location in the menu table. Once you have all your data in place, the Construction Set displays a sample of your drop-down menu with all the parameters you defined.

GEOS text appears in the menu in proportional type, so initially you may not get the menu spacing correct. If you make a mistake, tell the program N when it asks if you want the menu saved.

The program then allows you to correct your data. When you finish editing your data, the drop-down menu sample appears on the screen again. You can repeat this procedure as often as you wish. When the menu is finished, and your parameters are entered, the table is saved to disk under any name you wish for later recall. The commands you require for recall are DLOADM (load a menu table from diskette, see Chapter 5) and PDMENU described below.

PDMENU (210) (c)

PDMENU activates (displays) a drop-down menu and allows access using the mouse pointer. The only additional parameter needed is an address.

Format: PDMENU AD

AD is the starting address of the code table +95. If the table address starts at location 24000, AD must contain the value 24095. The menu then appears on the screen for easy selection. As soon as you click a menu option, the option blinks and the menu closes (the hi-res screen restores the area where the menu had been).

A sub-menu will appear below this menu item (a set of selections connected with the menu item) for further selection. When you click on an item from a sub-menu, both the sub-menu and the menu disappear, and the program continues.

MENUCODE (271) (f)

This function returns the code of the menu or sub-menu item you clicked.

Format: MC = MENUCODE (CD)

CD returns the number of either the clicked main menu item or the number of the corresponding sub-menu item. Values for CD range from 0 to 10. CD=0 gives you the number of the clicked menu item in MC. CD=1 returns the number of the sub-menu item.

The program can react to this data. Here is how the data reacts to menus, using the ON GOSUB command:

```
100 ON MENUCODE(0) GOSUB 1000,2000,3000
...
1000 ON MENUCODE(1) GOSUB 1100,1200
...
2000 ON MENUCODE(1) GOSUB 2100,2200
...
3000 ON MENUCODE(1) GOSUB 3100,3200 …
```

Line 100 reads the menu. If you click the first menu item, the program branches to line 1000, where it checks for the sub-menu item selected. If you select the second menu item, the program jumps to line 2000, where the sub-menu item routine branches, and so on.

NOTE: If no sub-menu exists below a menu item, MENUCODE(0) returns the value 0. If this occurs, MENUCODE(1) becomes the clicked menu item.

7.1.1 Using the Drop-down Menu Construction Set

This program is on the distribution diskette under the name DDM.C.S, and can be opened by double-clicking its icon from the deskTop. After the program starts, it asks you where you want the data table placed.

Shorter menus can be easily stored in the cassette buffer (memory locations 828-1023). Longer menus should be placed in the memory range starting at memory location 24000 (directly under the second hi-res bitmap).

The next data requested is that of the main menu. First, you must determine the number of menu items (1-10), then decide whether the menu should be displayed horizontally or vertically. Next, the construction set needs the hi-res position of the menu. Finally, the text for each item is requested.

Once you've entered this data, the program asks whether or not you wish to create sub-menus. If so, you must enter the data for every menu requested. When all data is ready, press any key to see your drop-down menu on the screen.

Click on a menu item to end the display. The construction set asks if you like the menu. If so, you must enter the name under which you want the code table stored on diskette. If you answer the question with N, the program doesn't save the data to diskette, but reserves it in memory for the moment.

When the program ends, after the data table is stored on diskette, the screen displays a message on how to load and start your drop-down menu.

Try entering this data into the construction set program. Enter each value at the prompt and press the <RETURN> key (don't type in the commas):

```
24200,5,0,80,60,218,73,GEOS,FILE,VIEW,DISK,SPECIAL,
N,Y,5,1,109,73,154,144,OPEN,DUPLICATE,RENAME,INFO,PRINT,
N,Y,6,1,158,73,200,158,OPEN,CLOSE,RENAME,COPY,VALIDATE,FORMAT,N
```

This creates a familiar menu, the GEOS deskTop menu. To keep it simple, you are asked above to enter the data for the second and fourth sub-menus only. If you save this table to diskettee under the name PDMEX, you'd retrieve it in a program as follows:

```
10 PBCEND 24199:'LIMIT MEMORY'
```

```
20 DLOADM "PDMEX"
30 PDMENU 24200+95
```

7.2 Dialogue boxes

Creating dialogue boxes is as easy as making drop-down menus. BeckerBASIC features the Dialogue Box Construction Set, and two commands for dialogue box access.

DIALOGBOX	(213)	(c)
DIALCODE	(270)	(f)

DIALOGBOX activates (displays) a dialogue box. DIALCODE reads the button clicked within the dialogue box (YES, NO, etc.). Text input is also allowed in dialogue boxes.

Format: DIALOGBOX AD [,EG]: ... :CD = DIALCODE

AD is the memory address of the dialogue box's code table. As a rule, this table is small enough to be stored in the cassette buffer (memory locations 828 to 1023).

EG is the memory location at which the dialogue box text input is stored.

In most cases, dialogue boxes need some sort of button to allow the user to exit. When you define several buttons (e.g., YES and NO), DIALCODE reads the button the user clicked. This code corresponds to the code assigned to the button:

1 OK button
2 CANCEL button
3 YES button
4 NO button
5 OPEN button
6 DISK button
14 Click anywhere on the screen

Code 14 reads a click anywhere on the screen. This is functional only when installed from the Dialogue Box Construction Set. You might want to add information about Code 14 in the GEOS INFO screen.

7.2.1 Using the Dialogue Box Construction Set

The Dialogue Box Construction Set is on your distribution diskette under the name D.C.S. Double-click this program's icon from the deskTop.

The program first asks for the desired starting address of the code table. The cassette buffer area starting at location 828 is ideal for this purpose. Next, the program asks whether you want the standard dialogue box. The standard box is the one you see for file operations (e.g., `rename`). If you don't want the standard box, you'll need to supply the coordinates of the upper left and lower right corners of the dialogue box. The next parameter requested is the desired fill pattern for the box shadow. A fill pattern of 0 casts no shadow.

After this general data, parameters get more detailed. The order of the parameter codes isn't as crucial here as with the Drop-down Menu Construction Set, but you can't just throw these codes in at random. Furthermore, you must enter these codes as individual input routines.

The upper area of the screen displays a table of the available codes. For example, if you want an OK button, you type a 1 and press the <RETURN> key. The program then asks the position at which you want the button placed from the left (X-coordinate) and top (Y-coordinate) of the dialogue box.

After entering this input, you can assign the next code or codes. All these codes are placed in the table as you go along.

Text output within the dialogue box is made possible through code 11. Again, the program asks for the spacing from the left and top of the dialogue box. The final request is for the text you want.

Code 13 enables data input. Here the program asks for the starting position of the input line from the left and top of the dialogue box. Then you are asked for the maximum length of the input.

NOTE: When a dialogue box asks for input, the DIALOGBOX command must contain the address at which the data should be placed (e.g., DIALOGBOX 850,828: The computer puts the data at memory location 828). From this location you must use the GEOSASCW command to convert text to ASCII text format (see Section 7.3). Finally, the data must move to a string variable using MGETV (see Chapter 4).

The Construction Set always puts the data at memory location 828. Text input requires a table starting address higher than 828.

Code 33 displays the numbers of your input so far.

Select code 0 to end the input procedure. Now press a key to see your dialogue box. When you're done looking at it, click on a button to exit. NOTE: If you haven't put in any buttons before looking at this test display, you can't get out of the display. Power down and start over.

The program then asks if the dialogue box looks okay. Type y or n and press the <RETURN> key. Select y if you want to save the data; the program asks for the name under which you want the table saved. Select n to start over.

This program has the disadvantage that you can't edit one code at a time. That is, you have to start over if you want to change the dialogue box parameters.

Here's a practical example of using this Construction Set: Enter the first value at the first prompt, press the <RETURN> key, then enter the rest of the values in the same manner (don't type in the commas):

```
828,N,10,10,270,90,1,1,1,20,2,10,10,3,19,10,4,1,55,5,10,55,6,19,55,
11,8,35,PLEASE CLICK ON A BUTTON-ANY BUTTON.,0
```

Tell the program you want to see this dialogue box, and watch the result that appears: A box with six buttons and a line of text appears. Save this to diskette under the name DBEX and exit the Construction Set. Then type in, save and run this program to put your new dialogue box to use:

```
10 PBCEND 24575:'LIMIT MEMORY'
20 DLOADM"DBEX"
30 DIALOGBOX 828
```

DIALCODE reads the clicked button.

7.3 Entering and displaying hi-res text

BeckerBASIC has four commands for text input and output in hi-res mode. HRPRINT displays text; HRGET is the high-resolution equivalent of INPUT; ASCGEOSW and GEOSASCW convert ASCII text into GEOS format and GEOS text into ASCII format respectively.

HRPRINT (214) (c)

HRPRINT writes text to a specific position on the hi-res graphic screen. You can use different typestyles (e.g., italics or bold) with this text.

Format: HRPRINT X,Y,TX$

X,Y are the X-coordinate and the Y-coordinate of the hi-res pixel where the text begins. Values for X range from 0 to 319. Values for Y range from 0 to 199.

TX$ is the string containing text to be displayed. The text in TX$ can be up to 255 bytes long, and must end with a null (CHR$(0)).

Text output appears in proportional type. If the text contains uppercase characters (i.e., characters created by holding down the <SHIFT> key while pressing letter keys), you must convert the text using the ASCGEOSW command before using the text in HRPRINT. No conversion is necessary when using only lowercase lettering, since these letters appear as uppercase lettering in GEOS.

You have the following control characters available:

CHR$(08) deletes the last character displayed.

CHR$(09) moves the text cursor one character to the right.

CHR$(10) moves the text cursor one character down.

CHR$(11) moves the text cursor to the home position (X-coordinate=0,Y-coordinate=0).

CHR$(12) moves the text cursor one character up.

CHR$(13) moves the text cursor to the start of the next line of text following a carriage return.

CHR$(14) enables underlining mode. All characters following are underlined.

CHR$(15) disables underlining mode.

CHR$(18) enables reverse mode. All characters following appear in reverse video.

CHR$(19) disables reverse mode.

CHR$(24) enables bold mode. All characters following appear in bold style.

CHR$(25) enables italic mode. All characters following appear in italic (cursive) style.

CHR$(26) enables outline mode. All characters following appear in outlined style.

CHR$(27) disables all typestyles (italics, bold, etc.).

The different typestyles can be used in combination (e.g., bold italic text). Not all combinations give good-looking results.

Examples:

HRPRINT 150,100,CHR$(25)+"OUTPUT"+CHR$(0) displays the word OUTPUT in the center of the screen in italic text.

```
10 GEOSON
15 HRDEL
20 T$=CHR$(10):HRPRINT 20,20,"G"+T$+"E"+T$+"O"+T$+"S"+CHR$(0)
30 WAITKEYA
40 GEOSOFF
```

The above program displays the word GEOS one letter under the next.

```
10 GEOSON
12 HRDEL
14 HRGDCOL 15
15 HRPTCOL 0
20 TX$=CHR$(14)+CHR$(26)+"AN EXAMPLE"+CHR$(27)+CHR$(0)
25 HRPRINT 10,90,TX$
30 WAITKEYA
40 GEOSOFF
```

The text AN EXAMPLE appears in outlined, underlined text. CHR$(27) disables the typestyles in line 20.

ASCGEOSW (216) (c)
GEOSASCW (215) (c)

GEOS uses a different ASCII from Commodore BASIC. This means that every text you use must be converted to GEOS ASCII using the ASCGEOSW command.

GEOSASCW has the opposite effect. Data must be converted from GEOS format to normal BASIC format.

Format: ASCGEOSW AD,AE: ... :GEOSASCW AD,AE

AD is the starting address of the memory range whose contents must be converted to another format. Values for AD can theoretically range from 0 to 65535.

AE is the ending address of the memory range whose contents must be converted to another format. Values for AE can theoretically range from 0 to 65535.

Text contained within a string can be handled directly with both commands. You can use the following routine for this:

```
10 W=VARADR(TX$):WL=PEEK(W+1):WH=PEEK(W+2)
20 AD=WL+256*WH:AE=AD+PEEK(W)-1
30 ASCGEOSW AD,AE
```

VARADR conveys the address of the variable TX$. The current memory position of the contents of TX$ goes into AD and the ending address of TX$ goes into AE.

HRGET (217) (c)

HRGET reads data input in hi-res mode. A vertical blinking "text cursor" appears.

Format: HRGET X,Y,GT$

X,Y are the X-coordinate and the Y-coordinate of the upper left corner at which the text should be read. Values for X range from 0 to 319. Values for Y range from 0 to 199.

GT$ is the variable for the text input. GT$ must fulfill two conditions: The input length for GT$ must be defined beforehand, and the string must end with a null (CHR$(0)). For example, GT$=".....+CHR$(0) sets an allowable input length of five characters. You can use spaces or other characters instead of periods; these just set the input length. If you prefer other characters, the text in GT$ must be converted to GEOS text format by ASCGEOSW.

When you define GT$ as described above, the text cursor appears in the hi-res graphic at the end of the reading position. To move it to the left, you must press one of the cursor keys (all the cursor keys move the cursor left). Characters erase to the left of the cursor as you move the cursor left. To make the cursor appear at the beginning of the reading position, define GT$ as follows: GT$=CHR$(0)+"....."

Press the <RETURN> key to end the hi-res input. The text entered is in GEOS text format in GT$. The text must be converted to normal BASIC text format with GEOSASCW, and the CHR$(0) end marker removed (this can be done with GT$=LEFT$(GT$,LEN(GT$)-1)).

Examples:

```
10 GEOSON
12 HRDEL
15 GT$="          "+CHR$(0)
```

```
20 HRGET 100,50,GT$
30 VGETM 828,GT$:GEOSASCW 828,828+LEN(GT$)
40 MGETV GT$,LEN(GT$)-1,828
50 WAITKEYA
60 GEOSOFF
```

Line 10 defines GT$ to ten characters and gets input from the hi-res position 100/50. VGETM puts the text at memory location 828, where GEOSASCW converts the text to ASCII format. MGETV gets data from the variable GT$, and removes the end marker (CHR$(0)).

```
10 AB$="EXAMPLE"+CHR$(0)
20 HRGET 200,50,AB$
30 VGETM 828,AB$:GEOSASCW 828,828+LEN(AB$)
40 MGETV AB$,LEN(AB$)-1,828
```

The major difference between this and the other example is that the text EXAMPLE is the given text.

8. High-resolution graphics

High-resolution, or hi-res, graphics are definitely among the C64's finest features. The major disadvantage to the commands supporting these graphics is that they consume lots of memory. BeckerBASIC only contains the most important commands and functions supporting high-resolution graphics.

These commands are designed for efficient programming. Besides that, many commands can be simulated using a combination of hi-res instructions. Since BeckerBASIC commands optimize time wherever possible, speed is almost never a problem.

BeckerBASIC supports many GEOS specialties. For example, you can draw a filled rectangle with HRBOX, using one of the 45 GEOS fill patterns and 256 different combinations of line patterns. If you prefer, HRSTRING allows you to combine drawing commands into one string for fast execution.

You may want to review the descriptions of GEOS commands (see Chapter 7) before reading this chapter. You'll find a number of commands there that deal with data input and output while using GEOS's hi-res screen.

NOTE: that the commands and functions listed in this chapter are not accessible in the Input System. You wil have to switch to the Testing System with <CTRL>+<SHIFT> to test your program.

8.1 Initializing graphics

BeckerBASIC uses the first bitmap of GEOS for hi-res mode. This bitmap lies in memory locations 40960 to 48960. GEOS commands and hi-res commands can be used in parallel. Bitmap I lies outside of BASIC memory, so the two won't interfere with each other.

HRON	(137)	(c)
HROFF	(138)	(c)
HRGTON	(269)	(f)

HRON enables the hi-res graphic screen, and is identical to the GEOSON command in Chapter 7. HROFF disables the hi-res graphic screen.

HRGTON tells the user which mode is currently active. If HRGTON returns 0, the text screen is on; if HRGTON equals 1, hi-res mode is on.

Format: HRON: ... :HROFF: ... :CD = HRGTON

NOTE: Before you use a hi-res command, you must use a HRON command first. Failure to do so leads to a system crash, since you can't do hi-res pictures without hi-res mode. The same goes for the opposite direction: you can't use a standard text screen command without turning hi-res mode off with HROFF. Remember these rules whenever you work with hi-res graphics.

HRDEL	(220)	(c)

HRDEL clears the hi-res screen.

Format: HRDEL

HRGDCOL	(218)	(c)
HRPTCOL	(219)	(c)
HRGTCOL	(272)	(f)

HRGDCOL sets the background color of the hi-res screen. HRPTCOL sets the current point color of the hi-res graphic. HRGTCOL returns the current hi-res colors.

Format: HRGDCOL F1: ... :HRPTCOL F2: ... :FB = HRGTCOL (CD)

F1 is the color code of the unset pixels. Values for F1 can range from 0 to 15.

F2 is the color code of the set pixels. Values for F2 can range from 0 to 15.

FB is the current color read. CD=0 returns the current foreground pixel color; CD=1 returns the current background color.

You can use these commands in combination:

```
100 HRON:'GRAPHIC ON'
110 HRDEL:'CLEAR BITMAP'
120 HRGDCOL 1:'BACKGROUND COLOR'
130 HRPTCOL 2:'POINT COLOR'
135 HRPLOT 160,100:HRPLOT 161,100:HRPLOT 162,100
140 WAITKEYA:HROFF
```

8.2 Creating graphics

Commodore 64 hi-res graphics take up a total of 64,000 individual pixels, in a screen resolution of 320 pixels in the X-coordinate (horizontal) direction by 200 pixels in the Y-coordinate (vertical) direction. You can specify a pixel by stating its X-coordinate (0-319) and Y-coordinate (0-199). The coordinate system starts at the upper left corner of the screen.

HRPLOT	**(229)**	**(c)**
HRTESTP	**(273)**	**(f)**

HRPLOT lets you access any one of the 64,000 pixels. HRTESTP tells whether the pixel is set or unset.

Format: HRPLOT XK, YK [,ZM]: ... :CD = HRTESTP (XK,YK)

XK, YK are the X- and Y-coordinates of the desired pixel. Values for XK range from 0 to 319. Values for YK range from 0 to 199. Values outside these ranges result in an ILLEGAL QUANTITY ERROR.

ZM states whether the pixel is set (ZM=0) or unset (ZM=1). The default value for ZM is 0.

CD is the pixel status. If CD equals 1, the pixel is set; otherwise the value for CD is 0.

Examples:

HRPLOT 150,100:FL = HRTESTP(150,100) places a pixel in the middle of the screen. The variable FL contains the value 1.

HRPLOT 0,0:HRPLOT 319,0,1:HRPLOT 319,199,1:HRPLOT 0,199 sets the two left corner pixels and unsets the two right corner pixels.

HRLINE (224) (c)

HRLINE draws a line between two points on the hi-res screen.

Format: HRLINE X1, Y1, X2, Y2 [,ZM]

X1,Y1 are the coordinates of the first point in the line. Values for X1 range from 0 to 319; values for Y1 range from 0 to 199.

X2,Y2 are the coordinates of the last point of the line. Values for x2 range from 0 to 319. Values for Y2 range from 0 to 199. Values outside these ranges result in an ILLEGAL QUANTITY ERROR.

ZM indicates the character mode. A value of 1 for ZM means the pixel is unset; a set pixel has a value of 0. The default value is 0.

Example:

HRLINE 0,0,319,199:HRLINE 319,0,0,199 draws a diagonal line on the screen.

HRHLINE (225) (c)
HRVLINE (226) (c)

HRHLINE draws horizontal lines on the hi-res screen. HRVLINE draws vertical lines. These commands execute much faster than HRLINE. You can draw lines in up to 256 patterns with both commands.

Format: HRHLINE X1,Y1,X2,Y2,ZM: ... :
HRVLINE X1,Y1,X2,Y2,ZM

X1,Y1 are the coordinates of the leftmost (HRHLINE) or topmost (HRVLINE) pixel of the line. Values for X1 range from 0 to 319. Values for y1 range from 0 to 199.

X2,Y2 are the coordinates of the rightmost (HRHLINE) or bottom (HRVLINE) pixel of the line. Values for X2 range from 0 to 319. Values for Y2 range from 0 to 199.

ZM is the drawing mode. Values for ZM can range from 0 to 255. You can figure out your line pattern by converting the number into an 8-bit binary number. For example, ZM=170 would be 10101010 in binary notation. Every set bit of the pattern corresponds to a 1, and every unset bit is a 0. 170 gives a dotted line as a pattern.

Examples:

HRHLINE 10,10,300,10,255 draws a solid horizontal line.

HRVLINE 130,25,130,180,0 deletes any vertical line that might have been in the same position.

HRHLINE 50,100,150,100,102:HRVLINE 100,50,100,150,102 draws a dotted cross. 102 decimal equals 0110110 in binary notation.

HRFRAME (228) (c)

HRFRAME draws a rectangular frame of any size on the screen. This command uses the same drawing patterns as HRHLINE and HRVLINE.

Format: HRFRAME X1,Y1,X2,Y2,ZM

X1,Y1 are the coordinates of the upper left corner of the frame. Values for X1 range from 0 to 319. Values for Y1 range from 0 to 199.

X2,Y2 are the coordinates of the lower right corner of the frame. Values for X2 range from 0 to 319. Values for Y2 range from 0 to 199.

NOTE: The sequence of these X- and Y-coordinates is very important. If you give them in the wrong sequence (e.g., giving the lower right corner first), the drawing routine runs into trouble computing the frame.

ZM is the drawing mode. Values for ZM can range from 0 to 255. You can figure out your frame pattern by converting the number into an 8-bit binary number. For example, ZM=170 would be 10101010 in binary notation. Every set bit of the pattern corresponds to a 1, and every unset bit corresponds to a 0.

Example:

HRFRAME 0,0,319,199,170 draws a dotted frame.

HRBOX (227) (c)

HRBOX draws a filled rectangle of any size. You have 45 fill patterns available.

Format: HRBOX X1,Y1,X2,Y2,FM

X1,Y1 are the coordinates of the upper left corner of the box.Values for X1 range from 0 to 319. Values for Y1 range from 0 to 199.

X2,Y2 are the coordinates of the lower right corner of the box.Values for X2 range from 0 to 319. Values for Y2 range from 0 to 199.

NOTE: The order of these X- and Y-coordinates is very important. If you give them in the wrong sequence (e.g., giving the lower right corner first), the drawing routine runs into trouble computing the box.

FM is the fill pattern. Values for FM can theoretically range from 0 to 255, but the useful values are up to 44. FM=1 produces a completely filled box, while FM=0 deletes the box area.

Try this routine to see the available patterns:

```
100 HRON
110 FOR FM=0 TO 44
120 HRBOX 0,0,319,199,FM:HRPRINT 10,100,"PATTERN #"+CHR$(0)
```

```
122 Q$=STR$(FM)
123 HRPRINT 90,100,Q$+CHR$(0)
125 HRPRINT 10,140,"PRESS A KEY FOR NEXT PATTERN"+CHR$(0):WAITKEYA
130 NEXT FM
140 HROFF
```

Press a key to see each fill pattern.

Example:

HRBOX 20,15,100,130,10:HRBOX 50,55,170,140,23 displays two overlapping boxes with two different fill patterns.

HRINV (221) (c)

HRINV inverts the hi-res graphic display, i.e., set pixels become unset and unset pixels become set.

Format: HRINV

HRSTRING (230) (c)

HRSTRING lets you place a series of commands into a single string. This speeds up execution time and saves memory.

Format: HRSTRING KM$+CHR$(0)

KM$ is the string containing the codes required for the hi-res commands. KM$ can be up to 255 bytes in length. The codes for KM$ are as follows:

01 sets the intended graphic cursor at a certain point, using the coordinate set `Xlow/Xhigh/Y`.

02 draws a line between any two points. The coordinates of both ends of the line are set using the coordinate set `Xlow/Xhigh/Y` (the first pixel of the line is set by code 01).

03 draws a filled rectangle. The upper left coordinates of the rectangle are set by code 01. The lower left coordinates are set after code 03 using the coordinate set `Xlow/Xhigh/Y`.

05 Assigns a particular area pattern to a drawn filled rectangle. The pattern code (0-44) directly follows 05 (see `HRBOX`) .

07 draws a rectangular frame. The coordinates of the upper left corner are set by code 01, while the coordinates of the lower right corner directly follow 07 using the coordinate set `Xlow/Xhigh/Y`.

08 places the graphic cursor to the right by the coordinates stated in the form `Low/High`.

09 moves the graphic cursor a single byte number down.

00 is the end marker of the string. Failure to end an HRSTRING code string with this causes a system error.

Here are two examples to demonstrate the practicality of using HRSTRING:

```
10 HRON
20 HRDEL
30 'GRAPHIC CURSOR AT 10/10'
40 T1$=CHR$(1)+CHR$(10)+CHR$(0)+CHR$(10)
50 'DRAW A LINE FROM 10/10 TO 280/180'
60 T2$=CHR$(2)+CHR$(24)+CHR$(1)+CHR$(180)
70 HRSTRING T1$+T2$+CHR$(0)
80 WAITKEYA
90 HROFF
```

This program draws a line from coordinates 10,10 to coordinates 280,180. Press a key to end the program.

```
90 HRON
95 HRDEL
100 'SET GRAPHIC CURSOR TO 25/40'
110 T1$=CHR$(1)+CHR$(25)+CHR$(0)+CHR$(40)
120 'SET DRAWING PATTERN 17'
130 T2$=CHR$(05)+CHR$(17)
140 'DRAW BOX, 2ND COORDINATE 100/100'
150 T3$=CHR$(03)+CHR$(100)+CHR$(00)+CHR$(100)
155 WAITKEYA
```

```
160 T4$=CHR$(1)+CHR$(40)+CHR$(0)+CHR$(70)
170 'MOVE GRAPHIC CURSOR DOWN AND RIGHT'
180 'SET DRAWING PATTERN 9'
190 T5$=CHR$(05)+CHR$(09)
200 'DRAW BOX, 2ND COORDINATE 120/145'
210 T6$=CHR$(03)+CHR$(120)+CHR$(00)+CHR$(145)
220 HRSTRING T1$+T2$+T3$+T4$+T5$+T6$+CHR$(0)
230 WAITKEYA
240 HROFF
```

This program produces two overlapping rectangles with different fill patterns.

As already mentioned, HRSTRING saves time since the commands are read as machine language instead of interpreted BASIC.

When you have a number of these graphic strings in a program, it may help if you place these in a sequential or relative file. The DGETV command lets you easily read this data into the computer (see Chapter 5 for more on DGETV).

8.3 Loading and saving graphics

Loading and saving hi-res graphics can be done with DLOADM and DSAVEM (see Section 5.3). These commands LET you load or save parts of graphic screens. The next two commands access entire graphic screens.

HRDLOAD	(222)	(c)
HRDSAVE	(223)	(c)

HRDLOAD loads a hi-res screen from diskette. HRDSAVE saves a hi-res screen to diskette.

Format: HRDSAVE NA$: ... :HRDLOAD NA$

NA$ is the name assigned to the screen being saved or loaded. This name can be up to 16 characters long.

Examples:

HRDSAVE "HIRES" saves the hi-res bitmap under the name HIRES.

HRDLOAD "HIRDAT" loads the graphic file HIRDAT into graphic memory.

NOTE: BeckerBASIC saves hi-res graphics so that they load into memory byte-for-byte, without formatting or compression. Remember this when loading BeckerBASIC graphics into other graphic programs, or when loading other graphics into BeckerBASIC.

9. Sprite commands

The Commodore 64 allows up to eight sprites on the screen at once. The main purpose of freely movable graphic objects is in game programming, although sprites can be used effectively in other applications. For example, you can use sprites to create a title screen, just as in the DEMO program on your BeckerBASIC distribution diskette.

BeckerBASIC supports sprite development and movement through easy to use commands, in addition to the BASIC 2.0 POKE and DATA instructions.

Now for some fundamental information about sprites. Like normal characters which are defined in groups of pixels, sprites are also made up of pixels. Sprites have horizontal resolutions of 24 pixels and vertical resolutions of 21 pixels.

A pixel is set (on) or unset (off) according to the bytes setting up the sprite matrix. Every 24-pixel line takes up three bytes, while the 21 columns use up 63 bytes. You have a total of 63 values to control to make up the sprite's shape.

The best method of drawing a sprite design is with a sprite editor. This lets you see the sprite magnified, so you can control its shape easily. You'll find a sprite editor at the end of this chapter in Section 9.6. Once you design the sprite, the data must be fed into memory. You can put this data in most areas of memory.

There are two things to remember when working with sprites:

1) The starting address of the memory range must be divisible by 64 without a remainder.

2) The memory segment must be in the same 16K memory block as the screen. This means that you shouldn't place the data in active BASIC memory or any other area used heavily by the computer and the program.

MBDESIGN (Section 9.1) puts sprite data into memory - as long as this sprite data exists in a string. It's fairly easy to convert sprite data to a string; it saves memory and you can actually store this data in integer arrays or variables. The sprite editor at the end of the chapter converts data into strings, so they can be easily read by MBDESIGN.

You must state the location in memory you want the sprite data. This is done with the command MBBLOCK. This allows you to quickly switch between two blocks of sprite data.

9.1 Setting up sprites

MBDESIGN	(139)	(c)
MBDATA	(248)	(f)

MBDESIGN places a string containing sprite data into memory as a 64-byte memory segment. MBDATA does the opposite: It reads data in memory into a string variable.

Format: MBDESIGN BL, DA$: ... :SD$ = MBDATA (BL)

BL is the number of the desired memory block. You compute this block of memory with the formula BL = STARTING_ADDRESS/64. Values for BL range from 0 to 1023. Naturally, not all of these block numbers are useful; avoid active BASIC memory and zeropage memory.

DA$ is the 63-byte string expression containing the sprite data.

You must make sure that the sprite data is in the same 16K memory range for the sprite design as the active screen memory. That is, if you put the sprite design data in the first memory segment, the active screen should be in the first memory segment. The following table lists practical values for BL (the corresponding starting memory addresses are in parentheses):

```
MEMORY SEGMENT I              MEMORY SEGMENT II
(0-16383)                     (32768-49151)
13 (832) - 15 (960)           552 (35328) - 559 (35776)
```

As you can see, available screen memory is very small. The normal text screen starting at 1024 has enough room for three sprites at a time. Memory segment II (hi-res) is a little bigger; here you can fit eight sprite matrices.

NOTE: Sprite blocks 552 and 553 already have the data for the GEOS mouse pointer and the GEOS text cursor. Don't change these when you use GEOS commands that requires these two sprites. You can change the appearance of these sprites, however.

The sprite editor in Section 9.6 places the data into the variable MT$. This data can immediately be used with MBDESIGN BL, MT$. The matrix can be read into memory with MT$=MBDATA(BL). Both commands work in conjunction with the sprite editor. For example, if you include a GOSUB "MATDEF/S":MBDESIGN 13,MT$ in a program containing the sprite editor, MT$ defines a sprite matrix and places it in memory block 13.

MT$=MBDATA(15):GOSUB "MATDAR/S" reads MT$ from memory block 15 and places the result on the screen. Once the sprite data is read, you can move and manipulate the sprite with a number of commands.

MBCLR (182) (c)

MBCLR clears a sprite data block.

Format: MBCLR BL

BL is the number of the data block. Values for this block can range from 0 to 1023 (see MBDESIGN above).

Example:

MBCLR 13 clears block 13 (memory addresses 832-895).

MBINV (140) (c)

MBINV inverts the data block of a sprite matrix.

Format: MBINV BL

BL is the number of the data block. Values for this block can range from 0 to 1023 (see MBDESIGN above).

Example:

MBINV 558 inverts data block 558. The inversion "exchanges" the background color and the foreground color.

MBMOVE (183) (c)
MBCHANGE (184) (c)

MBMOVE and MBCHANGE allows the copying (MBMOVE) or exchange (MBCHANGE) of a sprite block.

Format: MBMOVE B1,B2: ... :MBCHANGE B1,B2

B1,B2 are blocks of memory. Values for B1 and B2 can range from 0 to 1023 (see also MBDESIGN).

MBMOVE copies the data block B1 into block B2. MBCHANGE swaps the contents of block B1 and block B2.

Examples:

MBMOVE 13,14 copies the contents of data block 13 into block 14.

MBCHANGE 13,14 exchanges the contents of data blocks 13 and 14.

MBAND (185) (c)
MBOR (186) (c)
MBEOR (187) (c)

These three commands allow you to compare sprite data blocks with each other.

Format: MBAND B1,B2: ... :MBOR B1,B2: ... MBEOR B1,B2

B1,B2 are blocks of memory. Values for B1 and B2 can range from 0 to 1023 (see also MBDESIGN).

Data blocks B1 and B2 are compared with each other using a logical AND (MBAND), logical OR (MBOR) or logical EXCLUSIVE OR (MBEOR). The result of this comparison appears in B1.

Examples:

MBAND 554,13 compares blocks 554 and 13 for logical AND. The result is in data block 554.

MBOR 555,556:MBEOR 555,557 compares block 555 and 556 for logical OR, then compares block 555 with block 557 for an EXCLUSIVE OR.

Logical comparisons let you manipulate data blocks quickly to achieve some interesting effects. Try these out with a few sprite matrices.

MBBLOCK (141) (c)
MBGTBLK (254) (f)

MBBLOCK arranges all eight sprites into one data block. MBGTBLK reads the current assignments for the individual sprites.

Format: MBBLOCK SC,NR,BL: ... :BL = MBGTBLK (SC,NR)

SC is the number of the desired screen used by MBBLOCK. When you want the sprite on the normal text screen, SC should have a value of 1. A value of 35 displays the sprite on the hi-res screen.

NR is the sprite number used by MBBLOCK. Values for NR range from 1 to 8.

BL is the data block number containing the sprite data. See MBDESIGN for further information on BL.

Examples:

MBBLOCK 1,5,13 assigns sprite 5 to data block 13. The corresponding data pointer moves to normal screen memory (starting at location 1024).

MBBLOCK 35,2,556:SCPRINT MBGTBLK (35,2) assigns sprite 2 to data block 556. The 35 sends the sprite to active hi-res memory. The SCPRINT following the MBBLOCK returns 556.

The next six commands control sprite color.

MBMODE (188) (c)
MBGTMOD (257) (f)

Sprites have two modes: The *single-color* and *multicolor* modes. Multicolor mode offers you a total of three colors for your sprites.

Sprite matrices interpret their setups by bits. The three bit combinations are 10, 01 and 11, and when a sprite is assigned multicolor mode, these combinations have their own colors (the combination 00 equals the background color). MBMODE lets you set the color mode for each sprite. MBGTMOD returns the current color mode for every sprite.

Format: MBMODE NR,NM: ... :NM = MBGTMOD (NR)

NR is the desired sprite's number. Values for NR range from 1 to 8.

NM is the color mode for the desired sprite. If NM=0, the sprite is in single-color mode; NM=1 means that the sprite is in multicolor mode.

Example:

MBMODE 5,1:MBMODE 8,0:A=MBGTMOD(5) assigns sprite 5 multicolor mode and sprite 8 single-color mode. The variable A contains a 1.

MBSETCOL (143) (c)
MBGTCOL (256) (f)

MBSETCOL sets the desired color of a sprite. MBGTCOL reads the current color code for individual sprites.

Format: MBSETCOL NR,FB: ... :FB = MBGTCOL (NR)

NR is the number of the sprite. Values for NR range from 1 to 8.

FB is the color code assigned/read. Values for FB range from 0 to 15 (see PCOLORS, Chapter 3).

Example:

MBSETCOL 4,9:MBSETCOL 7,0:SCPRINT MBGTCOL (4) turns sprite 4 brown and sprite 7 black. The SCPRINT command displays 9.

MBEXCOL (142) (c)
MBGTEXCL (255) (f)

MBEXCOL sets the additional colors for the bit combinations 01 and 11 in multicolor mode. MBGTEXCL reads the current additional colors.

Format: MBEXCOL F1,F2: ... :FN = MBGTEXCL (ZN)

F1 is the color code for bit combination 01. Values for F1 range from 0 to 15 (see PCOLORS, Chapter 3).

F2 is the color code for bit combination 11. Values for F2 range from 0 to 15 (see PCOLORS, Chapter 3).

FN,ZN set the color code (FN) according to the value in ZN. If ZN=1, FN is the code for the first additional color. If ZN=2, FN is the code for the second additional color.

Example:

MBEXCOL 3,7:CF=MBGTEXCL(2) assigns the color cyan to bit combination 01, and the color yellow to bit combination 11. The number 7 is assigned to FC (color code for the second additional color).

When two or more sprites appear at the same place on the screen, these sprites must be assigned priorities. Priority states which sprite passes in front of another when two or more overlap.

Also, it must be established whether a sprite can pass in front of or behind the other sprite. The first case (sprite/sprite priority) states that sprites with higher numbers pass in front of sprites with lower numbers. Therefore, if sprites 2 and 7 cross, sprite 2 passes behind sprite 7.

You cannot directly alter these priorities, but you can change them indirectly using MBBLOCK to switch sprite matrices. This automatically swaps priority.

MBPRIOR (169) (c)
MBGTPR (258) (f)

The MBPRIOR command can determine overlaps between sprite and background on a hi-res screen. MBGTPR reads the value set by MBPRIOR.

Format: MBPRIOR NR,PR: ... :PR = MBGTPR (NR)

NR is the number of the desired sprite. Values for NR can range from 1 to 8.

PR determines the sprite's priority. If PR is equal to 0, the sprite has higher priority than the background. If PR equals 1, then the sprite travels behind the background if the two objects cross.

Example:

MBPRIOR 2,1:MBPRIOR 8,0:T=MBGTPR(2) forces sprite 2 to travel behind the background, while sprite 8 has a higher priority than the background. Variable T has the value 1 assigned to it.

MBXSIZE (189) (c)
MBYSIZE (190) (c)

Sprites can be expanded horizontally with MBXSIZE or vertically with MBYSIZE.

Format: MBXSIZE NR,MD: ... :MBYSIZE NR,MD

NR is the number of the sprite. Values for NR range from 1 to 8.

MD is the control for sprite size. If MD=0, then the sprite appears in normal size. When MD=1, the sprite's size doubles. MBXSIZE expands the sprite horizontally; MBYSIZE expands the sprite vertically.

Example:

MBXSIZE 6,0:MBYSIZE 6,1 sets sprite 6 to normal size horizontally and expands it vertically.

MBGTXSZ (259) (f)
MBGTYSZ (260) (f)

MBGTXSZ returns the horizontal sprite size code, MBGTYSZ returns the vertical sprite size code.

Format: XD = MBGTXSZ (NR): ... :YD = MBGTYSZ (NR)

NR is the number of the desired sprite. Values for NR range from 1 to 8.

XD
YD XD and YD are the size codes. When XD equals 0, the horizontal size is normal. When YD equals 0, the vertical size is normal. If either size is expanded, YD or XD return values of 1.

Example:

```
10 SELECT MBGTYSZ(3)
20 CASE 0:MBYSIZE 3,1:'DOUBLE SIZE'
30 CASE 1:MBYSIZE 3,0:'NORMAL SIZE'
40 ENDSEL
```

This routine sets the size of sprite 3 according to selection (see Chapter 6 for information on the SELECT/ENDSEL commands).

9.2 Positioning and moving sprites

Like normal screen displays, sprites operate on a coordinate system which lets you put a sprite anywhere on the screen. The sprite coordinate system is so accurate that you can put sprites literally anywhere on the visible screen. This means that you can place a sprite so it's only partially visible (off one edge of the border).

The coordinate system originates at the upper left corner of the screen. The X-coordinate (horizontal position) has 512 possible values (0-512); the Y-coordinate (vertical position) has 256 possible values (0-255). The visible screen area for sprites lies between coordinates 24/50 (upper left corner), 344/50 (upper right corner), 344/250 (lower right corner) and 24/250 (lower left corner).

MBSETPOS (144) (c)

MBSETPOS places a sprite at any X- and Y-coordinate on the screen.

Format: MBSETPOS NR,XK,YK

NR is the number of the desired sprite. Values for NR range from 1 to 8.

XK is the horizontal coordinate at which the upper left corner of the sprite should appear - whether that corner is visible or not. Values for XK range from 0 to 511.

YK is the vertical coordinate at which the upper left corner of the sprite should appear - whether that corner is visible or not. Values for YK range from 0 to 255.

Examples:

MBSETPOS 3,150,180 places sprite 3 at the approximate center of the screen.

MBSETPOS 1,400,20 positions sprite 1 to the right of the visible screen.

MBRXPOS (240) (f)
MBRYPOS (241) (f)

These two functions establish the current sprite's coordinates. MBRXPOS returns the horizontal position; MBRYPOS returns the vertical position.

Format: XK = MBRXPOS (NR): ... :YK = MBRYPOS (NR)

NR is the number of the desired sprite. Values for NR range from 1 to 8.

XK is the X-coordinate of the sprite. Values for XK range from 0 to 511.

YK is the Y-coordinate of the sprite. Values for YK range from 0 to 255.

Examples:

A=MBRXPOS(2):B=MBRYPOS(7) list the current horizontal position of sprite 2 in A and the current vertical position of sprite 7 in B.

MBSETPOS 5,MBRXPOS(4),MBRYPOS(4) places sprite number 5 at the same position as sprite 4 so the two sprites are layered.

The MBSETPOS command moves sprites. Movement is nothing more than a more or less continuous replacement of a sprite using the MBSETPOS command. For example, if you wanted to move sprite 1 horizontally across the screen, you'd use a command sequence something like this:

```
10 FOR X=0 TO 511 STEP SP:'MAKE SP A VALUE FROM 1 TO 5'
20 MBSETCOL 1,1:MBON 1:'SEE 9.3 FOR MBON'
30 MBSETPOS 1,X,100
40 NEXT X
```

This short program moves a very primitive sprite across the screen. The step value SP sets the speed of the sprite movement. A value of 1 results in a fairly slow movement, while a value of 5 moves the sprite very quickly across the screen.

You can fine-tune the speed using a blank FOR/NEXT loop (e.g., FOR I=1 TO 25:NEXT I). See Chapter 6 for information about loop construction, particularly FOR/NEXT loops.

A combination of MBSETPOS and loops will be enough for most users, since this command is very flexible when compared to the normal C64 sprite commands.

Take a look at the sprite demonstration program, stored on your distribution diskette under the name SPRITEDEMO. When you load and run this short game program, use the <Cursor up> and <Cursor down> keys to move the paddle up and down the screen. The object of the game is to catch the ball with the paddle. Your score appears on the upper left corner of the screen. Press the <RETURN> key to end the game at any time.

9.3 Enabling and disabling sprites

MBON (145) (c)
MBGTON (261) (f)

The MBON command turns the corresponding sprite on. The MBGTON command tells whether a sprite is on or off.

Format: MBON NR: ... :MD = MBGTON (NR)

NR is the number of the desired sprite. Values for NR range from 1 to 8.

MD is the status number of the desired sprite. If MD=1, the sprite is visible on the screen. If MD=0, the sprite is inactive.

Example:

MBON 3:MBON 7:WA=MBGTON(3) enables sprites 3 and 7 and returns the status of sprite 3 (WA=1).

MBOFF (146) (c)
MBALLOFF (147) (c)

The MBOFF command turns the corresponding sprite off. MBALLOFF removes all active sprites from the screen.

Format: MBOFF NR: ... :MBALLOFF

NR is the number of the desired sprite. Values for NR range from 1 to 8.

Example:

MBON 2:MBON 8:MBON 5: ... :MBOFF 8: ... :MBALLOFF turns on sprites 2, 5 and 8 on the screen. MBOFF 8 turns sprite 8 off, then MBALLOFF removes the rest of the sprites.

9.4 Loading and saving sprite data blocks

MBDSAVE	(191)	(c)
MBDLOAD	(192)	(c)

MBDSAVE saves the indicated 64-byte block to diskette. MBDLOAD loads sprite data into any given sprite block in memory.

Format: MBDSAVE NA$,BL: ... :MBDLOAD NA$,BL

NA$ is the filename under which the data is loaded/saved. This filename can be up to 16 characters long.

BL is the corresponding data block number (see also MBBLOCK).

NOTE: When you save or load several data blocks at once, use the DSAVEM or DLOADM commands (see Section 5.3) since these commands can handle any memory size.

Example:

MBDSAVE "SPRBL",13 saves data block 13 to diskette under the name SPRBL.

MBDLOAD "SPRBL",553 loads this same sprite data into memory block 553.

9.5 Testing for collisions

The following functions operate in conjunction with the commands used for checking sprite priority. Collisions occur in the visible screen area between sprites. These collisions set the appropriate sprite matrix bits to 1.

NOTE: The VIC collision register is designed so that reading the register clears the register. The functions below store the codes in variable memory for later reading.

This is the reason you can't do multiple readings of MBCHECKS or MBCHECKG on multiple sprite collision; all eight sprites use one register for checking sprite/sprite or sprite/background collision. Use the functions MBCHECKALLS and MBCHECKALLG for multiple reading.

MBCHECKS (242) (f)

This command reads whether a sprite collides with another.

Format: MD = MBCHECKS (NR)

NR is the number of the desired sprite. Values for NR range from 1 to 8.

MD lists the number of the crashed sprite. Otherwise this value is 0.

Example:

A=MBCHECKS(7) tells whether sprite 7 collided with one of the other seven sprites. If so, A=1; otherwise A=0.

MBCHECKALLS (263) (f)

If you want to convey a collision between sprites, use the MBCHECKALLS function.

Format: MD = MBCHECKALLS

All eight sprites return values according to the following table:

Sprite number:	1	2	3	4	5	6	7	8
Value:	1	2	4	8	16	32	64	128

The total value of MD is the sum of the individual values. If, for example, sprites 2, 3 and 5 collide, MD returns 22.

Examples:

MD=9 means that sprites 1 and 4 collided.

MD=212 means that sprites 3,5,7 and 8 collided.

MD=0 means that no collision occurred since the last reading.

MBCHECKG (243) (f)

This command looks for a collision between a certain sprite and a screen character or hi-res graphic.

Format: MD = MBCHECKG (NR)

NR is the number of the desired sprite. Values for NR range from 1 to 8.

MD defaults to 0 if no collision occurs. Collisions change this value to 1.

Example:

HG=MBCHECKG(2) tells whether sprite 2 has collided with a background character. If so, HG becomes equal to 1; otherwise HG remains at 0.

MBCHECKALLG (262) (f)

Similar to sprite/sprite collision reading, except that all sprites are checked for background collision.

Format: MD = MBCHECKALLG

All eight sprites return values according to the following table:

Sprite number:	1	2	3	4	5	6	7	8
Value:	1	2	4	8	16	32	64	128

The total value of MD is the sum of the individual values. If, for example, sprites 2, 3 and 5 collide, MD returns 22.

Examples:

MD=32 means that sprite 6 has hit either a character or part of a hi-res graphic.

MD=26 signals that sprites 2, 4 and 5 have collided with the background.

MBDELCOLL (193) (c)

This command clears both VIC registers for sprite/sprite and sprite/background collisions. Use this command at the beginning of a program so false readings left from previous programs are not encountered.

Format: MBDELCOLL

9.6 The BeckerBASIC sprite editor routine

The program listed here is included on your BeckerBASIC distribution diskette under the name SPRITE-EDIT. It allows you to create sprites on the screen. The resulting values are computed in the editor and stored in the string variable MT$, so you can read them with MBDESIGN.

To start the editor you need the instruction GOTO "MATDEF/S". You can now use your cursor keys to move around an enlarged 21 x 24 matrix. Press the <F1> key to set a pixel; press the <F3> key to erase a pixel. Press <F7> to accept the completed matrix, or <F8> rejects the matrix. Accepted sprites load into the variable MT$.

After you define the sprite matrix, the editor allows you to load stored sprite matrices into memory for editing, or clear the matrix to use new sprites. Load the matrix into MT$ (using MBDATA, for example) and call the editor with GOTO "MATDAR/S".

You can change the position of the editor's matrix area by changing the value of the variable SW. This variable always contains the upper left corner of the editor.

To move the editor to the center of the screen, for example, you need to set SW to PEEK(648)*256+88 (see lines 1080,1250 and 1580).

When you call the editor as a subroutine (e.g., GOSUB "MATDEF/S" or GOSUB "MATDAR/S"), you must change the END in line 1690 to RETURN.

The branches in the editor use labels, so you can change line numbers (with PRENUMBER, see Section 2.1.1) for merging. The structured design of the routine allows easy addition of new functions (e.g., mirroring the matrix).

```
100 'AFTER MERGING TO ANOTHER PROGRAM, DELETE THESE LINES'
110 :
115 CLS:LETTERON:CRCOL 1
120 SCPRINT AT 3,1;"THIS PROGRAM CANNOT START ON ITS OWN!"
130 SCPRINT AT 6,1;"IT IS INTENDED TO BE INTEGRATED"
140 SCPRINT AT 7,1;"WITH  OTHER PROGRAMS YOU"
150 SCPRINT AT 8,1;"HAVE WRITTEN."
160 SCPRINT AT 17,6;"PLEASE PRESS A KEY."
165 WAITKEYA:END
170 :
180 :
190 :
1000 'SPRITE-EDITOR'
1010 '(C) 1986 BY MARTIN HECHT'
1020 :
1030 'SAVED ON DISKETTE UNDER THE NAME SPRITE-EDIT'
1040 :
1050 :
1060 "MATDAR/S":'DISPLAY MATRIX'
1070 :
1080 CLS:SW=PEEK(648)*256:'HOME POSITION=START OF MATRIX'
1090 FOR Z1=0 TO 60 STEP3
1100 : FOR Z2=0 TO 2
1110 :   AW=ASC(MID$(MT$,Z1+Z2+1,1))
1120 :   FOR Z3=0 TO 7
```

```
1130 :      AW=AW/2
1140 :      IF AW=INT(AW) THEN PW=46:ELSE PW=160:ENDIF
1150 :      POKE SW+Z1/3*40+Z2*8+7-Z3,PW:AW=INT(AW)
1160 :   NEXT Z3
1170 : NEXT Z2
1180 NEXT Z1
1190 GOTO "MATDEFINP/S":'MATRIX INPUT CONTROL'
1200 :
1210 :
1220 "MATDEF/S":'DEFINE MATRIX'
1230 :
1240 'DRAW PATTERN ARRAY'
1250 CLS:SW=PEEK(648)*256
1260 FOR Z1=0 TO 20
1270 : HV=SW+Z1*40:MYFILL HV,HV+23,46
1280 NEXT Z1
1290 :
1300 "MATDEFINP/S":'INPUT CONTROL'
1310 H=SW-PEEK(648)*256:PZ=INT(H/40)+1:PS=H-(PZ-
1)*40+1:'STARTPOSITION'
1320 ZE=PZ:SP=PS:'CURSOR IN STARTPOSITION'
1330 "MATINPUT/S":CRSET ZE,SP:CRON:WAITKEYA:GET EG$:CROFF
1340 SELECT ASC(EG$):'CURRENT INPUT'
1350 CASE 29:'CURSOR RIGHT'
1360 "CR":IF NOT(SP=PS+23) THEN SP=SP+1
1370 ELSE IF NOT(ZE=PZ+20) THEN SP=PS:ZE=ZE+1:ENDIF
1375 ENDIF
1380 CASE 157:'CURSOR LEFT'
1390 IF NOT(SP=PS) THEN SP=SP-1
1400 ELSE IF NOT(ZE=PZ) THEN SP=PS+23:ZE=ZE-1:ENDIF
1410 ENDIF
1420 CASE 17:'CURSOR DOWN'
1430 IF NOT(ZE=PZ+20) THEN ZE=ZE+1:ENDIF
1440 CASE 145:'CURSOR UP'
1450 IF NOT(ZE=PZ) THEN ZE=ZE-1:ENDIF
1460 CASE 133:'F1=SET PIXEL'
1470 POKE PEEK(648)*256+40*(ZE-1)+SP-1,160:GOTO "CR"
1480 CASE 134:'F3=DELETE PIXEL'
1490 POKE PEEK(648)*256+40*(ZE-1)+SP-1,46:GOTO "CR"
1500 CASE 136:'F7=ACCEPT MATRIX'
1510 GOTO "UEBERNAHME/S"
1520 CASE 140:'F8=CANCEL'
1530 GOTO "ENDE"
1540 ENDSEL
1550 GOTO "MATINPUT/S"
1560 :
1570 "UEBERNAHME/S":'ACCEPT MATRIX'
1580 SW=PEEK(648)*256:AW=0:MT$=""
```

```
1590 FOR Z1=0 TO 60 STEP3
1600 : FOR Z2=0 TO 2
1610 :    FOR Z3=0 TO 7
1620 :      PW=PEEK(SW+Z1/3*40+Z2*8+Z3)
1630 :      IF PW=160 THEN AW=AW+2^(7-Z3):ENDIF
1640 :    NEXT Z3
1650 :    MT$=MT$+CHR$(AW):AW=0
1660 : NEXT Z2
1670 NEXT Z1
1680 :
1690 "ENDE":END:'IF SUBROUTINE,THEN REPLACE WITH RETURN'
```

10. Sound commands

You don't have to be a musician to use BeckerBASIC sound commands. Along with sound effects, the SID chip can perform anything from a simple keyboard beep up to complete synthesized sounds. Sound is for everyone, and BeckerBASIC has special commands that let anyone create sounds or program musical pieces. BeckerBASIC sound commands regulate the SID chip without lots of POKE commands.

In many cases, you'll only need a few commands to program sound effects. You can take the programs on the next few pages and re-use them in other programs as procedures (see Chapter 6).

Each section of this chapter examines a different aspect of the sound chip. Section 10.1 takes you through the basics of making single notes. Section 10.2 shows you how to turn notes on and off. The last two sections demonstrate synchronization, filters and ring modulation.

NOTE: that the commands and functions listed in this chapter are not accessible in the Input System. You wil have to switch to the Testing System with <CTRL><Commodore> to test your program.

10.1 Making sounds

The C64 sound chip (the SID chip) has a total of three tone generators, which produce three voices.

SDCLEAR (148) (c)

The SDCLEAR command initializes (resets) the sound chip. This command can be used at the beginning or end of a program to reset all sound registers to their normal states.

Format: SDCLEAR

SDVOLUME (149) (c)

The SDVOLUME command sets the volume for all three voices.

Format: SDVOLUME VL

VL is the volume level. Values for VL range from 0 to 15.

NOTE: You'll use 15 as a volume level most of the time.

SDFREQUENCY (150) (c)

This command sets the frequency of a voice in Hertz.

Format: SDFREQUENCY VC,FR

VC is the number of the desired voice. Values for VC range from 1 to 3. Numbers outside this range result in an ILLEGAL QUANTITY ERROR.

FR is the frequency of the note. Values for FR range from 0 to 3848. You can include decimals after a decimal point for fine tuning of up to 1/17 Hertz).

Example:

SDFREQUENCY 1,1497.2:SDFREQUENCY 3,2850 sets voice 1 to 1497.2 Hz and voice 3 to 2850 Hz.

SDNOTE (151) (c)

When you find it impractical to use the SDFREQUENCY command, especially when you're programming musical compositions, SDNOTE offers a more comfortable method of note input.

Format: SDNOTE VC,NT$

VC is the number of the desired voice. Values for VC range from 1 to 3.

NT$ contains the characters indicating the note you want played, as well as the desired octave. This sequence reads NOTEOCTAVE. NOTE names use normal letter names from A to G. Values for OCTAVE range from 0 to 7. The SDNOTE command translates the note and octave into the equivalent frequency.

```
10 RESTORE:SDCLEAR:SDVOLUME 15
15 NR=1:'VOICE'
20 SDWAVEON NR,1:SDENVELOPE NR,0,0,15,0:SDVOICEON NR
25 LOOP
30 READ NT$:LPEXITIF NT$="DONE"
35 SDNOTE NR,NT$:FOR I=1 TO 200:NEXT I
40 ENDLOOP
45 SDVOICEOFF NR:END
50 :
100 DATA C0,C#0,D0,D#0,E0,F0,F#0,G0,G#0,A0,A#0,B0
110 DATA C1,C#1,D1,D#1,E1,F1,F#1,G1,G#1,A1,A#1,B1
120 DATA C2,C#2,D2,D#2,E2,F2,F#2,G2,G#2,A2,A#2,B2
130 DATA C3,C#3,D3,D#3,E3,F3,F#3,G3,G#3,A3,A#3,B3
140 DATA C4,C#4,D4,D#4,E4,F4,F#4,G4,G#4,A4,A#4,B4
150 DATA C5,C#5,D5,D#5,E5,F5,F#5,G5,G#5,A5,A#5,B5
160 DATA C6,C#6,D6,D#6,E6,F6,F#6,G6,G#6,A6,A#6,B6
170 DATA C7,C#7,D7,D#7,E7,F7,F#7,G7,G#7,A7,A#7
180 DATA DONE
```

This program plays the complete SID range on voice 1, and contains a complete list of the notes and octaves accessible to the SID chip (look at the DATA statements). Please note that the scale stops at the top A#, since that is the highest note of the SID chip's range.

SDWAVEON (153) (c)
SDWAVEOFF (154) (c)

These commands control the waveform creation for individual voices. SDWAVEON turns the appropriate waveform or waveforms on (you can activate several waveforms at once with SDWAVEON). The SDWAVEOFF command turns a waveform off.

Format: SDWAVEON VC, WF [,PW]: ... :SDWAVEOFF VC,WF

VC is the desired SID chip voice. Values for VC range from 1 to 3.

WF is the identifier for the desired waveform. You have four waveforms available:

1	triangle
2	sawtooth
3	pulse
4	noise

PW is an additional parameter which controls the width of the pulse wave (WF=3) as a percentage. Values for PW can range from 0 to 100, with provisions for decimal places. This parameter has no effect an any other waveform.

Examples:

SDWAVEON 2,1 assigns a triangle wave to voice 2.

SDWAVEON 1,2:SDWAVEON 1,3,40 assigns a sawtooth wave and pulse wave to voice 1; the pulse wave receives a level of 40 percent.

SDWAVEON 3,4: ... :SDWAVEOFF 3,4:SDWAVEON 3,1 assigns a noise wave to voice 3, then changes that voice to a triangle wave.

10.2 Turning voices on and off

SDENVELOPE (152) (c)

This command gives you the power to create a *software envelope* which controls the *attack* (start), *decay* (dying out), *sustain* (hold) and *release* (end) of a note.

Format: SDENVELOPE VC,A,D,S,R

VC is the desired voice number. Values for VC range from 1 to 3.

A controls the attack (starting) phase of a note. Values for A range from 0 to 15 (maximum volume).

D controls the decay phase of a note, when the sound dies out. Values for D range from 0 to 15.

S controls the sustain phase of a note (when a note is held). Values for S range from 0 to 15. S determines its parameter based upon the attack phase maximum of 15, instead of basing itself on time. If, for example, S=15, the volume reaches its maximum during the attack phase. A value of 0 for S means that the tone holds until the decay phase begins. The increments from S are linear. That is, if S=7, the volume drops to about half the value of the attack phase.

R controls the release (ending) phase of a note. Values for R range from 0 to 15.

The table below contains the individual values for time (sc=seconds, ms=milliseconds) and parameters:

Parameter value	Attack (a)	Decay (d) / Release (r)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	36 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 sc
11	800 ms	2.4 sc
12	1 sc	3 sc
13	3 sc	9 sc
14	5 sc	15 sc
15	8 sc	24 sc

See the descriptions of SDVOICEON and SDVOICEOFF for practical examples of SDENVELOPE.

SDVOICEON (155) (c)
SDVOICEOFF (156) (c)

These commands let you enable or disable any one of the voices at any time.

Format: SDVOICEON VC: ... :SDVOICEOFF VC

VC is the number of the desired voice. Values for VC can range from 1 to 3. The desired voice is audible only if parameters such as volume, waveform, etc. are set before turning the voice on. The frequency of a tone can be changed while the voice is on.

When you set up an envelope with the SDENVELOPE command (see above), SDVOICEON uses that envelope for its sound parameters. The attack executes, then the decay and sustain phases run. The release phase actually holds the tone until the SDVOICEOFF command disables the voice. As long as the voice is on, the tone continues (see below for a concrete example). You can use PAUSE to sustain notes for certain periods of time (see Chapter 3). SDVOICEON 2:PAUSE4:SDVOICEOFF 2 turns on voice 2 for exactly 4 seconds. You can fine-tune this timing by using a FOR/NEXT loop instead (e.g., FOR I=1 TO 100:NEXT I).

Examples:

The example below sounds a standard signal tone that you might use for audible errors, etc.

```
1000 NR=1:'VOICE'
1005 SDCLEAR:'INITIALIZATION':SDVOLUME 15:'VOL'
1010 SDFREQUENCY NR,500:'500 HERTZ FREQUENCY'
1015 SDWAVEON NR,1:'TRIANGLE WAVE'
1020 SDENVELOPE NR,0,0,15,0:'ENVELOPE'
1025 SDVOICEON NR:'VOICE ON':PAUSE 3
1030 SDVOICEOFF NR:'VOICE OFF'
1035 END
```

This example uses the noise waveform.

```
1100 NR=1:'VOICE'
1105 SDCLEAR:'INITIALIZATION':SDVOLUME 15:'VOL'
1110 SDFREQUENCY NR,3250:'3250 HERTZ FREQUENCY'
1115 SDWAVEON NR,4:'NOISE WAVE'
```

```
1120 SDENVELOPE NR,0,9,0,0:'ENVELOPE'
1125 SDVOICEON NR:'VOICE ON':PAUSE 1
1130 SDVOICEOFF NR:'VOICE OFF'
1135 END
```

This program demonstrates two different sounds: a flute and an oboe.

```
1200 'FLUTE'
1203 CLS:SCPRINT AT 7,17;"FLUTE "
1205 NR=1:'VOICE'
1210 SDCLEAR:'INITIALIZATION':SDVOLUME 15:'VOL'
1215 SDFREQUENCY NR,600:'600 HERTZ FREQUENCY'
1220 SDWAVEON NR,1:'TRIANGLE WAVE'
1225 SDENVELOPE NR,8,5,15,8:'ENVELOPE'
1226 SCPRINT AT 10,10;"PLEASE PRESS A KEY."
1230 SDVOICEON NR:'VOICE ON':WAITKEYA:'WAIT FOR KEYPRESS'
1235 SDVOICEOFF NR:'VOICE OFF'
1300 'OBOE'
1303 CLS:SCPRINT AT 7,17;"OBOE  "
1305 NR=1:'VOICE'
1310 SDCLEAR:'INITIALIZATION':SDVOLUME 15:'VOL'
1315 SDFREQUENCY NR,450:'450 HERTZ FREQUENCY'
1320 SDWAVEON NR,3,6.11:'PULSE WAVE, WIDTH 6.11%'
1325 SDENVELOPE NR,4,9,15,8:'ENVELOPE'
1326 SCPRINT AT 10,7;"PLEASE PRESS A KEY TO END."
1330 SDVOICEON NR:'VOICE ON':WAITKEYA:'WAIT FOR KEYPRESS'
1335 SDVOICEOFF NR:'VOICE OFF'
1340 END
```

10.3 Filters

The sound chip can alter voice qualities using a filter. This filter is like the tone control on your stereo system. One common filter affects all three voices, but you can state which voices are filtered and which voices come through "straight."

SDFILTER (157) (c)

This command sets up the filter parameters. The numbers below set the operating mode and resonance of the filter.

Format: SDFILTER FQ,FA,RS

FQ is the top, cutoff or middle frequency, setting the frequency at which the voice should be filtered. Values for FQ range from 30 to 11800, and represent Hertz (cycles per second). Values outside this range result in an ILLEGAL QUANTITY ERROR.

FA sets the filter operating mode. FA=1 sets up a high-pass filter, affecting only the frequencies above the frequency FQ. FA=2 sets up a low-pass filter, which affects frequencies below the frequency FQ. FA=3 activates a bandpass filter, which affects frequencies in the area of FQ. FA=4 enables parallel switching of high-pass and low-pass filters, called a *notch filter*.

RS sets the resonance of the filter. Values for RS range from 0 to 15. This parameter adds richness to the frequencies in the area of FQ. RS=0 causes minimal resonance, while RS=15 results in maximum resonance.

SDVCFTON (158) (c)
SDVCFTOFF (159) (c)

These commands set a specific voice for filtering.

Format: SDVCFTON VC: ... :SDVCFTOFF VC

VC is the number of the tone generator. Values for VC can range from 1 to 4 (!). Voice 4 represents the audio input.

SDVCFTON assigns a voice to the filter. SDVCFTOFF removes a voice from filtering (SDVCFTOFF is the default).

Like all other SID parameters, you can turn filtering on or off at any time.

Examples:

This example creates an explosion.

```
2000 NR=1:'VOICE'
2005 SDCLEAR:'INITIALIZATION':SDVOLUME 15:'VOL'
2010 SDFREQUENCY NR,500:'500 HERTZ FREQUENCY'
2015 SDWAVEON NR,4:'NOISE WAVE'
```

```
2020 SDENVELOPE NR,0,11,3,12:'ENVELOPE'
2025 'SET FILTER TO HIGHPASS, 500 HERTZ FREQUENCY, 12 RESONANCE'
2030 SDFILTER 1,500,12:SDVCFTON NR:'ASSIGN VOICE TO FILTER'
2035 SDVOICEON NR:PAUSE2
2040 SDVOICEOFF NR:SDVCFTOFF NR
2045 END
```

The program below imitates a banjo.

```
2100 NR=1:'VOICE'
2105 SDCLEAR:'INITIALIZATION':SDVOLUME 15:'VOL'
2110 SDFREQUENCY NR,450:'450 HERTZ FREQUENCY'
2115 SDWAVEON NR,2:'SAWTOOTH WAVE'
2120 SDENVELOPE NR,0,9,0,0:'ENVELOPE'
2125 'FILTER:NOTCH, 2000 HZ FREQUENCY, RESONANCE 15'
2130 SDFILTER 4,2000,15:SDVCFTON NR:'VOICE INTO FILTER'
2135 SDVOICEON NR:PAUSE 1
2140 SDVOICEOFF NR:SDVCFTOFF NR
2145 END
```

10.4 Synchronization and ring modulation

SDSYNCHRON (160) (c)
SDSYNCHROFF (161) (c)

These commands allow synchronization between one voice and another.

Format: SDSYNCHRON VC: ... :SDSYNCHROFF VC

VC is the number of the voice set for synchronization. The voice used for synchronizing VC is hardware-set. VC=1 means that voice 1 is synchronized with voice 3; VC=2 synchronizes voice 1 with voice 2; and VC=3 synchronizes voice 2 with voice 3.

SDSYNCHRON activates syncrhonization; SDSYNCHROFF turns it off at any time. You can have multiple synchronization. Example:

SDSYNCHRON 1:SDSYNCHRON 2 synchronizes both voice 1 and voice 2.

SDRINGMODON (162) (c)
SDRINGMODOFF (163) (c)

These commands control ring modulation between two voices.

Format: SDRINGMODON VC: ... :SDRINGMODOFF VC

VC is the number of the voice to be combined with a second voice for ring modulation. VC=1 creates ring modulation between voices 1 and 3; VC=2 creates ring modulation between voices 2 and 1; VC=3 creates ring modulation between voices 3 and 2.

SDRINGMODON turns modulation on; SDRINGMODOFF turns it off. Multiple ring modulation can also be produced.

Example:

SDRINGMODON 1:SDRINGMODON 3 puts all three voices in ring modulation mode. The first command creates modulation between voices 1 and 3; the second produces modulation between voices 3 and 2.

NOTE: Waveforms must be set to triangle wave before you can get an audible ring modulation.

SDVOICE3OFF (164) (c)
SDVOICE3ON (163) (c)

Synchronized or ring modulated voices are normally audible. If you use voice 3, you can make it "inaudible" with SDVOICE3OFF. (Because of hardware design, this option exists only for the third voice).

Format: SDVOICE3OFF: ... :SDVOICE3ON

When SDVOICE3OFF is used in concert with a synchronization, or ring modulation, only voice 1 is audible. SDVOICE3ON restores the normal status (audible third voice).

Examples:

Both examples make siren noises using BeckerBASIC SID chip commands in different ways.

```
3000 'SYNCHRONIZATION'
3005 N1=1:N2=3:'VOICES'
3010 SDCLEAR:SDVOLUME 15
3015 SDENVELOPE N1,0,0,15,0:'ENVELOPE'
3020 SDWAVEON N1,3,40:'PULSE WAVE':SDSYNCHRON N1
3025 SDWAVEON N2,2:'SAWTOOTH WAVE':SDVOICE3OFF
3030 SDFREQUENCY N2,150:'150 HZ FREQUENCY'
3035 SDVOICEON N1:SDVOICEON N2
3040 FOR I=100 TO 2000 STEP 3
3045 SDFREQUENCY N1,I
3050 NEXT I
3055 SDCLEAR:END

3100 'RING MODULATION'
3105 N1=1:N2=3:'VOICES'
3110 SDCLEAR:SDVOLUME 15
3115 SDENVELOPE N1,0,0,15,0:SDENVELOPE N2,0,0,15,0
3120 SDWAVEON N1,1:SDWAVEON N2,3,40:SDRINGMODON N1
3125 SDVOICEON N1:SDVOICEON N2
3130 'MOVE FREQ'
3135 FOR I=1 TO 2000 STEP 100
3140 FOR X=1 TO 2000 STEP 50
3145 SDFREQUENCY N2,X
3150 NEXT X
3155 SDFREQUENCY N1,I
3160 NEXT I
3165 SDCLEAR:END
```

Appendix A: Commands and functions listed by number

The following is a complete list of BeckerBASIC commands and functions. You can find command and function numbers by using the COMNUM command (see Chapter 1).

This book allows space after the original name for your own defined command names. This table is intended to help you quickly find information about BeckerBASIC commands.

The far right column of the table is the page reference for each command or function. When you want to find a command, the procedure is as follows: Find the command number on the screen using COMNUM, then look up the command number in the table.

For example, SCPRINT COMNUM("TRACE") returns a value of 32. Look in the table for the page number of command number 32. Other references may appear in this handbook under the main reference.

NOTE: Command or function numbers preceded by an asterisk (*) are available in all three interpreters - Input, Testing and Run-Only. A number sign (#) denotes a command or function available only from the Input-System. If no character precedes the command number, the command/function can be accessed only be the Testing and Run-Only-Systems, and not by the Input-System.

Commands

Number	Original name	New name	Page
* 001	GOTO	_____	103
* 002	GOSUB	_____	103
* 003	RESTORE	_____	103
# 004	LIST	_____	20
* 005	RUN	_____	103
* 006	TRON	_____	35
* 007	PAUSE	_____	21
* 008	DESKTOP	_____	22
# 009	PAUTO	_____	23
# 010	PRENUMBER	_____	24
# 011	POLD	_____	26
# 012	PMERGE	_____	25
# 013	PDEL	_____	25
* 014	PBCEND	_____	26
# 015	PMEM	_____	26
# 016	PDFKEY	_____	27
# 017	PKEY	_____	28
# 018	PCOLORS	_____	49

Number	Original name	New name	Page
# 019	PHELP	_____	13
# 020	NEWCOMTAB	_____	12
# 021	OLDCOMTAB	_____	12
# 022	RENCOM	_____	15
# 023	DSCOMTAB	_____	17
# 024	DLCOMTAB	_____	17
* 025	ONERRORGO	_____	30
* 026	ONERROROFF	_____	30
* 027	RESUMECUR	_____	30
* 028	RESUMENEXT	_____	31
* 029	RESUME	_____	31
* 030	ERRSHOWON	_____	29
* 031	ERRSHOWOFF	_____	29
* 032	TRACE	_____	34
* 033	KEYREPEATON	_____	37
* 034	KEYREPEATOFF	_____	37
* 035	STOPON	_____	38
* 036	STOPOF	_____	38
* 037	WAITKEYA	_____	38

Number	Original name	New name	Page
* 038	WAITKEYS	_____	39
* 039	KGETV	_____	39
* 040	KBGETV	_____	40
* 041	ONKEYGO	_____	42
* 042	RETKEY	_____	43
* 043	SGETV	_____	44
* 044	SGETM	_____	44
* 045	DGETV	_____	80
* 046	DGETM	_____	80
* 047	SCPRINT	_____	46
* 048	AT	_____	46
* 049	RVSON	_____	47
* 050	RVSOFF	_____	47
* 051	BORDER	_____	49
* 052	GROUND	_____	50
* 053	CLS	_____	50
* 054	SCRON	_____	50
* 055	SCROFF	_____	50
* 056	SCRDSAVE	_____	51

Number	Original name	New name	Page
* 057	SCRDLOAD	_____	51
* 058	CRHOME	_____	51
* 059	CRSET	_____	51
* 060	CRCOL	_____	52
* 061	CRON	_____	52
* 062	CRREPEATON	_____	52
* 063	CRREPEATOFF	_____	52
* 064	CRFREQ	_____	52
* 065	TRANSFER	_____	55
* 066	DOKE	_____	58
* 067	MYFILL	_____	56
* 068	MGETV	_____	60
* 069	ASCBSCW	_____	57
* 070	BSCASCW	_____	57
* 071	SWAP	_____	21
* 072	DIR	_____	64
* 073	DSTATUS	_____	65
* 074	DSENDCOM	_____	65
* 075	DHEADER	_____	67

Number	Original name	New name	Page
* 076	DINIT	_____	67
* 077	DRESET	_____	68
* 078	DRENAME	_____	66
* 079	DSCRATCH	_____	66
* 080	DOPEN	_____	78
* 081	DCLOSE	_____	82
* 082	DSAVEB	_____	70
* 083	DSAVEM	_____	72
* 084	DCSAVEB	_____	70
* 085	DCSAVEM	_____	72
* 086	DVERIFYB	_____	72
* 087	DVERIFYM	_____	73
* 088	DLOADB	_____	74
* 089	DLOADM	_____	74
* 090	DLOADAM	_____	74
* 091	DRLOADB	_____	74
* 092	DOVERLAYK	_____	75
* 093	DOVERLAYW	_____	75
* 094	DSQOPEN	_____	82

Number	Original name	New name	Page
* 095	DSQCONCAT	_____	84
* 096	DRLOPEN	_____	84
* 097	DRLRECORD	_____	86
* 098	DUSOPEN	_____	88
* 099	DDAOPEN	_____	89
* 100	DDAPOINT	_____	92
* 101	DDAREADBL	_____	91
* 102	DDAWRITEBL	_____	92
* 103	DDABLALLOC	_____	93
* 104	DDABLFREE	_____	93
* 105	DDABLEXEC	_____	98
* 106	DMYPOKE	_____	96
* 107	DMYWRITEM	_____	97
* 108	DMYWRITEV	_____	97
* 109	DMYEXEC	_____	98
* 110	IF	_____	105
* 111	THEN	_____	105
* 112	ELSE	_____	105
* 113	ENDIF	_____	105

Number	Original name	New name	Page
* 114	WHILE	_____	110
* 115	DO	_____	110
* 116	ENDDO	_____	110
* 117	REPEAT	_____	111
* 118	UNTIL	_____	111
* 119	LOOP	_____	113
* 120	LPEXITIF	_____	113
* 121	ENDLOOP	_____	113
* 122	SELECT	_____	107
* 123	CASE	_____	107
* 124	OTHER	_____	107
* 125	ENDSEL	_____	107
* 126	PROCEDURE	_____	115
* 127	PROCEND	_____	115
* 128	CALL	_____	115
* 129	DSAVEPROC	_____	121
* 130	DLOADPROC	_____	121
* 131	DELPROC	_____	122
* 132	LDEL	_____	77

Number	Original name	New name	Page
* 133	LETTERON	_____	47
* 134	LETTEROFF	_____	47
* 135	LOCKON	_____	47
* 136	LOCKOFF	_____	47
137	HRON	_____	144
138	HROFF	_____	144
* 139	MBDESIGN	_____	154
* 140	MBINV	_____	155
* 141	MBBLOCK	_____	157
* 142	MBEXCOL	_____	159
* 143	MBSETCOL	_____	158
* 144	MBSETPOS	_____	162
* 145	MBON	_____	164
* 146	MBOFF	_____	164
* 147	MBALLOFF	_____	164
148	SDCLEAR	_____	173
149	SDVOLUME	_____	174
150	SDFREQUENCY	_____	174
151	SDNOTE	_____	174

Number	Original name	New name	Page
152	SDENVELOPE	_____	176
153	SDWAVEON	_____	175
154	SDWAVEOFF	_____	175
155	SDVOICEON	_____	178
156	SDVOICEOFF	_____	178
157	SDFILTER	_____	179
158	SDVCFTON	_____	180
159	SDVCFTOFF	_____	180
160	SDSYNCHRON	_____	181
161	SDSYNCHROFF	_____	181
162	SDRINGMODON	_____	182
163	SDRINGMODOFF	_____	182
164	SDVOICE3ON	_____	182
165	SDVOICE3OFF	_____	182
* 166	ONKEYOFF	_____	43
* 167	TROFF	_____	35
* 168	CROFF	_____	52
* 169	MBPRIOR	_____	160
* 170	PRLIST	_____	20

Number	Original name	New name	Page
* 171	PRPRINT	_____	48
* 172	PRCOM	_____	48
* 173	DB	_____	19
* 174	ON	_____	103
* 175	RESET	_____	22
* 176	KEYDEL	_____	38
* 177	NEW	_____	22
* 178	WAITST	_____	41
* 179	PFKEYON	_____	28
* 180	PFKEYOFF	_____	28
* 181	VGETM	_____	60
* 182	MBCLR	_____	155
* 183	MBMOVE	_____	156
* 184	MBCHANGE	_____	156
* 185	MBAND	_____	156
* 186	MBOR	_____	156
* 187	MBEOR	_____	156
* 188	MBMODE	_____	158
* 189	MBXSIZE	_____	160

Number	Original name	New name	Page
* 190	MBYSIZE	_____	160
* 191	MBDSAVE	_____	165
* 192	MBDLOAD	_____	165
* 193	MBDELCOLL	_____	168
* 194	DADRCHANGE	_____	68
* 195	DKDEVNB	_____	68
* 196	DPGOPEN	_____	88
* 197	DSAVEL	_____	71
* 198	DCSAVEL	_____	71
* 199	DVERIFYAM	_____	73
* 200	DRLCLOSE	_____	85
* 201	DMYREADM	_____	96
* 202	DMYREADV	_____	95
* 203	PDUMP	_____	27
* 204	POPREP	_____	113
* 205	POPWHL	_____	111
* 206	POPPROC	_____	120
* 207	POPLP	_____	114
* 208	POPIF	_____	107

Number	Original name	New name	Page
# 209	TABNAME	_____	18
210	PDMENU	_____	132
211	GEOSON	_____	130
212	GEOSOFF	_____	130
213	DIALOGBOX	_____	135
214	HRPRINT	_____	138
215	GEOSASCW	_____	140
216	ASCGEOSW	_____	140
217	HRGET	_____	141
218	HRGDCOL	_____	144
219	HRPTCOL	_____	144
220	HRDEL	_____	144
221	HRINV	_____	149
222	HRDLOAD	_____	151
223	HRDSAVE	_____	151
224	HRLINE	_____	146
225	HRHLINE	_____	146
226	HRVLINE	_____	146
227	HRBOX	_____	148

Number	Original name	New name	Page
228	HRFRAME	_____	147
229	HRPLOT	_____	145
230	HRSTRING	_____	149

Functions

Number	Original name	New name	Page
# 231	COMNUM	_____	14
* 232	STTEST	_____	41
* 233	CRPOSL	_____	51
* 234	CRPOSC	_____	51
* 235	DEEK	_____	58
* 236	TEEK	_____	59
* 237	VARADR	_____	61
* 238	EOF	_____	81
* 239	DMYPEEK	_____	95
* 240	MBRXPOS	_____	163
* 241	MBRYPOS	_____	163
* 242	MBCHECKS	_____	166
* 243	MBCHECKG	_____	167
* 244	DF	_____	19
* 245	CLGROUND	_____	50
* 246	CLBORDER	_____	49
* 247	CLCURSOR	_____	52
* 248	MBDATA	_____	154

Number	Original name	New name	Page
* 249	GTBCEND	_____	26
# 250	COMTAB	_____	13
* 251	COMNAME	_____	15
* 252	FILENUM	_____	80
* 253	DDEVADR	_____	68
* 254	MBGTBLK	_____	157
* 255	MBGTEXCL	_____	159
* 256	MBGTCOL	_____	158
* 257	MBGTMOD	_____	158
* 258	MBGTPR	_____	160
* 259	MBGTXSZ	_____	161
* 260	MBGTYSZ	_____	161
* 261	MBGTON	_____	164
* 262	MBCHECKALLG	_____	167
* 263	MBCHECKALLS	_____	166
* 264	LEVELIF	_____	106
* 265	LEVELREP	_____	113
* 266	LEVELWHL	_____	111
* 267	LEVELLP	_____	114

Number	Original name	New name	Page
* 268	LEVELPROC	_____	120
269	HRGTON	_____	143
270	DIALCODE	_____	135
271	MENUCODE	_____	133
272	HRGTCOL	_____	131
273	HRTESTP	_____	145

Appendix B: Commands and functions listed alphabetically

The following table lists the commands and functions alphabetically. The command or function name is followed by its number. C and F indicate whether if it a command or function. The last number on the line is the page number where the command or function is located.

Name	Number	Type	Page
KEYREPEATON	(033)	(C)	37
KGETV	(039)	(C)	39
LDEL	(132)	(C)	77
LETTEROFF	(134)	(C)	47
LETTERON	(133)	(C)	47
LEVELIF	(264)	(F)	106
LEVELLP	(267)	(F)	114
LEVELPROC	(268)	(F)	120
LEVELREP	(265)	(F)	113
LEVELWHL	(266)	(F)	111
LIST	(004)	(C)	20
LOCKOFF	(136)	(C)	47
LOCKON	(135)	(C)	47
LOOP	(119)	(C)	113
LPEXITIF	(120)	(C)	113
MBALLOFF	(147)	(C)	164
MBAND	(185)	(C)	156
MBBLOCK	(141)	(C)	157
MBCHANGE	(184)	(C)	156

Name	Number	Type	Page
PMEM	(015)	(C)	26
PMERGE	(012)	(C)	25
POLD	(011)	(C)	26
POPIF	(208)	(C)	107
POPLP	(207)	(C)	114
POPPROC	(206)	(C)	120
POPREP	(204)	(C)	113
POPWHL	(205)	(C)	111
PRCOM	(172)	(C)	48
PRLIST	(170)	(C)	20
PRENUMBER	(010)	(C)	24
PROCEDURE	(126)	(C)	115
PROCEND	(127)	(C)	115
PRPRINT	(171)	(C)	48
REPEAT	(117)	(C)	111
RENCOM	(022)	(C)	15
RESET	(175)	(C)	22
RESTORE	(003)	(C)	103
RESUME	(029)	(C)	31

Appendix C: Error messages

This table contains the complete set of BeckerBASIC error messages. The error messages coincide with the ONERRORGO command (see Section 2.2).

ERROR NUMBER	ERROR TEXT
01	TOO MANY FILES
02	FILE OPEN
03	FILE NOT OPEN
04	FILE NOT FOUND
05	DEVICE NOT PRESENT
06	NOT INPUT FILE
07	NOT OUTPUT FILE
08	MISSING FILENAME
09	ILLEGAL DEVICE NUMBER
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT GOSUB
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEF'D STATEMENT
18	BAD SUBSCRIPT
19	REDIM'D ARRAY
20	DIVISION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG
24	FILE DATA
25	FORMULA TOO COMPLEX
26	CAN'T CONTINUE
27	UNDEF'D FUNCTION
28	VERIFY
29	LOAD
30	BREAK
31	REMARK
32	COMMAND TOO LONG

ERROR NUMBER	ERROR TEXT
33	COMMAND TOO SHORT
34	PROCEND WITHOUT PROCEDURE
35	UNDEFINED PROCEDURE
36	PROCEDURE-PARAMETER
37	CONSTRUCT NOT CLOSED
38	ENDDO WITHOUT WHILE
39	UNTIL WITHOUT REPEAT
40	LPEXITIF/ENDLOOP WITHOUT LOOP
41	LABEL
42	CASE/OTHER/ENDSEL WITHOUT SELECT
43	RETKEY WITHOUT ONKEYGO
44	RESUME WITHOUT ONERRORGO
45	ILLEGAL COMMAND

Appendix D: Memory map

The following table is an overview of the BeckerBASIC and GEOS memory layout. As you have already seen, memory is divided into sections.

This memory map will help you when you want to make your own changes to BeckerBASIC or GEOS. In conjunction with this, you should refer to the BeckerBASIC memory access commands (see Chapter 4). This allows access to ROM lying in RAM, where the gross majority of GEOS routines lie.

You can find a C64 operating system memory map, as well as the BASIC interpreter layout, in your C64 ***Programmer's Reference Guide*** and ***Anatomy of the C64*** from Abacus.

Memory range	Layout
2-68	This range is used by BeckerBASIC, BASIC 2.0 and GEOS. You'll find buffer memory for the other values starting around address 32576.
251-254	Miscellaneous memory.
1024-2047	Text screen memory.
2048-6799	BeckerBASIC program code. The Input-, Testing- and Run-Only-Systems reside here.
16800-32575	Approx. 15,800 bytes of BASIC memory.
(24576-32575)	Hi-res bitmap II. Dialogue boxes and drop-down menu routines need this second bitmap. Using these features reduces your available memory by around 8K.
32576-32767	Important `BeckerBASIC` routines and zero page buffer memory. Do not use this memory.

32768-40959	GEOS program and data memory. Hi-res color memory for bitmap I stays in 35840-36839.
40960-48959	Hi-res bitmap I. Hi-res graphics appear here for both GEOS and BeckerBASIC. When you avoid using the hi-res graphics in either the Testing-Systen or Run-Only-System, you have 8K more BASIC memory available. The Input-System uses this area for the BeckerBASIC command table (the Input-System cannot use hi-res graphics).
(40960-43959)	Command name table I (user-assigned names).
(43960-45800)	Command name table II (original names).
(47104-48103)	This area serves as buffer memory for the GEOS routines that normally lie in text screen color memory from 55296 to 56295 (for switched off hi-res graphics).
48960-65535	GEOS program and data memory.

Appendix E: BeckerBASIC in action

The following is a program designed as an example for demonstrating BeckerBASIC in action. It is on the BeckerBASIC disk under the name ADDRSAMPLE.

Let's assume that you want to save a list of names on your disk. For our sample program we'll need a drop-down menu.

Let's run the DDM.C.S program on the disk. First it will want to know where the table needed by the menu will be stored in memory. Enter `24200`. Next it will ask for how many item will appear on the menu bar. Enter `2`. Now, will it be a horizontal menu or vertical? Enter `0` for horizontal. Next enter `0` then `0` for the upper left corner of the menu. Enter `79` and `13` for the lower right corner.

When creating menus, it is best to enter height in multiples of 14, starting with 0. such as 0,13,27,41 and so on.

Now you need to enter the text to appear on the menu. Enter `FILE`. And enter `DATA`.

Next, we'll create the sub-menus. It now asks if a sub-menu should be created for FILE. Answer `Y`. Now enter `2` for the number of items. Enter `1` for vertical menu. Enter `0`, `13`, `59`, and `41` as the location coordinates. Now enter the text for items on the sub-menu. Type `CREATE/LOAD` and `QUIT`. To create the second sub-menu, enter the following text at the appropriate prompts: `Y` (for yes, you want a sub-menu for data), `2` (number of items), `1` (vertical), `20`, `13`, `64`, `41` (coordinates), `ENTER`, `READ` (names of the sub-menus).

Once you have created your menu, the program will wait for a keypress then display it. Click on a sub-menu item to continue. After you click, it will ask if the menu is acceptable. If you answer `Y`, it will ask for a name to save it under. If you answer `N`, you will be able to go through and reenter your data. Answer `Y` now to save the menu. Enter the name `ADDRMENU`.

You have now just created the table for your menu. To use it in our program, we will load it into memory with `DLOADM"ADDRMENU"`.

Now let's create a dialogue box. Load and run D.C.S off the BeckerBASIC disk. Now enter 828 for the address where the table will be loaded into memory. Enter N so you can create your own size of dialogue box. Enter the coordinates 60 and 50 for the upper-left corner. Now enter 260 and 150 for the lower-right corner. Next enter 0 for no shadow.

To add a button to the dialogue box, enter the number for the desired button. For our example we only want the CANCEL button so enter 2. Now the program wants to know how far over the button should be placed. Enter the number in bytes (divide actual pixels by 8, i.e., you want the button over 16 pixels so you would enter 2). Enter 2 now. Enter 70 for the number of pixels down from the top. You have just added a CANCEL button to the dialogue box.

To add text, enter 11. Do this now. Next enter the coordinates of were the text should be placed. Enter 16 and 14. Enter the text Please enter filename. at the prompt. Now add an input prompt so the use can enter text using the dialogue box. Type 13 for an input prompt. Enter the coordinates of the prompt as 16 and 35. Next enter the number of characters that will be allowed to be entered. Type 14.

Our dialogue box is now complete. You can enter 33 to see if the data for the buttons was entered correctly. Enter 0 and the dialogue box will be displayed. Before you display a dialogue box, make sure you added a button or you will not be able to return from the dialogue box. Click on the button to exit. If everything is okay, then enter Y to save the table. Enter ADDRDIAL as the name of the table.

With both the drop-down menu and the dialogue box tables saved, we are ready to enter our program. This program has a little of nearly everything. The following explains the program ADDRSAMPLE (on the BeckerBASIC disk) section by section. It will show some of BeckerBASIC's highlights in action.

The first section is labeled "SETUP". The first line sets up the error routine used for editing and debugging the program as it was written. When an error is encountered, it does the following: jumps to the line number after the ONERRORGO statement, places the error number in EN, places the error text in EN$, and places in EL the line number where the error occurred.

The next line sets the end of memory for your BeckerBASIC program. Since we want to protect our menu starting at location 24200, we set the end of BASIC to 24199. CLR resets variables and pointers.

GEOS is turned on next. The program sets up the arrays for the screen text and the address data. Then it loads the menu and dialogue tables into memory. Next it defines the strings that are to be placed on the screen. Finally it jumps to the routine that creates the screen.

The section labeled "MENU" first displays the drop-down menu. Then it defines where it should jump to when a menu item is selected. The labels that it jumps to correspond to the names of the items on the menu.

"DRAWSCREEN" first clears the screen. It then draws a box filled with the background pattern. Now it draws a smaller black box. Next it draws a black box for a shadow, then a white box overlapping the black box. A frame is then added.

The next three lines draws a wide, narrow box that will be used to display information.

The last group of lines prints the screen title in bold, italic and reversed type and the field descriptions in bold type.

"CREATE/LOAD" uses a dialogue box to enter the name of the data file. First it assigns N$ the default filename. Then N$ is stored into memory at 880. Next it's converted from ASCII to GEOS text. The dialogue box is called with the following line and the text inputted is to be placed at memory location 880. Since we placed the contents of N$ there, it will be displayed at the input prompt. The next line checks to see if the CANCEL button was selected. If it was the program flow returns to "MENU". The entered text is converted from GEOS back to ASCII. N$ is filled with the contents of 880 through 893. The REPEAT UNTIL loop looks for the end of the text in N$ so the extra characters can be stripped out. If the first character in N$ is CHR$(0) then there is no text and it returns back to "MENU". Next N$ is stripped of the extra characters. A sequential file is then created under the name N$. The program then checks to see if a file under that name exists. If it does exist than it is checked to see if data can be appended to it. If it cannot append data then an error occurs and is displayed in the info box we created on the screen. Before it jumps back to the "MENU" loop, a flag is set to indicate that a file exists to be used.

The "ENTER" routine permits data to be added to the sequential file. First it checks to see of a file has been cleared. If it was not cleared, it prints "NO FILE" in the info box. If it has, it continues and opens the file. The HRBOX command clears what data might be on the screen. The FOR-NEXT loop sets the entry variables to blanks and allows up to 15 characters to be entered. The next FOR-NEXT loop allows the user to enter the data and does the necessary conversions. Also it strips the entries of any extra text. Next it asks if the entry was okay. If it was not, then it allows you to make corrections on the text already entered. Once the text entered is satisfactory, the program then saves the data to disk. It then asks you if you want to enter more data.

The next section is "READ". This routine checks to see if the file has been okayed then opens the file. Next it INPUTs the data in the D$ array and displays each record until the EOF.

"INFOBOX" is used to display information on the screen. It then waits for a single keypress and stores what key was pressed in AN$. It then clears the box.

'ERRORS' uses the "INFOBOX" routine to display any errors that might occur. Once the program is thoroughly debugged, you may want to take this section out along with the ONERRORGO command.

"QUIT" turns GEOSOFF and ENDs the program.

If you want to RUN this program from the GEOS deskTop, first replace the END statement in the last line with DESKTOP. When you QUIT the program, it will return to the deskTop.

The next thing you will need to do is to run the CONVERTER program. This program converts your program so that it is accessible from the deskTop. You can add a creation date and design your own icon. More information on the CONVERTER program is in Section 1.1.4.

Appendix F: Distribution of the Run-Only System

Abacus grants to you a royalty-free right to copy and distribute the "Run-Only System" of BeckerBASIC provided that you:

(a) distribute the "Run-Only System" ONLY in conjunction with your own software program created using BeckerBASIC

(b) leave the Run-Only System unchanged and named "SYSTEM 3" upon the disk

Appendix G: Examples of DB and DF

Section 1.4 (page 19) described the DB and DF commands which allow the machine language programmer to add a command or function to BeckerBASIC. Here are two BeckerBASIC programs demonstrating each of these function from BASIC.

DB:

```
10 POKE 25500,169:POKE 25501,65:POKE 25502,32:POKE 25503,210
20 POKE 25504,255:POKE 25505,96
30 :'OTHER PROGRAM CODE AS NEEDED'
40 DB
```

The above sequence, which prints the A character on the screen when the DB in line 40 executes, is the equivalent of the macine language program:

```
LDA #$41
JSR $FFD2
RTS
```

DF:

```
10 POKE 25000,160:POKE 25001,1:POKE 25002,76:POKE 25003,162
20 POKE 25004,179
30 :'OTHER PROGRAM CODE AS NEEDED'
40 SCPRINT DF
```

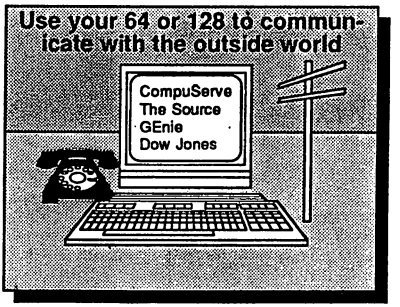
The above program, which returns a value of 1 when the SCPRINT DF executes in line 40, is the equivalent of the machine language program:

```
LDY #$01
JMP $B3A2
```

Selected Abacus Products for Commodore computers

SpeedTerm

Terminal Software for both the C-128 and C-64



As a group, Commodore owners are one of the largest users of online communication services, such as CompuServ, The Source, Delphi and GEnie. **SpeedTerm** was designed to handle the communication needs of this rapidly growing base of Commodore owners who access these services. Both programs are packaged together, so it's easy for you to order and stock **SpeedTerm.**

SpeedTerm sets a high standard in economical telecomputing software—this package offers more power per dollar than any other terminal program for the '64 and '128. **SpeedTerm** is a completely command-driven program that is easy to learn and use, yet provides great power and flexibility.

Even though **SpeedTerm** is simple in design, it packs numerous features that aren't found in others terminal packages. For instance, it supports both Xmodem and Punter file transfer protocols so that large files can be uploaded and downloaded without error. In addition to these popular file transfer protocols, **SpeedTerm** includes partial DEC VT52 terminal emulation. In addition to the standard options found in other terminal programs, manages a large 45K capture buffer and permits user defined function keys. **SpeedTerm** understands more than 30 powerful commands.

SpeedTerm is compatible with most of the inexpensive modems for the C-64 and C-128, and if properly interfaced, will function with all Hayes® compatible RS-232 modems. **SpeedTerm's** versatile capture buffer which can be used to both send and receive ASCII text files, or to record an online session.

The complete **SpeedTerm** package includes a 70 page manual with easy to understand tutorial.

SpeedTerm Features:
- Xmodem and Punter protocols for error-free filetransfer
- Supports partial VT52 terminal emulation
- Manages large capture buffer for recording long sessions (C-128 version has a 45K buffer, C-64 version has 24K)
- Use buffer to copy sequential files from disk to disk, and split files too large to fit into wordprocessors.
- Execute disk commands, e.g. scratching/renaming files
- Lists sequential files on the screen or printer.
- Displays directory listings
- Send commands to the disk and read the error channel
- Has powerful command mode with over 30 commands
- Complete access to the DOS
- Permits flexible user-defined function keys
- Works with most popular modems
- Works with either 40 or 80 column monitors
- Includes 70-page manual with easy to uinderstand tutorial

Modems:
- Commodore 1600, 1650, 1660
- Hayes and Hayes-compatibles

Hardware requirements:

SpeedTerm-64
- Commodore 64
- 1541/MSD or 1571 disk drive
- 40-column monitor

SpeedTerm-128
- Commodore 128
- 1541/MSD or 1571 disk drive
- 40- or 80-column monitor

Suggested retail price:
Program disk contains
both '64 and '128 versions **$39.95**

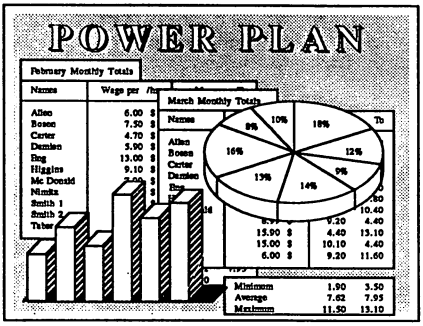
Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

PowerPlan-64

Spreadsheet and Graphics package for the C-64

"PowerPlan is one of the best programs ever written for the Commodore 64, giving Lotus 1-2-3® a run for the money... It was a pleasure to work with this amazing product. I strongly recommend PowerPlan to anyone interseted in using spreadsheet programs for business."

—Al Willen
Commodore Magazine



Ever since VisiCalc and Lotus 1-2-3 stormed the personal computer market, the computer has become an important financial planning tool. By using an electronic ledger, you can perform hundreds of calculations and "what-if" analyses quickly and easily, and reduce reams of data into meaningful information.

PowerPlan-64 offers the '64 user a software tool that combines spreadsheet operations with a powerful built-in graphics program to display data in graphs as well as numbers. **PowerPlan-64** can handle up to 255 rows by 63 columns—a total of 16,065 individually protected cells. This outstanding package includes all major math functions, manually controllable calculation mode, and built-in disk commands. The integrated graphics program, PowerGraph, has eight different windows —you can select bar charts, curve graphs, point charts, pie charts, or min-max charts in two or three directions.

PowerPlan-64's menus make it easy to use for the first-time spreadsheet user. All of **PowerPlan-64**'s selections are clearly displayed on the screen for the user to choose from. In addition, online HELP screens are available at the touch of a key.

PowerPlan-64's complete 200-page handbook has a plain-speaking tutorial that gently introduces the user to spreadsheets.

Hardware requirements:

Commodore 64
1541 disk drive (or MSD disk drive)

Printer optional

Self-running demo available

PowerPlan-64 Features:

- Menus make PowerPlan-64 easy to learn
- Large capacity spreadsheet serves all the user's analysis needs
- Convenient built-in notepad documents user's important memos
- Flexible online calculator gives access to quick computations
- Powerful options such as cut, copy and paste operations speeds the user's work
- Integrated graphics summarize hundreds of data items
- Draws pie, bar, 3D bar, line and area charts automatically (8 chart types)
- Multiple windows emphasize the analyses

Printers:
PowerPlan-64 works with the following printers:

- Commodore 1525, 1526
- Epson MX, FX, Homewriter 10 and compatibles (Star Gemini SG-10, 10x, 10C, 15x, Panasonic KXP 1080)
- MPS 801, 802

Suggested retail price:
C-64 version **$39.95**

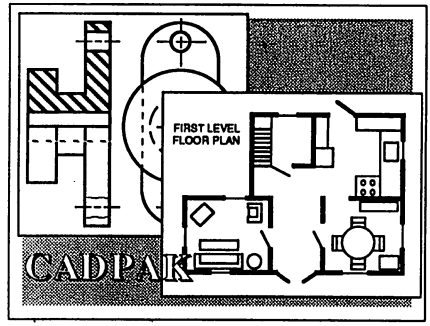
Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Selected Abacus Products for Commodore computers

Cadpak

Enhanced and 1351 Mouse Versions!

Computer-Aided Design package for the C-64 or C-128



Cadpak is a superb tool for computer aided design and drawing for the Commodore 64 and 128—it's been our bestselling software package for the last year and a half. It offers Commodore users a simple, versatile solution for producing high quality computer-aided designs and drawings without requiring any programming knowledge. **Cadpak** was designed to be simple enough for the novice, yet incorporate the design functions and printout capabilities of a truly professional CAD system. Its simplicity, accuracy and speed make **Cadpak** a standout.

Cadpak can be used with either the keyboard, an optional lightpen or optional mouse to draw directly on the screen and create and edit pictures, drawings, layouts and renderings. The feature that sets **Cadpak** apart from its competition is its exclusive **dimensioning** feature, which allows exact scaled output of designs to most popular dot-matrix printers (listed below). Choose from the menu options and draw on the screen at an exact location with **Cadpak's** exclusive *AccuPoint* cursor positioning.

Cadpak's menu options make it easy to use for beginners. **Cadpak** also boasts many sophisticated features for the advanced user. Using the two graphics screens, you can draw lines, boxes, circles, ellipses; fill with solid colors or patterns; draw freehand; copy sections of the screen. The user can zoom in to do detailed design on a small section of the screen. **Cadpak's** improved object editor lets the user define and save furniture, electronic circuitry, machinery, etc. as intricate as the screen resolution permits. Perfect for all design needs on the Commodore C-64, 64C and C-128.

Cadpak-64 has two screens with 320 x 200 resolution. **Cadpak-128** has a first screen resolution of 640 x 360 and the second screen resolution of 320 x 200.

Hardware requirements: (Lightpen and mouse optional)
Cadpak 64:
Commodore 64
1541 disk drive (or MSD disk drive).

Cadpak 128:
Commodore 128
1571/1541 disk drive (or MSD disk drive)

1351 Mouse version now available!

Exclusive **Dimensioning** feature assures exact scaled output of designs

Cadpak Features:

- Precision scaled output to most dot-matrix printers —objects retain exact proportions when printed
- Two screens for flexible copy operations
- Drawing using either the keyboard, high-quality lightpen (optional), or new MOUSE version (optional, available April 87)
- Pre-defined or user-defined fill patterns
- SAVE/RECALL designs to and from diskette
- Library contains pre-defined objects, symbols, fonts—create and add custom symbols/fonts
- Mirror, rotate and zoom functions
- Advanced labeling/ measuring features
- Sophisticated graphic functions for lines, boxes, ellipses, arcs, etc.
- Custom-created text fonts or graphic symbols

Printers:

- Commodore 1525, 1526
- Comrex CR-220
- Epson MX, FX, Homewriter 10 and compatibles (Star Gemini SG-10, 10x, 10C, 15x, Panasonic KXP 1080)
- MPS 801, 802, 803, 1000
- Okidata Microline
- Okimate-10 b/w and color.
- Prowriter 8510A, 8510SC color
- Seikoska 1000
- Siemens PT88/89

Suggested retail price:

C-64 version	$39.95
C-128 version (Mouse version July 87)	$59.95

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

COBOL

for the C-64 or the C-128

COBOL is the most widely used commercial programming language in use today. The **COBOL-64** and **COBOL-128** packages let users learn the COBOL language using their Commodore 64 or Commodore 128 home computer. The COBOL language uses English-like sentences. This makes it an easy to learn language. And since **COBOL-64** and **COBOL-128** are designed with easy of use in mind, it's perfect for the beginner. Since the COBOL language is common to many different computers, every aspect of COBOL learned on the '64 and '128 is valid for larger system versions.

Our COBOL software includes a syntax checking editor, a compiler, an interpreter and symbolic debugging aids. So you'll be able to write and test your COBOL programs very quickly.

COBOL-128 is more than a conversion of our popular **COBOL-64**. It takes advantage of the new '128 features. **COBOL-128** works with either a 40- or 80-column monitor. In addition, because of the increase memory of the '128, **COBOL-128** runs much faster than the C-64 version.

COBOL-64 and COBOL-128 Features:

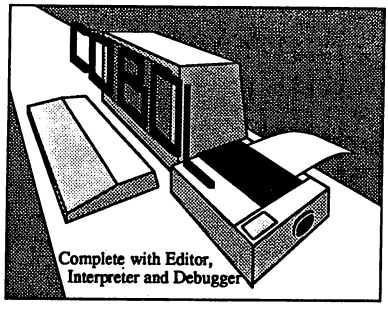
- Includes integrated editor for creating COBOL source
- Fast compiler/interpreter to transform source into excutable program
- Features symbolic debugging tools: breakpoint, trace, single step.
- Supports subset of ANSI COBOL '74
- Includes a crunch function to reduce the memory size of your programs
- Includes sample programs demonstrating file handling
- Complete 150-page manual

Hardware requirements:

COBOL-64:
Commodore 64 with 1541 or 1571 disk drive

COBOL-128:
Commodore 128 with 1541 or 1571 disk drive
(supports 40- or 80-column monitor)

Works with most popular dot-matrix printers (optional).




```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  BUILD-DATA1.
000300 AUTHOR. VISIONARY-SOFTWARE.
000400 ENVIRONMENT DIVISION.
000500 CONFIGURATION SECTION.
000600 SOURCE-COMPUTER.  C64.
000700 OBJECT-COMPUTER.  C64.
000800 INPUT-OUTPUT SECTION.
000900 FILE-CONTROL.
001000     SELECT DATA1 ASSIGN TO DISK-1541 DRIVE-8
001100     FILE STATUS IS FILE-ST.
001200 DATA DIVISION.
001300 FILE SECTION.
001400 FD  DATA1
001500     LABEL RECORDS ARE OMITTED
001600     VALUE OF FILE-ID IS "00:DATA1".,
001700 01  DATA-RECORD.
001800     02  NAME-FIELD    PIC X(20).
001900     02  ADDR-FIELD    PIC X(20).
002000 01  DATA-RECORD-2.
002100     02  NAME-FIELD-EXIT PIC X(4).
002200     02 FILLER PIC X(36).
002300 WORKING-STORAGE SECTION.
002400 77  WRITE-FLAG PIC X VALUE "N".
002500 77  RVS-ON VALUE CHR 18 PIC X.
002600 77  RETURN-CODE VALUE CHR 13 PIC X.
002700 77  CLEAR-HOME VALUE CHR 147 PIC X.
002800 77  FILE-ST PIC XX.
002900 PROCEDURE DIVISION.
003000 START-UP.
003100     DISPLAY CLEAR-HOME
003200     OPEN OUTPUT DATA1
003300     IF FILE-ST IS NOT EQUAL TO
003400        "00" DISPLAY "OPEN ERROR"
003500           STOP RUN.
003600     PERFORM GET-DATA-LOOP THRU LOOP-EXIT.
003700 END-IT.
003800     CLOSE DATA1
003900     IF FILE-ST NOT EQUAL TO "00"
004000        DISPLAY "CLOSE ERROR".
004100     STOP RUN.
```

Suggested retail price:

C-64 version	**$39.95**
C-128 version	**$39.95**

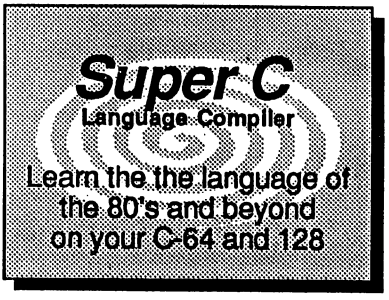
Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Super C

C language development package for the C-64 or C-128

"The Super C Compiler provides an ideal introduction to a very functional version of the C language...it is the best starter C package (for the Commodore 64) *and the price is right"*

—Walt Lounsberry
Commodore Microcomputers



The C language is one of the most popular in use today—it's an excellent development tool, produces fast 6510 machine language code and is very easy to transport from one computer to another. To maintain C's portability, our **Super C** development packages support the Kernighan & Ritchie C standard (except for bit-fields), making them very complete.

Super C's powerful full-screen editor lets the user create source files up to 41K in length (larger on C-128). **Super C's** editor includes Search and Replace functions and features horizontal and vertical scrolling on a 40-column monitor. The C-128 version supports 40- or 80-column monitors.

The fast compiler (maximum of 53K object code) creates files which the linker turns into a ready-to-run machine language program. **Super C's** linker combines up to seven separately compiled modules into one executable program.

The I/O library includes many of the standard functions, including `printf` and `fprintf`, with libraries for math functions and graphics. The runtime library may be called from machine language or included as a BASIC lookalike program.

```
 3 char buffer[41];
 4
 5 main()
 6 {
 7    putc(CLR,STDIO);
 8    BASICset(charrami);
 9    while()
10    {
11        do{
12            gets(buffer,40,STDIO);
13            putc(CR,STDIO);
14
15     }while(strcmp(buffer,"read\n");
16
17        puts("\nnames:",STDIO);
18        gets(buffer,40,STDIO);
19        putc(CR,STDIO);
20        readset(buffer,charrami);
```

Super C Features:

- Built-in editor with search, replace, block commands, and much more
- Supports strings and arrays
- Handles object code up to 53K
- Supports recursive programming techniques
- Includes very complete math functions and library
- Includes standard I/O and fast graphics libraries
- C-128 version features high-speed RAM disk support and 40/80 column
- Complete with 275-page manual

Hardware requirements:
Super C 64:
Commodore 64 with 1541 or 1571 disk drive

Super C 128:
Commodore 128 with 1541 or 1571 disk drive
(supports 40- or 80-column monitor)

Printer optional.

Suggested retail price:	
C-64 version	$59.95
C-128 version	$59.95

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Super Pascal

Pascal language development package for the C-64 or C-128

Super Pascal is a complete program development system for the Commodore 64 or Commodore 128. **Super Pascal** is so capable that hundreds of schools are using it to teach Pascal programming to their students. But Pascal is more than just a learning language. **Super Pascal** features language extensions for serious system level programming.

Super Pascal implements the full Jensen & Wirth compiler plus extensions for graphics. The package consists of an easy-to-use, very complete source file editor; an online assembler for optionally coding in machine language; and a super-fast compiler to turn the source file into executable code and a high-speed DOS for speeding up disk access to the 1541/1571.

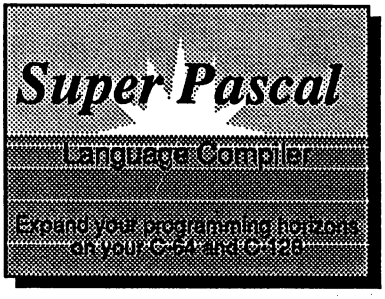
Other **Super Pascal** package features include a high-precision 11-digit arithmetic; a very fast compiler; overlays; automatic loading of editor and source program; exact error messages and localization during compilation; complete statistics reporting; free runtime package, and much more.

Super Pascal 128 contains all the features found in our popular C-64 version while taking advantage of the C-128's 40/80 column modes; it's high-resolution graphics package runs in 80 columns and makes some truly remarkable artwork possible.

Another "extra" of **Super Pascal 128** is its RAM disk, which allows for *ultra-fast* loading/compiling, and supports 1571 Burst mode.

Super Pascal Features:

- Full implementation of Jensen & Wirth Pascal
- High speed DOS is three times faster than 1541 DOS
- Includes many language extensions for systems programming
- Integrated assembler for machine code requirements
- Built-in editor with renumber, auto, find, etc.
- Includes fast graphics libraries
- Works with one or two disk drives
- Large 48K workspace
- C-128 version supports 80-column hi-res graphics and supports RAM disk
- Complete with 200-page manual




```
      GET_NUM(FROM);
      IF NOT EOLN THEN GET_SECND('-') ELSE
       TIL:=FROM
     END
    END;

PROCEDURE GET_TITLE(FOR_GET:BOOLEAN);
 BEGIN
  TEST_SYNTAX
   IF INPUT^='*' THEN
    BEGIN
     IF NO_DEF THEN STOP (TITLE_ND);
     IF FOR_GET THEN TEST_FOR_SAVE
    END
   ELSE
    BEGIN
     IF NOT (INPUT^ IN LETTER) THEN
     STOP (ILL TITLE);
    READ (TITLE);
    IF FOR_GET THEN TEST_FOR_SAVE;
    NOT_DEF:=FALSE;COMMON^:=TITLE
   END
 END;
```

Hardware requirements:
Super Pascal-64:
Commodore 64 with 1541 or 1571 disk drive

Super Pascal-128:
Commodore 128 with 1541 or 1571 disk drive
(supports 40- or 80-column monitor)

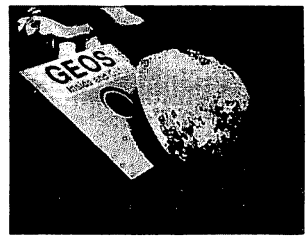
Printer optional

Suggested retail price:

C-64 version	$59.95
C-128 version	$59.95

Abacus Inc.
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

GEOS INFO



Another Abacus Best Seller!

GEOS Inside and Out

If you use GEOS then our new book, ***GEOS Inside and Out***, has the info you need.

A detailed introduction is laid out for the novice–beginning with how to load the GEOS operating system...how to create a backup...how to alter the preference manager...how to format disks...learn geoWrite and geoPaint in detail...use geoPaint for designing floor plans or drawing electronic diagrams. Easy-to-understand examples, diagrams and glossary are included to enlighten the beginner.

The advanced user will find more detailed information on GEOS's internals and useful tricks and tips. Add a constant display clock–includes assembly and BASIC listing...complete listing of our **FileMaster** utility (converts your programs to GEOS format with an icon editor) with a line by line explanation...create a single-step simulator for observing memory and the various system registers...learn about windows and how to use them to your advantage...understand GEOS file structure.

If you're just getting started with GEOS or getting to the point of wanting to add your own applications, then ***GEOS Inside and Out*** will help you on your way. $19.95

Coming Soon!

GEOS Tricks & Tips

Continuing the tradition established by our famous C-64 reference library, ***GEOS Tricks & Tips*** is a collection of helpful techniques for anyone who uses GEOS with their Commodore. It's easy to understand without talking down to the reader, and detailed in the applications of the routines. Includes a font editor to create up to 64 point text and a machine language monitor. A perfect companion volume to ***GEOS Inside and Out***. Available Second Quarter. $19.95

GEOS, geoWrite, geoPaint are tradenames of Berkeley Softworks.

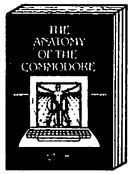
To receive your copy of ***GEOS Inside and Out*** and/or ***GEOS Tricks & Tips***, call now for the name of the dealer or bookstore near you. Or order directly using your Visa, MC or Amex card. Add $4.00 per order for shipping and handling. Foreign orders add $10.00 per book. Call or write today for your free catalog. Dealer inquires welcome—2000 nationwide.

Order both today!

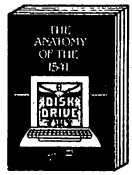
You Can Count On **Abacus**

5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Just a few of our books...



Anatomy of the C-64 Insider's guide to 64 internals. Graphics, sound, I/O, kernal, memory maps, and much more. Complete commented ROM listings. 300pp $19.95

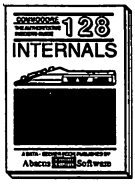


Anatomy of the 1541 Drive Best handbook on this drive, explains all. Filled with many examples programs, utilities. Fully commented 1541 ROM listings. 500pp $19.95

Tricks & Tips for the C-64 Collection of easy-to-use techniques: advanced graphics, improved data input, CP/M, enhanced BASIC, data handling and more. 276pp $16.95

Peeks & Pokes for the 64 Includes in-depth explanations of PEEK, POKE, USR, and other BASIC commands. Learn the "inside" tricks about your 64. 200pp $14.95

Graphics Book for the C-64 Best reference, covers basic and advanced graphics. Sprites, Hires, Multicolor, 3D-graphics, IRQ, CAD, projections, curves. 350pp $19.95



C-128 INTERNALS Important C-128 information. Covers graphic chips, MMU, I/O, 80 column graphics and fully commented ROM listings, more. 500pp $19.95

1571 INTERNALS Essential reference. Internal drive functions. Explains various disk and file formats. Fully-commented ROM listings. 450pp $19.95

C-128 TRICKS & TIPS Fascinating and practical info on the C-128. 80-col hires graphics, bank switching. 300 pages of useful information for everyone. $19.95

C-128 PEEKS & POKES Dozens of programming quick-hitters, techniques on the operating system, stacks, zero page, pointers, and BASIC. 240pp $16.95

C-128 BASIC 7.0 Internals Get all the inside info on BASIC 7.0. This exhaustive handbook is complete with fully commented BASIC 7.0 ROM listings. $24.95

Call now for the name of your nearest dealer. Or order direct with your credit card by calling 616/241-5510. Add $4.00 per order for S&H. Foreign add $12.00 per book.

Other books and software also available. Call or write for your free catalog. Dealers inquires welcome—over 2000 nationwide.

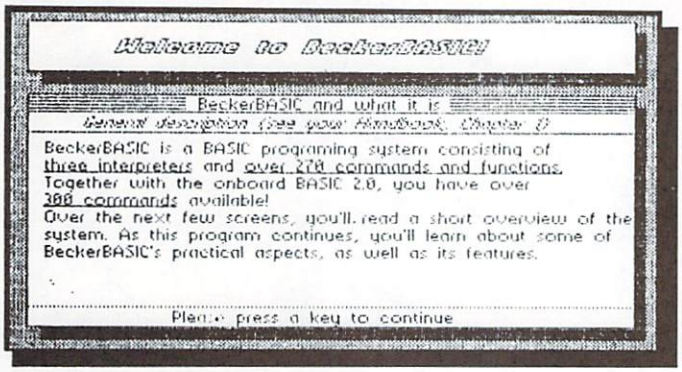
Commodore 64 and Commodore 128 are trademarks of Commodore Ltd.

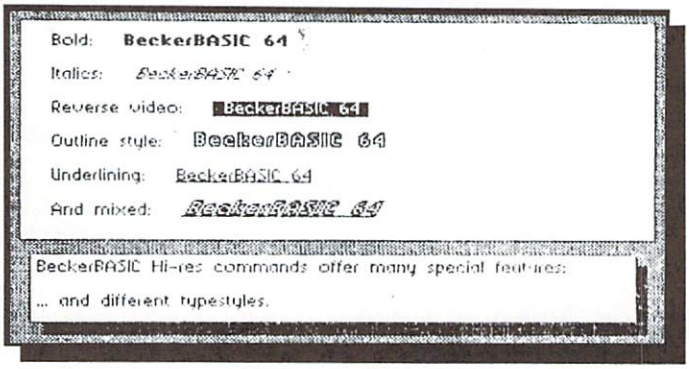
Abacus

Abacus Software
5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

BeckerBASIC for GEOS

Now you can write BASIC applications to work with GEOS





Introducing **BeckerBASIC**. If you already know BASIC, you can now write your own GEOS applications in BASIC, easily.

BeckerBASIC gives you the power of over 270 new commands and functions.

Over 20 commands to make your programming easier. For example, TRACE, RENUMBER, DUMP, DIR, etc.

Packed with over 50 commands for easy disk access. Load and save blocks of memory or selected lines of your program. You can even PEEK and POKE into your disk drive's memory.

10 commands can be used for easier cursor control. Turn the cursor on and off. Set how quickly it flashes. Position it at any location on the screen.

20 commands are available for all your hires programming needs. Creat boxes, plot points, and draw lines.

18 additional commands are dedicated to creating sound. Set ring modulation, change the filter, alter the waveform and set the envelope.

Over 35 commands let you create and animate sprites with ease. Load and save sprites directly. Alter their size, change their positions and check for collisions. Use the sprite editor to create sprites and icons.

Use the *Pulldown Menu Construction Set* and *Dialog Box Construction Set* to aid in the creation of you own applications

Royalty-free distribution of your **BeckerBASIC** applications.

Now anyone can create applications in BASIC to run with GEOS. Only $49.95

For credit card orders call 1-800-451-4319
Michigan residents call 1-616-698-0330

For technical support call 1-616-698-0330

Call today or mail the coupon for your free catalog covering our complete line of software and books for the Commodore 64 and 128. Or ask for the location of the dealer nearest you. You can order direct by phone using your VISA, American Express or MasterCard or detatch and mail your completed coupon. Dealer inquiries welcome—over 2400 nationwide.

Abacus

5370 52nd Street SE
Grand rapids, MI 49508
Telex 709-101 • FAX 616/698-0325

GEOS is a trademark of Berkeley Softworks.
Commodore is a trademark of Commodore Electronics Ltd.

If your Commodore dealer doesn't carry Abacus products, then have him order them for you. Or you can order direct using the following order blank or by calling—**1-800-451-4319**

Qty.	Product	Price	Total
____	**BeckerBASIC** for the Commodore 64	$49.95	____
	In USA add $4.00 for S & H per order. Foreign add $12.00 per item.		____
	Michigan residents include 4% sales tax		____
	Total amount enclosed (US funds)		____

Payment: ( ) MasterCard ( ) VISA ( ) American Express
( ) Money Order ( ) Check

Card No._____ Exp._____
Name_____
Address_____
City_____ State_____ Zip_____
Phone No._____

B·e·c·k·e·r B·A·S·I·C

CONTEST

$25,000 in prizes for the best GEOS applications using BeckerBASIC

PRIZE LIST

1st Prize	$1000 CASH (1 winner)
2nd Prize	Choice of Abacus books and software (2 awards) $500 value
3rd Prize	Choice of Abacus books and software (2 awards) $400 value
4th Prize	Choice of Abacus books and software (2 awards) $300 value
5th Prize	Our complete C-64 Library set-$227 value (100 awards)

To enter:

Return this entry form and your 51/4" diskette.the BeckerBASIC **Entries must be received by midnight, August 31,1988**, to be eligible. To win, you must comply with the competition rules. (Over)

Mail entry and this form to:
Abacus BeckerBASIC contest
5370 52nd Street
Grand Rapids, MI 49508

Abacus

CONTEST RULES

- Write your entries using BeckerBASIC to run under GEOS.Entries must be submitted on a diskette.
- You can submit multiple entries provided that all entries fit on a single diskette.
- Entries must be accompanied by the official entry form you'll find inside the BeckerBASIC package. Xerox or reproductions of the entry form are not acceptable.
- Your entry is received by Abacus no later than August 31, 1988.
- We'll announce the winning entries by October 31, 1988.
- Entry forms must be completed in full to be valid. No responsibility is assumed for late, lost or misdirected mail.
- This competition is open to registered owners of the BeckerBASIC software program. All prizes will be awarded. Prizes are non-transferable and not redeemable for cash. No substitution of prizes are permitted. Prizes to consist of (1) first prize of: $1000 CASH, (2) 2nd prize of: $500 value Abacus books and software, (2) 3rd prize of: $400 value Abacus books and software, (2) 4th prize of: $300 value Abacus books and software, (100) 5th prize of: Our complete C-64 Library Set-$227 value.
- Winners will be notified by mail, and must claim their prize within 30 days or an alternate winner will be selected. Prizes won by a minor will be awarded to the winner's parent or legal guardian. For a list of the winners, send a stamped, self-addressed envelope to Abacus Software.
- All federal, state, provincial, and local taxes will be the responsibility of the prize winner. Winners may be required to execute an affidavit of eligibility and release.
- The competition is open to all registered owners of BeckerBASIC software program from within the U.S. and Canada, except employees and their families of Abacus Software Inc. or their affiliates, subsidiaries, or agents. Void where prohibited by law.
- Selection of winners will be conducted by Abacus whose decision will be final. No correspondence will be entered into, and all entries become the property of Abacus Software.
- Entrants grant Abacus Software, without limitation the right to use their names, likeness, and competition entry for any advertising and/promotion purpose.or marketing

---

ENTRY FORM

Registration#____ Program name:_____

Name_____

Address _____

City _____ State_____ Zip _____

Abacus 

Register this software and be eligible to win additional software free in our monthly drawing.

Return this card to register your purchase and to receive free technical support for this product. You may also order a *non-copy protected* backup of this program.

Monthly drawing winner will be notified by mail.
Good Luck!

FOLD HERE→ AND TAPE

REGISTRATION CARD

705
Product ID

Registration# 173714 Program name:
Name
Address
City State Zip

Purchase Information:
Dealer
Address
City State Zip

Return this registration card to obtain a *non-copy protected* backup of the above program for a handling charge of $10.00. A check, money order, or credit card number must accompany this request. Purchase orders are not acceptable.

Non-copy protected backup?

☐ No, do not send a *non-copy protected* backup, but register my purchase.

☐ Yes, send a *non-copy protected* backup. $10.00 payment is enclosed.

Credit card#
Expiration Date___/___/___

Abacus
5370 52nd Street SE
Grand Rapids, MI 49508

Abacus

5370 52nd Street SE • Grand Rapids, MI 49508

ISBN 1-55755-033-6