Text

This Chapter explains how to use the advantages of AOZ for handling written text. You may want to remind yourself of the visible character set by running this simple routine:

```
For 0=32 To 255
    Print Chr$(C);" =Ascii Code";
    Print Asc(Chr$(C)) : Wait 10
Next C
```

Printing on the screen

The PRINT instruction is one of the most familiar command words in most Basic languages.

PRINT

instruction: print items on screen

```
Print items
```

Items are printed on the screen, starting from the current cursor position, and they may include the characters in any group of variables or constants, up to the maximum line length of 255 characters. The PRINT command is also used to display graphics and information on screen, as is demonstrated throughout this User Guide. This Chapter will deal with printing text only.

Print statements can occupy their own lines, but if more than one element to be printed is written as a single line of your program, each element must be separated from the next by either a semi-colon character or a comma. An element to be printed can be a string, a variable or a constant, and is placed inside a pair of quotation marks.

A semi-colon is used to print elements immediately after one another, like this:

```
Print "Follow";"on"
```

A comma moves the cursor to the next "Tab" position on the screen, as follows:

```
Print "Next","Tab"
```

A Tab is an automatic marker that sets up a location for printing, and is often used to lay out columns of figures, or to make indentations in text, and setting Tab positions is explained later. Normally, the cursor is advanced downwards by one line after every PRINT command, but by using the semi-colon or comma, the rule can be changed. Here is an example:

```
Print "AOZ"
Print "2"
Print "A";
Print "OZ",
Print "2"
```

Setting text options

PEN

instruction: set the colour of text

```
Pen index number
```

This command sets the colour of the text displayed in the current window, when followed by the colour index number of your choice. The default setting of the pen colour is index number 2, which is white, and alternative colours may be selected from one of up to 64 choices, depending on the current graphics mode. For example:

```
For INDEX=0 To 15
    Pen INDEX
    Print "Pen number ";INDEX
Next INDEX
```

PEN$

function: return a control index number to set the pen colour

```
p$=Pen$(index number)
```

This function returns a special control sequence that changes the pen colour inside a string. This means that whenever the string is printed on the screen, the pre-set pen colour is automatically assigned to it. The format of the string returned by PEN$ is: Chr$(27)+"Pen"+Chr$(48+number). Here is an example:

```
P$=Pen$(2)+"Well all WHITE, "+Pen$(6)+" I still got the BLUES"
Print P$
Pen 4
Print "In the RED"
```

PAPER

instruction: set colour of text background

```
Paper index number
```

To select a background colour on which your text is to be printed, the PAPER command is followed by a colour index number between 0 and 63, depending on the graphics mode in use, in exactly the same way as PEN. The normal default colour index number is 1, giving an orange background colour, with other possibilities listed under the SCREEN OPEN command in this User Guide. Run the following simple example:

```
Pen 2: For INDEX=0 To 15
    Paper INDEX: Print "Paper number ";INDEX;Space$(23)
Next INDEX
```

PAPER$

function: return a control index number to set background colour

```
b$=PAPER$(index number)
```

Similarly to the PEN$ function, PAPER$ returns a character string that automatically sets the background colour when the string is printed on the screen. For example:

```
Pen 1
B$=Paper$(3)+"Flash Harry"+Paper$(1)+"The Invisible Man"
Print B$
```

Changing text options

INVERSE ON/OFF

instructions : toggle inverse mode of subsequent text

```
Inverse On
Inverse Off
```

The INVERSE instruction swaps over the text and background colours already selected by the PEN and PAPER
commands, and so sets up an inverse mode for printing. For example:

```
Pen 2 : Paper 4: Print "I appear normal"
Inverse On : Print "Poetry inverse"
Inverse Off : Print "Don't be so negative"
```

SHADE ON/OFF

instructions: toggle shading of subsequent text

```
Shade On
Shade Off
```

The appearance of your text can be changed more subtly by introducing a mask pattern that reduces the brightness of the characters when printed. To make use of this shading facility, simply turn it on and off like this:

```
Shade On :Print "Shady Lady"
Shade Off:Print "Norman Normal"
```

Setting text styles

As well as customising the appearance of your text by changing the text options, you can also use the standard typeface techniques available to printers and word processors.

UNDER ON/OFF

instructions: toggle underline mode of subsequent text

```
Under On
Under Off
```

To underline text when printed on screen like this, use the UNDER instructions, as follows:

```
Under On : Print "This is where we draw the line"
Under Off: "That is groundless"
```

In Section XX.X (11.1) there is a full explanation of how to take advantage of any number of different type faces or fonts, by making use of what is known as "graphic text". For the time being, try the next example:

```
Cls: For S=0 To 7: Set Text S
Text 100,S*20+20,AOZ" : Next S
```

SET TEXT

instruction: set the style of a text font

```
Set Text style number
```

The SET TEXT command allows you to change the style of a font by selecting one of eight different styles that are produced by mixing the following three elements

- Bit 0 Underline
- Bit 1 Bold
- Bit 2 Italic

Set the appropriate bits in the form of a style number from 0 to 7, as in the last example.

TEXT STYLES

function: return current text style

```
s=Text Styles
```

This function returns the index reference of the text style you last selected using SET TEXT. The result is a bit-map in the same format as explained above:

```
Set Text 2: Print "Style Two"
Print Text Styles
```

Changing the text mode

For even more flexibility in presenting your text on screen, you can select the way it is combined with other screen data.

WRITING

instruction: select text writing mode of subsequent text

```
Writing value1
Writing value1,optional value2
```

The WRITING command is used to control how the subsequent text interacts with what is already on the screen, and it can be followed by either one or two values.

The first value selects one of five writing modes:

- 0 REPLACE New text replaces any existing screen data
- 1 OR Merge new text with screen data, using logical OR
- 2 XOR Combine new text with screen data, using OR
- 3 AND Combine new text and screen data, using logical AND
- 4 IGNORE Ignore all subsequent printing instructions

A number set as the optional second value selects which parts of the text are to be printed on the screen, as follows:

- 0 Normal Print text and background together
- 1 Paper Only the background to be drawn on screen
- 2 Pen Ignore paper colour and print text on background colour zero

The default value for both of the WRITING parameters is zero, giving normal printed output.

Positioning the text cursor

Characters are always printed at the current position of the text cursor, and the AOZ programmer is
offered several methods of controlling the cursor in order to make text look more orderly, attractive or eye-catching.

LOCATE

instruction: position the text cursor

```
Locate x,
Locate ,y
Locate x,y
```

This command moves the text cursor to the coordinates of your choice, and this new location sets the start position for all subsequent text printing until you command otherwise. All screen positions are measured in "text coordinates", which are measured in units of one printed character on screen, with the x-coordinate controlling the horizontal position and the y-coordinate referring to the vertical. So, the top left-hand corner of the screen has coordinates of 0,0 whereas text coordinates of 15,10 refer to a position 15 characters from the left-hand edge of the screen and 10 characters from the top.

The range of these coordinates will depend on the size of your character set and the dimensions of the display area allocated, known as a "window". All coordinate measurements are taken using text coordinates relative to the current window. If you try and print something outside of these limits, an error will be generated. Windows are dealt with in the next Section, but the current screen is automatically treated as a window, so there is no need to "open" one to test the following examples:

```
Print "0,0": Locate 10, : Print "Stay on current line"
Locate ,5 : Print "Six from the top."
Locate 10,10 : Print "Ten down and ten across"
```

HOME

instruction: force text cursor home

```
Home
```

Whenever you need to move the text cursor back to the top left-hand corner of the screen in a hurry, simply tell it to go HOME and it will automatically be relocated to coordinates 0,0 like this:

```
Cls: Locate 10,10: Print "I am going"
Wait 100: Home : Print "Home!"
```

CMOVE

instruction: move text cursor

```
Cmove width
Cmove height
Cmove width,height
```

It is also possible to move the text cursor a pre-set distance away from its current position, which can come in useful if you need to show speech bubbles or shunt your text to one side temporarily. The CMOVE command is followed by a pair of variables that represent the width and height of the required offset, and these values are added to the current cursor coordinates. Like LOCATE, either of the coordinates can be omitted, as long as the comma is positioned correctly. An additional technique is to use negative values as well as positive offsets. For example:

```
Cls : Print "Iceland"
Cmove 5,5: Print "Scotland";
Cmove ,-3 : Print "Norway"
Cmove 10,14: Print "France"
```

CMOVE$

function: return control string to move text cursor

```
a$=Cmove$(x,y)
```

Characters can be printed relative to the current cursor position by setting up a string using the CMOVE$ function.

The following example prints a string at coordinates 10,10 from the current text cursor:

```
A$=Cmove$(10,10)
A$=A$+"AOZ"
Print A$
```

AT

function: return a string to position the text cursor

```
a$=At(x,y)
```

You may also change the position of the text cursor directly from inside a character string. This is ideal for positioning text once and for all on screen, no matter what happens in the program, because the text cursor can be set during the program's initialisation phase. The string that is returned takes the following format:

```
Chr$(27)+"X"+Chr$(48+X)+Chr$(27)+"Y"+Chr$(48+Y)
```

So whenever this string is printed, the text cursor will be moved to the text coordinates held by X and Y. For example:

```
A$="A"+At(10,10)+"Of"+At(2,4)+"String"+At(20,20)+"Pearls"
Print A$
```

Imagine a Hi-Score table positioned like this:

```
SCORE=999
Locate 12,10: Print "Hi Score ";SCORE
```

By using the AT function, the same table can be moved by editing a single string, no matter how many times it is used in the program, like this:

```
HI_SCORES=At(12,10)+"Hi Score"
SCORE=999
Print HI_SCORE$;SCORE
```

CENTRE

instruction: print text centrally on current line

```
Centre a$
```

Programmers often need to position text in the centre of the screen, and to save you the effort of calculating the text coordinates in order to achieve this, the CENTRE command takes a string of characters and prints it in the middle of the line currently occupied by the cursor. For example:

```
Locate 0,1
Centre "ABOVE"
Cmove ,3
Centre "suspicion"
```

TAB$

function: move text cursor to next Tab

```
t$=Tab$
```

The TAB$ function returns a special control character called TAB, which carries the Ascii code of 9. When this character is printed, the text cursor is automatically moved to the next tabulated column setting (Tab) to the right.

The default setting for this is four characters, which can be changed as follows:

SET TAB

instruction: change Tab setting

```
Set Tab number
```

This simple command specifies the number of characters that the text cursor will move to the right when the next TAB$ is printed. For example:

```
Cls : Print "Home"
Print Tab$;"And"
Set Tab 10 : Print Tab$;"Away"
```

CDOWN

instruction: move text cursor down

```
Cdown
```

Use this command to force the text cursor down a single line, like this:

```
Cls: Print "Over" : Cdown : Print "the Moon"
```

CDOWN$

function: return control character to move text cursor down

```
c$=Cdown$
```

The effect of summoning up the special control character (Ascii 31) is exactly the same as printing after a CDOWN command. The advantage of this alternative is that several text cursor movements can be combined in a single string, using CDOWN$. For example:

```
C$="Going Down"+Cdown$
For A=0 To 20
    Print C$
Next A
```

CUP

instruction: move text cursor one line up

```
Cup
```

CLEFT

instruction: move text cursor one character left

```
Cleft
```

CRIGHT

instruction: move text cursor one character right

```
Cright
```

These three commands are self-explanatory, and work in exactly the same way as CDOWN. their equivalent
functions are listed below, and work in the same way as CDOWN$:

CUP$

function: return control character (30) to move cursor up one line

```
x$=Cup$
```

CLEFT$

function: return control character (29) to move cursor left one character

```
x$=Cleft$
```

CRIGHT$

function: return control character (28) to move cursor right one character

```
x$=Cright$
```

CLINE

instruction: clear some or all text on current cursor line

```
Cline
Cline number
```

This command is used to clear the line currently occupied by the text cursor. If CLINE is qualified by a number, then that number of characters get cleared, starting from the current cursor position and leaving the cursor exactly where it is. For example:

```
Print "Testing Testing Testing";
Cmove -7,
Cline 7
Wait Key
Cline
```

Tracking the text cursor

To track down the exact position of the text cursor, the following pair of functions may be used

XCURS

function: return the x-coordinate of the text cursor

```
x=Xcurs
```

YCURS

function: return the y-coordinate of the text cursor

```
y=Ycurs
```

In this way, a variable is created that holds the relevant coordinate of the cursor, in text format, and these two functions may be used independently or together. For example:

```
Locate 5,10: Print Xcurs; : Print Ycurs
```

MEMORIZE X/Y

instructions: save the x or y text cursor coordinates

```
Memorize X
Memorize Y
```

The MEMORIZE commands store the current position of the x or y text cursor, so that you can print any text on the screen without destroying the original cursor coordinates. These may be reloaded using the REMEMBER commands, as follows:

REMEMBER X/Y

instructions: restore the x or y text cursor coordinates

```
Remember X
Remember Y
```

Use REMEMBER to position the text cursor at the coordinates saved by a previous MEMORIZE command. If
MEMORIZE has not been used, the relevant coordinate will automatically be set to zero. There is a ready-made example demonstrating these commands to be found under the SET CURS command, which is below.

Changing the text cursor

CURS PEN

instruction: select colour of text cursor

```
Curs Pen index number
```

As a default, whenever your screen mode provides four or more colours the text cursor is set to index number 3, which is endowed with a built-in flash. The flashing can be turned off and back on again at any time using the FLASH OFF and FLASH commands, but as soon as you select another colour for your text cursor, the automatic flash will not apply. To change colours, use the CURS PEN command, followed by the index number of your
choice. For example:

```
Curs Pen 2
```

Note that the new colour only effects the text cursor in the current open window, and has no influence over other cursors used by any other windows. If you want to introduce a flash to that last example, you could add this line before the CURS PEN command:

```
Flash 2,"(FFF,15)(000,15)"
```

SET CURS

instruction: set the shape of the text cursor

```
Set Curs L1,L2,L3,L4,L5,L6,L7,L8
```

To customise the text cursor into something a little more personalised, you can change its shape into anything you like, providing you limit yourself to the eight lines of eight bits each that represent its appearance. Lines are numbered one to eight from top to bottom, and every bit set to 1 results in a pixel drawn in the current cursor pen colour, whereas a zero displays the current paper colour. To familiarise yourself with the technique, try the next example, which changes the text cursor into a Hallowe'en mask:

```
L1=%00111100
L2=%01111110
L3=%01011010
L4=%11100111
L5=%10111101
L6=%01011010
L7=%00100100
L8=%00011000
Set Curs L1 L2 , L3 L4 L5, L6 , L7, L8
```

Your routine will appear slightly different from that, because the system automatically strips away any leading zeros in binary listings.

CURS ON/OFF

instructions: toggle text cursor

```
Curs On
Curs Off
```

This pair of commands is use to hide and reveal the text cursor in the current window. It has no effect at all on any cursors used in other windows.

Advanced text commands

ZONE$

function: create a zone around text

```
z$=ZONE$(text$,zone number)
```

The AOZ programmer is allowed to create powerful dialogue boxes and on- screen control panels without the need to employ complex programming. The ZONES function surrounds a section of text with its own screen zone, so that the presence of the mouse pointer can be detected using the ZONE function. Simply supply the two parameters in brackets, which are the string of text for one of your control "buttons", followed by the number of the screen zone to be defined.

The maximum number of zones will be limited by the value specified in a previous RESERVE ZONE command.
The format for the control string is as follows:

```
Chr$(27)+"ZO"+A$+Chr$(27)+"R"+Chr$(48+n)        TODO: check new syntax.
```

BORDER$

function: create a border around text

```
b$=Border$(text$, border number)
```

This works in much the same way as ZONES, by returning a string of characters that create a border around the required string of text. The AOZ programmer can use it with ZONES to set up special "buttons" for alert windows and control consoles.

In this case, the text held in the string will start at the current text cursor position. Border numbers can range from 1 to 16, for example:

```
Locate 1,1: Print Border$("AOZ",2)
```

The control sequence returned by BORDER has the following format:

```
Chr$(27)+"EO"+A$+Chr$(27)+"R"+Chr$(48+n)    TODO check return value today!
```

HSCROLL

instruction: scroll text horizontally

```
Hscroll number
```

This command scrolls all text in the current open window horizontally, by a single character position. The following numbers can be used:

- 1 Scroll current line to the left
- 2 Scroll entire screen to the left
- 3 Scroll current line to the right
- 4 Scroll entire screen to the right

VSCROLL

instruction: scroll text vertical

```
Vscroll number
```

Similarly to HSCROLL, the values given to this command result in different vertical scrolling effects, one character at a time.

- 1 Scroll down text on and below current cursor Line
- 2 Scroll down text from top of screen to current cursor line only
- 3 Scroll up text from top of screen to current cursor line only
- 4 Scroll up text on or below current cursor line

Note that blank lines are inserted to fill any gaps left by these scrolling operations.

Advanced printing

The AOZ programmer is not restricted to the standard PRINT command for displaying information.

?

instruction: print

```
Print
```

The question mark character (?) can be used instead of PRINT as a keyboard short-cut.

When used in this way, it is automatically displayed as PRINT as soon as the line has been entered into your listing.

`"AOZ"`

USING

instruction: format printed output

```
Print Using format$;variable list
```

USING is always employed with the PRINT command to allow subtle changes in the way output is printed from a list of variables. The format string contains special characters, and each one has a different effect, as explained below.

- '~' : tilde character. Every ~ in the string variable is replaced by a single character from left to right, taken from an output string. For example:

```
Print Using "This is a ~~~~~~ example";"simple"
```

- '#' : hash character. Each # specifies one digit at a time, to be printed out from a given variable, with any unused digits being replaced by spaces. For example:

```
Print Using "###";123456
```

- '+' : plus character. This adds a plus sign to a positive number or a minus sign if the number is negative. For example:

```
Print Using "+##";10 : Print Using "+##";-10
```

- '-' : minus character. This gives a minus sign to negative numbers only. Positive numbers will be preceded by a space. For example:

```
Print Using "-##";10:Print Using "-##";-10
```

- '.' : full stop character. When used with PRINT USING, the full stop (period) character places a decimal point in a number, and automatically centres it on screen. For example:

```
Print Using ".###";Pi#
```

- ';' : semi-colon character. This centres a number, but will not output a decimal point. For example:

```
Print Using "Pl is #;###";Pi#
```

- '^' : exponential (circumflex) character. This causes a number to be printed out in exponential format. For example:

```
Print Using "This is an exponential number^";10000*10000.5
```

Sending text to a printer

Chapter XX.X (10.3) is devoted to the exploitation of the printer device by AOZ. The following command offers easy access to a printer from inside an AOZ program or via Direct mode.

LPRINT

instruction: output a list of variables to a printer

```
Lprint variable list
```

The LPRINT command is exactly the same as a PRINT command, but it sends data to a printer instead of the
screen, like this:

```
Lprint "Greetings from AOZ!"
```