THE TOOL

FOR THE COMMODORE

64

MICRO APPLICATION

THE TOOL, for COMMODORE's 64 personal computer is a powerful complement to your system, to facilitate and improve greatly the performance of your programs.

It consists of a collection of new BASIC-type instructions, that you will quickly learn, to enhance the management of the screen, to draw lines and columns, to realize efficient data acquizition routines, to play with the colors and the screen, to draw high-resolution graphics and have characters in the same time and to save, store and load screen-pages (graphic and text) on the disk.

THE TOOL also includes very useful programmer's aid instructions to design and debug all programs and has many others features including a joystick reserved function, and instructions to work with string of characters.

The DOS support instructions have been included to ease your work with the disk drive.

This document has been designed to be a teaching aid and a reference manual. A careful reading will give you all the elements necessary for an efficient and rational utilization of THE TOOL.

A look at the examples and a reading with your 64 running should give you all the capabilities of the system very rapidly.

OVERVIEW

THE TOOL is stored in a cartridge.

Just plug it in the memory expansion port located at the back of your 64 and power on the system. THE TOOL will be automatically loaded in memory, leaving you 30k of memory to work on. You have now the full power of THE TOOL at your disposal.

Note :

The physical configuration of the cartridge should avoid any confusion but you can check on your 64 's user manual the pinouts for input/output devices.

OVERVIEW

The screen generator integrated into THE TOOL was designed to ease data acquisition on the screen. The acquisition is more reliable than normal BASIC routines and the execution, fundamental in screen management, much quicker, since it is written in machine language.

THE TOOL controls the following functions :

- display commands to draw lines, columns, to scroll on the screen, etc...,

- acquisition commands to define acquisition zones and associated controls, to allocate zone contents to BASIC variables,complete printusing, etc...,

- screen-page management commands includes save and load of screen-pages from the disks.

In the following pages, the names we shall use for variables are arbitrary; they were chosen for easier comprehension. Any BASIC variables would suit, aside from the reserved variables for THE TOOL : "zo" and "ok".

The screen is defined as a drawing board with the origin in the upper left-hand corner. The origin coordinates are 1,1.
Any point on the screen is defined by :

- l = line number
- c = column number

OVERVIEW (continued)

Generally speaking :

- l : line (or row) number (1 <= l <= 25)
- c : column number (1 <= c <= 40)
- ln : length (or number of columns) - horizontal line
- lg : height (or number of lines) - vertical line.

For a better understanding of the chapter, experiment with these commands on the screen of your system, in direct or programmed mode.

A SCREEN PAGE is everything on a screen at a given time. It is possible to create complete pages with a frame or a board, save them on disk, load them directly on the screen, fill them with data, and a lot more.

GRAPHIC MODE / TEXT MODE

We will see that it is possible to mix characters and graphics and save those graphics on disks. The graphic instructions will be reviewed in the following chapter. But, for now, all the following instructions won't be used with the graphics.

We will have two working modes : a TEXT mode and a GRAPHIC mode. Chapter 3 deals with the instructions in Text Mode and Chapter 4 deals with the instructions in Graphic mode. The default mode is the Text Mode, so we will give more details in the next chapter.

tline

Function : tline draws on the screen a horizontal line defined by its origin and length.

Syntax : tline ln, l, c

with ln : length of the line,
l : line # of the starting point,
c : column # of the starting point.

Utilization : The tline command is used to draw frames on the screen or to underline data and titles.

Example : tline 22,1,1
tline 22,12,1

These two instructions draw two parallel lines in the upper part of the screen.

tcol

Function : tcol draws on the screen a vertical line defined by its origin and length.

Syntax : tcol ln, l, c
with ln : height of the vertical line
l : line # of the starting point
c : column # of the starting point.

Utilization : The tcol command is used to draw frames (with tline)

Example : tcol 12,1,1
tcol 12,1,22

These two instructions draw two parallel columns in the upper part of the screen.

Notes:

```
10 tline 22,1,1 : tline 22,12,1
20 tcol  12,1,1 : tcol  12,1,22
```

These two BASIC lines draw a frame. Note that the intersections of the frame are correctly drawn.

clear

Function : clear clears the screen from an origin point.

Syntax : clear l, c, lg, ln
with l : line # of the starting point,
c : column # of the starting point.
lg : height (number of lines)
ln : lenght (number of columns)

Utilization : clear is used to clear parts of the screen. The command clears only the parts of the display required other parts.

Examples : To clear the first line of the screen

```
clear 1,1,1,40
```

To clear the frame already drawn and also all that is inside the frame

```
30 clear 1,1,12,22
```

(clears a "window" of 12 lines and 22 columns).

out

Function : out displays on the screen a string a$, at a given location.

Syntax : out a$,l,c
with a$: any string
l : line # of the starting point
c : column # of the starting point.

Utilization : out displays a string on any part of the screen without having to use the cursor keys in string form. Out also facilitates frame filling.

Example
```
40 a$ = "JOHN SMITH"
50 out a$,5,10
```

These two lines will display JOHN SMITH on the fifth line starting at the tenth column.

rev

Function : rev converts to reverse mode a window defined by its starting point, length, height and color.

Syntax : rev l,c,lg,ln [,co]

With : l : line # of the origin point
c : column # of the origin point
ln: length (or number of columns)
lg: height (or number of lines)
co: new color : optional.
(0 to 15 - see page 3-21)

Utilization : rev draws the user's attention to a section of the screen. By using the instruction more than once with the same coordinates, it is possible to make the window blink. The speed of the blinking can be regulated by using timing loops (for : next).

Example : To use the reverse mode for the names of the previous example.

```
140 rev 5,1,1,10,3 : rev 6,10,1,11,4
```

scroll

Function : scroll shifts, up or down, left or right, part of the screen or window, defined by its starting point, its length and height.

Syntax :

scroll l,c,lg,ln,ty
with

l : line # of the starting point
c : column # of the starting point
ln: length of the window (or number of columns)
lg: height of the window (or number of line)
ty: type of scroll
- u for up
- d for down
- l for left
- r for right

Utilization : Scroll displays data files too large to be shown on one line. With simple coding, it is possible to scroll data in a frame and, as soon as the value is found, to save it and to carry on with the program.

Example :

```
10 sclear
30 a$="JOHN":b$="PATTY":c$="JULIE"
40 out a$,11,4 :out b$,11,10 :out c$,11,17
50 for i=0 to 9
60 scroll 10-i, 4,2,4,u
70 clear  11-i, 4,1,4
80 scroll 11+i,17,2,5,d
90 clear  11+i,17,1,5
100 for j=1 to 200:next j
110 next i
120 end
```

Just type the program and see what happens. You can also run the program without the lines 70 and 90 to see the difference. You can do similar samples using the left and right scrolling.

Notes: - A scroll does not erase the previous output.

Overview :

THE TOOL acquisition instructions have been designed to be used instead of INPUT and GET to allow homogeneous, controlled, efficient and reliable data acquisition.

This section is dedicated to the acquisition commands. A zone is defined by certain parameters (starting point, size, allowed type of characters, possibilities of exit (other than return) and an identification number). (decz)

Once the zone is defined, the regz command instructs THE TOOL to bring data in it, performing the controls assigned and reproducing the screen control functions, such as : moving the cursor, deleting, inserting. The inz command performs information transfer from a zone to a string (outz performing the opposite operation). 128 different zones can be defined on one page or screen.

```
input a$        decz 1  : zone definition
                reqz 1  : controlled acquisition
                inz 1,a$: transfer into a$
```

We will give more details in the following pages.

decz

Function : decz is used to define all parameters of a screen zone for controlled aquisition.

Syntax : decz n,l,c,ln [,ty] [,f$]
with n : identification number (0<=n <=127)
l : line # of the starting point (1<= l <=23)
c : column # of the starting point (1<= c <=22)
ln : total length of the zone (1<= ln <=255)
ty : type of control zone
value of ty :
. ty = n numeric data only,
. ty = m data in capital letters,
. ty = r other possibilities of exit than with RETURN.
* for a numeric zone : almost all the keys, numeric ones excepted,
* for an alphanumeric zone : all non alphanumeric keys

. The ASCII code of the key you press for exit is put in the variable zo, so that you can define and use control keys, like

F1 to F8	: zo = 133 to 140
CRSR / DOWN	: zo = 17
CRSR / UP	: zo = 145
SHIFT/RETURN	: zo = 141
CTRL/ A to Z	: zo = 1 to 26

. ty = p zone defined in printusing + f$

f$: string containing the format for the print using.

A format can define a zone. When bringing or displaying data in this zone, its contents is formatted. The format is stored in a string (here : f$). The control characters are :

9 : numeric,
8 : numeric; set to zero if nil
1 : sign position (only -)
2 : sign position (with + or -)

Any alphanumeric character can be inserted into a format.

Ex : f$ = "$ 19,999,998.88"

Utilization : decz allows one to define, as precisely as possible, the aquisition zone. It is possible to work efficiently and easily on a zone just by using its identification number. The frames or boards management is simplified.

Data Acquisition Instructions

cz

Example : decz 2,1,2,10,n,p,"199 999.88"

This exemple defines a zone (number 2) with a starting point (1,2) of 10 characters. Only numbers will be accepted (n). Printusing format is used with the printing of a sign only if it is a minus and zero being forced at the end of the number. The number will be 5 digits long with 2 digits after the decimal point.

Notes :

. More than one type can be combined in the same decz.
all these different possibilities are accepted.

n	: Only numbers	-RETURN to leave the zone
n,r	: Only numbers	-A programmable key to leave the zone
n,p	: Only numbers using a format	-RETURN to leave the zone
n,r,p	: Only numbers using a format	-A programmable key to leave the zone
m	: Only capital letters	-RETURN to leave the zone
m,r	: Only capital letters	-A programmable key to leave the zone
r	: Every character accepted	-A programmable key to leave the zone
p	: (equivalent to n,p)	
r,p	: (equivalent to n,r,p)	

(nothing): every character is accepted. RETURN to leave the zone.

If the data overflows the aquisition zone, the zone is highlighted, and the whole zone erased. The cursor moves back to the beginning of the zone . This prevents leaving the zone if the format is not correct (the format is controlled when you press RETURN).

If you try decz instructions without using a reqz after it, nothing will happen but the declaration will remain.

reqz

Function : reqz performs the acquisition using the parameters defined in the corresponding decz.

Syntax : reqz n
With n : zone identification number

Utilization : reqz is used in association with decz. If a corresponding decz doesn't exist (or hasn't been loaded in a screen-page), an error message will be generated.

With this command, THE TOOL takes the data acquisition under its control, as defined in the zone declaration. On execution of this command results a cursor displayed at the beginning of the zone.

The CLR, HOME, CURSOR BACK and FORWARD, INST/DEL functions can be used in the zone.

The data acquisition is controlled as defined in the zone declaration.

If the mode to define the zone is r, and the exit code different from 13 (zo), the cursor will return to the position occupied on exit from zone if this zone is re-entered.

Example : If the previous example has been entered (decz 2,...), juste type

reqz 2

to see what happens. Only numbers will be accepted.

inz

Function : inz transfers the contents of a zone to a string variable.

Syntax : inz n,a$
With n : zone identification number
a$: any string variable.

Utilization : inz is used to transfer zone contents to a string when all data in a zone is correct.

Example : inz 2,a$
If the two previous examples have been tried, this one should have transfered the contents of the zone into a$. To verify, just type

```
out a$,3,5

10 decz 1,25,10,5,r
20 reqz 1:inz 1,a$
30 print zo, a$
40 goto 20
```

Notes
. The string variable receiving the contents of a zone will have the same length as the zone, regardless of the contents of the zone.

outz

Function : outz displays a string in a zone.

Syntax : outz n,a$
with n : zone identification number
a$: any string variable.

Utilization : outz is the opposite instruction of inz. Data goes from a variable to a zone. This instruction is similar to the display instruction out but it is easier to use and more precise.

Easier because the zone number instead of the coordinates of the starting point is required.

More precise because it assumes all the controls defined in the zone by the decz.

Example : still using the previous examples
a$ = "11111"
outz 2,a$

Notes :

- in case of a zone defined in printusing
if an overflow of decimals occurs, the number is truncated;
if an overflow of capacity occurs, the zone is filled with"*****".

- in case of an outz in a zone declared with no sign (printusing without 1 or 2), of a string with a +,- or blank as a or blank as first character, the string can be shortened by a digit, or the display can be replaced by stars. To avoid that, you have to extract the sign by doing:

```
a = len(mo$) - 1 : mo $ = right$(mo$,a)
```

clearz

Function : clearz clears a zone.

Syntax : clearz n1 [to n2]
with n1,n2 zone identification numbers.

Utilization : clearz can be used to "clear" the contents of a frame without clearing the frame itself, after an acquisition and before a new acquisition.

Example : clearz 2

carget

Function : carget stops the execution of a program and waits for a pre-defined character.

Syntax : carget a$ [, l, c]
With a$: string containing the allowed characters.
l,c : (optionnal) position on the screen where a cursor will appear.

Utilization : carget replaces the famous BASIC line:
100 get a$: if a$="" then 100

and adds many fine features.

100 carget "abc"

will stop the execution of the program until a, b, or c is entered.

After a carget, the variables ok and zo take the following values.

ok - pointer of position within the string of the choosen character. (a ok=1, b ok=2, c ok=30)
zo - ASCII value of the choosen characters.
(a zo=65, b zo=66, c zo=67)

Example : BASIC

```
100 get a$:if a$="" then 100
110 if a$="A" then gosub 1000:goto200
120 if a$="B" then gosub 2000:goto200
130 if a$="C" then gosub 3000:goto200
140 if a$="Z" then gosub 4000:goto200
150 goto 100
200 rem ::::::the program goes on::::
```

THE TOOL

```
100 a$="ABCZ":carget a$,24,20
110 on ok then gosub 1000,2000,3000,4000
200 rem ::::::::the program goes on::::
```

Note:

If a$ is empty (a$=""), then all the characters are accepted.

```
100 carget ""
```

This will stop the program until any key is stroke.

If you're using carget with l,c parameters, you can define the flashing cursor speed by poke 703,n where n is the value of the speed (normal flashing = 100)

OVERVIEW

In a business application, the data acquisition frames are often similar and are used more than once with different information. In a normal BASIC program, a new frame has to be created each time, wasting both time and memory.

THE TOOL defines a screen page, with all the decz (declarations) instructions, and saves the page on disk, avoiding the necessity of creating a frame over and over again. Different applications can have their own screen page. Coding is simplified.

The page can be loaded on the screen later and filled with data. It is possible to exchange an actual screen page with a page on disk, for exemple, a HELP screen.

If during an acquisition, the control commands are forgotten, the program can easily exchange pages for a HELP page, by saving the actual page on disk, loading the HELP screen on the screen. When the verification is done, the reverse operation can be executed and the program goes on at the same point. The cursor will reappear at the exact point it was before the page exchange.

The three instructions perform the following action :

- saving a page from the screen	ssave	3 - 18
- loading a page to the screen	sload	3 - 19
- clearing a screen	sclear	3 - 20

NOTE:

As we will see with more details in the next chapter, sload and ssave can be used in Text or Graphic Mode.
A text screen-page will be 8 or 9 block-large depending on the number of zones.
A graphic screen-page will be 33 blocks large. You can load a page which doesn't correspond to the current mode (like to load a graphic page in text mode). It will load the page specified in the correct memory part.

ssave

Function : ssave saves a screen page on disk.

Syntax : ssave du, "name" [,n1] [,n2 to n3] [,..]
with
du : disk unit logical number (usually 8)
n1,n2,n3 : associated zones

Utilization : The current screen page will be saved. The zones (decz) can also be saved, using the syntax, n1 or , n2 to n3. This syntax is optional. In all cases, the entire page will be saved (frames, boards, data or display...)

Example : ssave 8,"prg1",1,12,30 to 70

Note : 1) ssave 8,"name",0 to 127

would save all associated declarations. There are no errors if non-declared zones are saved.

2) ssave 8,"@0:prg1" can be used.

3) ssave save the current screen display (text or graphic)
in case of graphic mode, associated zones give a syntax
error.

sload

Function : sload loads on the screen a page saved on disk.

Syntax : sload du,"name"
with
du : disk unit logical number (usually 8)

Utilization : Same principle as ssave.

The page will be directly displayed on the screen. All the zones will be declared again.

Example :
sload 8,"prog1"

Note : none

sclear

Function : sclear clears a page on the screen.

Syntax : sclear

without parameters.

Utilization : The sclear instruction will clear the screen, including the data that have been acquired by the decz, reqz , inz cycle. However sclear will not destroy the data declarations.

Example : sclear

Note : None

Screen Colour Definition

screen

Function : screen allows the modification of the colors of the screen, the borders and the characters displayed by THE TOOL.

Syntax : screen sc,br[,cr]
with sc : color of the screen (0<=sc<=15)
br : color of the border (0<=br<=15)
cr : color of the characters (0<=cr<=15)(optional)

Utilization :

Using the screen instruction, you can easely define the colors of your screen, border and characters displayed by THE TOOL (using tline, tcol, out, outz and reqz -see these instructions).

If the color mode is BLUE for the characters, when the 64 is turned on, the modification of this color using a SCREEN will be effective only for the characters displayed by THE TOOL instructions.

Examples :

```
10 rem :::: let's make some national flags::::::::::::
20 sclear:screen 1,6,2:out"U.S.A",12,20:carget""
30 sclear:screen 1,5,2:out"ITALY",12,20:carget""
40 sclear:screen 0,2,7:out"GERMANY",11,7:carget""
50 sclear:screen 1,2,2:out"JAPAN",11,9:carget""
```

Note : 1) Colour code table

Parameters	Color	Code
	Black	0
	White	1
FOR	Red	2
	Cyan	4
	Purple	5
	Green	5
SC	Blue	6
	Yellow	7
	Orange	8
	Light Orange	9
BR	Pink	10
	Light Cyan	11
and	Light Magenta	12
	Light Green	13
CR	Light Blue	14
	Light Yellow	15

2) STOP and RESTORE will give back the original colors black screen and white character.

Overview

We have briefly introduced in the previous chapter the distinction between the TEXT mode and the GRAPHIC mode. The last chapter described the screen generator with the display, data acquizition, and screen page management instructions which work with the normal 40 by 25 screen corresponding in THE TOOL to the TEXT mode.

We are going now to see the instructions thatr allow charts and graphs to be drawn using the high-resolution screen with a definition of 320 by 200. The high-resolution screen is defined as a drawing board with its origin in the lower left corner. Any point in the high-res. screen is defined by a horizontal coordonate and a vertical coordonate.

graphic

Function : graphic will tell the 64 that it must work now with the high-resolution graphics, e.g. GRAPHIC mode.

Syntax : graphic
without parameter.

Utilization : The graphic instruction has to be used each time a set of graphic instructions is going to be executed. Graphic will reserve the necessary memory to manage the high-resolution graphics.

Example:
```
10 tline 40,1,1 : tline 40,25,1
20 tcol 25,1,1 : tcol 25,1,40
30 ssave 8,"frame"
40 graphic
50 rem a set of graphic instructions (see next pages)
60 ssave 8,"graphic"
70 rem we are still in graphic mode.
```

Note : When a screen management instruction must be used (but ssave, sload, sclear) which is to say, when you want to go back to the TEXT mode, just use the TEXT instruction (page 4-7)

move

Function : move will put the graphic cursor in any part of the screen.

Syntax : move x,y
with : x - horizontal coordonate (0 (= x (= 319)
y - vertical coordonate (0 (= y (= 199)

Utilization : Move allows you to start to plot a point or draw a line from any point on the screen. The move instruction just moves the cursor. There is no output on the screen.

Example:
```
10 graphic
20 move 0,0 :rem moves at the origin (lower hand corner)
30 i=160 : j=100 : move i,j
40 text
50 rem more complete example in the next pages
60 ...
```

Note : The origin point is different from the screen generator. This is due to a practical usage. Data display and acquisition are always done top to bottom, so the origin in TEXT mode is the upper left corner. However, when drawing graphics, a more logical origin point is the lower left corner.

draw

Function : draw will plot (or erase) a line from the point where the cursor is actually (after a move for exemple) to the x,y coordonates.

Syntax : draw x,y,ty
with : x - horizontal coordonate (0 (= x (= 319)
y - vertical coordonate (0 (= y (= 199)
ty- any numeric variable
1 : traces a line.
0 : erases a line.

Utilization : draw is used each time a straight line has to be drawm (or erased) on the screen. Used with the move instruction, it allows any kind of graphics line.

Example:
```
05 rem :::::draw a 3-D cube:::::::::::::::::::::::
10 graphic:sclear
20 a=0:b=20:c=40:d=100
25 move a,a :rem moves at the origin (lower hand corner)
30 draw a,d,1:draw d,d,1:draw d,a,1:draw a,a,1
40 draw c,b,1:draw c+d,b,1:draw c+d,b+d,1
50 draw c,b+d,1:draw a,d,1
60 move d,a:draw d+c,b,1
65 move c,b:draw c,b+d,1
70 move d,d:draw d+c,d+b,1
80 carget "" : rem hit any character to resume
90 text
```

Note : 1) If you add to the previous example, lines 130, 140, 150, 160, 165 and 170, same as 30 to 70 but with 0 instead of 1, you will erase the cube, line by line. The sclear instruction will do the same thing more quickly (see 3 - 20).

2) The carget instruction (see 3-16) without line and column number can also be used in a graphic program.

3) We could also have used

draw a,d,dr

with dr=1 - instruction will draw
dr=0 - instruction will erase

plot

Function : plot will draw (or erase) a point at the x,y coordonates.

Syntax : plot x,y,ty
with : x - horizontal coordonate (0 (= x (= 319)
y - vertical coordonate (0 (= y (= 199)
ty- any numeric variable
1 : traces a point.
0 : erases a point.

Utilization : draw is used when something other than a line has to be drawm (or erased) on the screen.

Example:
```
05 rem :::::draw a circle in a square::::::::::::
10 graphic
20 a=0:d=100:e=50:pi=3.14159:pr=.01:dr=1
25 move a,a :rem moves at the origin (lower hand corner)
30 draw a,d,dr:draw d,d,dr:draw d,a,dr:draw a,a,dr
40 for i=0 to 2*pi step pr
50 plot e*(1+cos(i)),e*(1+sin(i)),dr
60 next i
80 carget "" : rem hit any character to resume
90 text
```

Note : If you run the same program with dr=0 (see line 20), you will erase the circle and the frame. A graphic stays in memory until a sclear or its clearing by draw - type 0.

point

Function : point will test if a point is (or is not) on the screen
at the x,y coordonates.

Syntax : point x,y,ty
with : x - horizontal coordonate (0 <= x <= 319)
y - vertical coordonate (0 <= y <= 199)
ty- any numeric variable - will contain
1 : presence of a point
0 : absence of a point.

Utilization : point is used to check the presence or absence of points on the screen. This is very useful when managing intersections.

Example:
```
05 rem ::::::::::::::::::::::::::::::::::::::::::
10 graphic
20 a=0:d=100:e=50:pi=3.14159:pr=.01:dim a(100)
25 move a,a :rem moves at the origin (lower hand corner)
30 draw a,d,1:draw d,d:draw d,a,1:draw a,a,1
40 for i=0 to 100
50 point 25,i,a(i)
60 next i
80 carget "" : rem hit any character to resume
90 text
100 for i=0 to 100: print a(i);:next i
110 rem a(0)=1 : a(100)=1 : vertical lines of frame
```

Note : None.

display

Function : display will print any string of characters on a graphic screen using the screen generator convention (l,c) on 40 columns and 25 lines.

Syntax : display a$, l ,c
with : a$- any string of characters
l - line # of the starting point (1 <= l <= 25)
c - column # of the starting point (1 <=c<= 40)

Utilization : display is a very powerful instruction allowing you to mix high-resolution graphic and characters. It is used exactly like the OUT instruction (see 3 - 6).

Example:
```
05 rem :::::same as previous exemple + line 70 and 75:::
10 graphic
20 a=0:d=100:e=50:pi=3.14159:pr=.01
25 move a,a :rem moves at the origin (lower hand corner)
30 draw a,d,1:draw d,d:draw d,a,1:draw a,a,1
40 for i=0 to 2*pi step pr
50 plot e*(1+cos(i)),e*(1+sin(i)),1
60 next i
70 display "This is a circle",5,10
75 display "----------------",6,10
80 carget "" : rem hit any character to resume
90 text
```

Note :

text

Function : text will stop the high-resolution mode and come back to the TEXT mode for data display and acquizition.

Syntax : text
without parameter

Example:
```
05 rem :::::same as previous exemple + line 70 and 75:::
10 graphic
20 a=0:d=100:e=50:pi=3.14159:pr=.01
25 move a,a :rem moves at the origin (lower hand corner)
30 draw a,d,1:draw d,d:draw d,a,1:draw a,a,1
40 for i=0 to 2*pi step pr
50 plot e*(1+cos(i)),e*(1+sin(i)),1
60 next i
70 display "This is a circle",5,10
75 display "----------------",6,10
80 carget "" : rem hit any character to resume
85 ssave8,"@0:circle.grf"
90 text
95 sclear
96 tline 40,1,1:tline 40,25,1:tcol 25,1,1:tcol 25,1,40
98 ssave8,"@0:frame.txt"
100 end

list
```

Note : 1) The TEXT mode is the mode when the system is powered on.

2) If a Graphic instruction is done in Text mode, it will be executed on the graphic screen; but will be seen only in graphic mode.

3) If a Text instruction is done in Graphic mode, it will be executed expect for ssave and sclear instructions.

color

Function : color is use to put a color matrix of 8 points by 8 points in the graphic memory.

Syntax : color [-] l,c,lg,ln,co
with : [-] optionnal character
: l - line # of the starting point (1 <= l <= 25)
c - column # of the starting point (1 <= c <= 40)
lg - height (or number of lines)
ln - lenght (or number of columns)
co - new color (0 to 15 - see page 3-21)

Utilization : color is an instruction wich allowing you to define a color block in the graphic screen.
It is used exactly like the REV instruction (see 3 - 7).

Example:
```
05 rem :::::same as previous exemple + line 70 and 75:::
10 graphic
20 a=0:d=100:e=50:pi=3.14159:pr=.01
25 move a,a :rem moves at the origin (lower hand corner)
30 draw a,d,1:draw d,d,1:draw d,a,1:draw a,a,1
40 for i=0 to 2*pi step pr
50 plot e*(1+cos(i)),e*(1+sin(i)),1
60 next i
70 color-1,1,25,40,7
75 color1,1,25,40,6
80 carget "" : rem hit any character to resume
90 text
```

Note :
The graphic screen is a set of points wich they are light on or light off. The line 70 is used to put the light off points in yellow color and the line 75 is used to put the light on points in blue color.

auto

Function : auto numbers automatically the lines of a program.

Syntax : auto n
with n : increment (1<=n <=255)

Utilization : To enter the auto line numbering, you just need to type auto and return. Then you start to type your program with you first line. After having typed return on that first line, the next line number will appear.

You still can put any number you want to change the automatic line number.

To stop the auto line numbering, just enter the command auto without any argument. It will also stop if you type return after a line just composed by a line number.

Example :
```
auto 10
10 for i = 1 to 10
20
```

Note : auto can be used only in direct mode.

delete

Function : delete deletes lines of a program.

Syntax : delete a
delete a-
delete -a
delete a-b

with a,b : line numbers.

Utilization : delete suppresses lines with the same syntax as the LIST function

1) delete a suppresses the line a.

2) delete a- suppresses all lines between the line a and the end of the program.

3) delete -a suppresses all lines from the beginning to the a line.

4) delete a-b suppresses all lines between the line a and the line b.

Example : delete 20

Notes : - delete without argument is not valid.
- delete can be used in direct mode and program mode.

renu

Function : renu renumbers the lines of a program.

Syntax : renu [a[,b[,c]]]
with all 3 arguments optional
(see utilization).

Utilization : renu will renumber all lines of a program
(including GOTO, GOSUB, etc...)

Four possibilities :

1) renu
Renumbers the whole program
The first line becomes 100
The line increment is 10

2) renu a
Renumbers the whole program
The first line becomes 100
The line increment is a

3) renu a,b
Renumbers the whole program
The first line becomes a
The line increment is b

4) renu a,b,c
Renumbers a part of the program starting at line a
That line a becomes b
The line increment is c

Example :

renu 20,10

This will renumber a whole program, the first line becoming 20, with a line increment of 10.

Note : only in direct mode

dump

Function : dump gives the list of the variables used in a program and their values.

Syntax : dump

Utilization : dump can be used at any time
- after a run of a program,
- after a STOP (and you can continue:CONT),
- after a syntax error.

Dump does not give the content of arrays to avoid an overload of the screen.

Example : a = 2 : b = 3 : a$ = "test"

dump

will give

```
a  = 2
b  = 3
a $="test"
```

Note : can be used in program and direct mode.

error

Function : error gives the location of an error in a BASIC line.

Syntax : error
with no argument

Utilization : When an error occurs in a program, BASIC stops the execution and gives an error message. To locate the error, just type

error

Error will display the faulty line and the error will be flagged in reverse video mode.

Example :

```
10 a$ = STR$(a
20 end
run
error
```

Note : only in direct mode.

No other commands must be used before error when the interruption occurs. (even LIST).

find

Function : find finds any string of characters within a program.

Syntax : find (de) ch (de) [A - B]
with (de) : any character to be used as a delimitor but " and a character within the string you are looking for.

ch : characters you are looking for

A-B: (optional)
A : line number where the search starts
B : line number where the search stops.

Utilization : Just give any character or string of characters and using the find instruction, you will be able to find it. This can be very useful when you are correcting syntax errors. Just search for the name with the error.

Example :

```
10 ?"WRITE"
20 ?"RED"
30 ?"BLUE"
40 ?"GLU"
50 END

FIND /"GLU"/ : rem or FIND @glu@
```

Note : can be used in program and direct mode.

Step-by-Step Execution Mode

trace
off

Function : trace allows to execute a program with one instruction at a time.
off resumes the trace function.

Syntax : trace with no argument
off with no argument

Utilization : When the trace mode is used, the program runs one instruction at a time. The executed line is displayed on the screen.

- to display the line, type a shift key at each instruction.
- to go to the next instruction release the same shift key.

- to stop the trace program in direct mode, press both a shift and the RUN keys, being careful to release first the Shift key to avoid the loading of a program.

Example : In program mode, you suspect to have an error in the subroutine 1000. To trace it :

```
100 rem your program
110 trace
120 gosub 1000
130 off
140 rem the rest of your program.
```

Note : can be used in program mode and direct mode.

Overview :

We have grouped under this chapter all the annex functions which are not running with the screen. These instructions simplify the controls, improve some performances of the BASIC and ease the programming especially for the work on string of characters.

These functions include :

These functions are all independant and can be studied separately.

hunt

Function : hunt will find the position of a character within a string.

Syntax : hunt [-] ca$,a$,pn
with - : optional symbol
without: search for the first character equal to,
with : search for the first character non equal to
ca$: string to find
a$: string where the search is to be done
pn : pointer within the string to indicate where the search starts.

Utilization :

hunt allows control operations (acquisition, validation,...) and also statistics. The result will be found in zo.

Example :
```
a$ = "patricia owen"
hunt "a",a$,1
? zo
2
hunt-"e",a$,1
?zo
1
```

Note : If the character does not exist, zo=0

creatst

Function : creatst creates a string.

Syntax : creatst a$,ln [,ca$]
with
ca$: string of one character
a$: name of the string
pn : length of the string to be created

Utilization :

The creatst command creates a string of ln characters long (ca$). In the absence of ca$, chr$(32) will be used. You can used any character you want.

Example : creatst a$, 80, " "
will create a string of 80 blanks
creatst b$,10,"*"
will create a string b$ with 10 stars.

Note : This instruction replaces the following BASIC loop.

```
for i= 1 to ln
a$=a$+ca$
next i
```

if then else

Function : if_then_else is an improved version of the BASIC allowing better programming.

Syntax :
```
10 if (condition) then (instruct.1) :else (instruct.2)
or
10 if (condition) then (instructions 1)
20 else (instructions 2)
```

Utilisation : If a condition is true, then the set of instructions will be executed. If the condition is untrue, then the instructions 2 will be executed. Up to 16 nested if-then-else are possible.

Example:
```
10 if (c1) then (i1)
20   if (c2) then (i2)
30     if (c3) then (i3)
40     else (e3)
50   else (e2)
60 else (e1)
```

Example:
```
10 a=2
20 a=a+1
30 if a=2 then ?"a=2" : else ?"a=3"
40 end
```

Run this program twice; the first time with the line 20, the second without.

Note : 1) If then else can be used only in program mode.
2) Up to 16 nested if-then-else are possible, and the number of elses must be equal to or smaller than the number of if .

hcopy

Function : hcopy hardcopies the contents of the screen on the printer. If that printer allows the high-resolution graphics, then any graphic on the screen will be printed.

Syntax : hcopy

without parameter in progam mode

Utilisation : As simple as it looks.

Example: Take any previous example, and add a line like
110 hcopy

Note : .The use of that function will stop the internal clock of the 64 during its execution.

.Poke 701,255 allows you to get double width characters on your printer 1515. Poke 701,0 disable the double width mode.

.Poke 702,255 allows you to get 9 lines per inch for semi-graphics characters. Poke 702,0 put the printer in 6 lines per inche mode.

joy(1),joy(2)

Function : joy(1) is not an instruction but a BASIC function in which can be found the values given by a joystick

Syntax : joy(1) joy(2)

with no parameters.

Utilisation : That variable allows to use one or two joysticks without having to peek in memory for programming easily any application using joysticks.

Example : Put your joystick in the right port (first port on the left of your 64)

```
10 ?joy(1): goto 10
run

10 rem ::::draw on the high res. screen using a joystick:::
11 graphic:sclear
12 x=125:y=199:ad=128:plot x,y,1
20 cd=joy(1):ad=cd
30 if cd=0 then 20
40 if ad<128 then plot x,y,0
50 if cd>128 then cd=cd-128
60 on cd gosub 110,120,100,140,150,160,100,180,190,200
70 plot x,y,1:goto 20
100 return
110 y=y+1:return
120 y=y-1:return
140 x=x-1:return
150 x=x-1:y=y+1:return
160 x=x-1:y-y-1:return
180 x=x+1:return
190 x=x+1:y=y+1:return
200 x=x+1:y=y-1:return
```

Note : 1) Used with the graphic instructions, the function joy(1) allows really nice programming. See demonstration program CA01 which creates with a joystick semi-automatic flowcharts.

2) JOY(2) corresponds to a second joystick using the second port. It is used exactly in the same way as JOY(1).

Overview

THE TOOL simplifies the operations for all disk user by including the DOS support which will allow you to use quick commands to manage the loading, saving, (and so on) on diskette.

@ : will give you the disk error message,

@$: will give you the catalog of your diskette,

@I : will initialize the floppy unit,

@V : will perform a collect,

@N : filename, identifier
: will format a diskette,

@R : newfilename = oldfilename
: will rename a file or program,

@S : filename
: will scratch a file or program.

@C : newfilename = oldfilename
: will make a copy of a file.

All commands are to be followed by hitting the RETURN key.

Note : For more details about all these operations and their syntax you can refer to your VIC-1540 or 1541 User's manual.

INSTRUCTIONS

```
carget a$ [,l,c]          - "clever" GET of a pre-defined char.3-16
clear l,c,lg,ln           - erases ln characters
                            (origin point :l,c) 3-5
clearz n                  - erases the nth zone.3-15
decz n,l,c,ln[,ty][,f$]   - declares the nth zone.3-10
inz n,a$                  - transfers the content of the
                            nth zone into a$3-13
out a$,l,c                - displays a$ (origin point :l,c)3-6
outz n,a$                 - displays a$ into the nth zone
                            and contols a$3-14
reqz n                    - requires an answer in the nth zone3-12
rev l,c,lg,ln[,co]        - reverses the video mode on part of the
                            screen3-7
sclear                    - clears the screen3-20
screen sc,br[,cr]         - modifies the colors of the screen, the
                            borders and the characters      3-21
scroll l,c,ln,lg,ty       - scrolls a part of the screen3-8
sload 8,"name"            - loads a screen page in memory (screen)3
-19
ssave 8,"name"[,n][,n2ton3]
                          - saves a screen page on disk with
                            the zones3-18
tcol ln,l,c               - draws a line of length ln (origin l,c)3
-4 tline ln,l,c           - draws a col. of length ln (origin l,c)3
-3
```

Reserved variables

```
zo                        - ASCII code for the return key in
request
                            mode (decz)
ok                        - pointer in string.
```

Not-Reserved Variables (Programmer's Choice)

```
a$                        - string - transfer
c                         - column number (x-coordonates)
ca                        - ASCII value of a character
co                        - colour (for the rev instruction)
f$                        - print-using format
l                         - line number (y-coordonates)
lg                        - width(or height) of a window(zone)(-
number
                            of lines).
ln                        - length of a window (zone) - number of
                            columns
                          - length of a string (creatst)
n1,n2,n3,n                - zone identification number
ty                        - formatting type (decz)
                          - scrolling type (scroll)-(u,d,l,r)
```

List of the error messages generated by THE TOOL with an explanation of the problem

BAD FORMAT

decz using the p-type :
the print-using defined does not correspond to the
variables used - length problems (see 3-10)

NO COLOR

A color parameter is missing or the value of a parameter is not allowed. (see instruction 3-21)

NO ZONE

A z-type instruction has been performed on an un-declared zone (see 3-9 to 3-15)

OUT OF PAGE

coordonates of the origin point or length or heig ht are incorrect in the screen generator instruct ions.

You are a THE TOOL user. We wish to know your appreciation of the product qualities and problems. Your comments will assist MICRO APPLICATION in improving its products and in serving you better.

If you wish a reply or/and if you want to be on MICRO APPLICATION mailing list, provide your name and address in this space.

Name :
Address :
City : Country (state) :
Code :

MASTER

Which are the most useful functions ?

Which are the least useful functions ?

What improvements would you like to see ?

For which kind of applications do you use THE TOOL ?

Bugs (mistakes) ?

Notes :

DOCUMENTATION

How did you find this manual ?

Strong-points ?

Weak points ?

Notes ?

FUTURE PRODUCTS

What kind of products (hardware or sofware) would you like to see on your computer ?

Notes :

Thank you

Please send this answer sheet to :
MICRO APPLICATION Immeuble "Le Consulat"
147 Av. Paul Doumer - 92100 RUEIL MALMAISON - FRANCE