TODO - "this chapter was skipped, to be converted?'

Interface Language

This Chapter deals with all of the AOZ Interface general purpose programming commands.

The Interface is a complete language, and includes a full set of general instructions that can be used to great effect in dialogue boxes. There is an extensive range of graphics commands, a testing facility and a pair of important
program control instructions for making jumps. The Interface even provides fully-operative procedures and userdefined functions!

The graphics functions

When defining a number of buttons, it is important to be able to arrange them neatly as part of the display. The AOZ Interface provides a number of simple functions to shoulder the burden of these problems.

As usual, these functions read their values directly from the number stack, so the numbers go before the operations.

BaseX

BaseY

Interface functions: get the coordinate base location

```
x=BX
y=BY
```

These functions are used to return the screen coordinates that are to be used as the starting point for all future calculations. These values are assumed to have been previously set with the BAse instruction.

SizeX

SizeY

Interface functions: get the size of the dialogue box

```
width=SX
height=SY
```

Use these functions to provide the precise dimensions of the current dialogue box, as set with the SIze command.

Screen Width

Screen Height

Interface functions: read dimensions of the current screen

```
width=SW
height=SH
```

The SW and SH functions find the height and width of the current AOZ screen. They can be used in conjunction with the SX and SY functions to position a dialogue box neatly in the centre of the display. For example:

```
SIze 240.100;
BAse SW SX 2/-,SH SY 2/-;
```

The next pair of functions keep track of the graphics cursor. This is automatically positioned at the location of the last drawing operation on the screen.

XA

YA

Interface functions: get previous coordinates of graphics cursor

```
x=XA
y=YA
```

XA and YA retain a copy of the graphic coordinates, at their position before the most recent graphics operation was performed. This pair of values can be very useful when objects need to be defined relative to one another, as you will only have to set the coordinates of the first object, and all of the others will be relative to it. For example:

```
GraphicSquare 10,10,30,30; draw a box at 10,10
GraphicSquare XA,40,XA 20+,30; draw a box immediately below at 10,40
```

XB

YB

Interface functions: get current coordinates of graphics cursor

```
x=XB
y=YB
```

These two functions complement the previous pair, and they return the position of the graphics cursor after the execution of the most recent instruction. Here are some schematic examples:

```
BUtton 1,160,100,...;[][]
BUtton 2,XB,YA,...;[][]
PRint 0,8,"Hello, first line",2;
PRint XA,YB,"This line will be directly under!",2;
```

The Graphics Commands

Here are all of the AOZ Interface graphics commands. They fall into the following main groups: boxes and bars, lines and outlines, and the text commands. Most of these Interface instructions are very similar to their normal AOZ equivalents.

Boxes and Bars

GraphicBox

Interface instruction: draw a filled box

```
GB x1,y1,x2,y2;
```

To draw a box that is filled with the current colour, the GraphicBox command is followed by familiar top left-hand corner coordinates, and then the coordinates of the diagonally opposite corner.

All coordinates are measured from the start location of the dialogue box, which can be set by the Interface instruction BAse, or the AOZ command DIALOG BOX. The default position is the top left-hand corner of the screen.

INk

Interface instruction: set current drawing colours

```
IN pen,background,outline;
```

The colours for boxes and bars and all future drawing operations are set by the INk command. The three parameters are as follows: The pen parameter is set by the colour index number to be used for drawing all future items.

The background colour is then chosen for filling bars and for the paper colour on which text is to be printed. This option is usually ignored, because the background is completely transparent, but it may be activated using a SetWriting mode, which is explained later.

The third parameter to be set is for the outline colour of a bar. This is only relevant if the outline mode has been switched on, using a previous SetPattern instruction, also explained later.

If one of these settings is to be left unchanged, simply enter a negative value for the relevant colour number. As you may expect, a wide range of fill patterns can also be used.

SetPattern

Interface instruction: set the fill pattern for dialogue box

```
SP pattern number,outline mode;
```

The Interface SetPattern command is a combination of the normal SET PATTERN and SET PAINT instructions, and it is used to set the fill pattern and toggle the outline mode of all subsequent GraphicBox commands.

The pattern for all future drawing operations is an index number from zero to 34. The outline mode can be set to zero to turn it off completely, or to 1 to activate the feature. If the setting is activated, bars will be automatically enclosed by a hollow box in the current outline colour.

To generate a hollow box in its own right, the next command is used.

GraphicSquare

Interface instruction: draw a hollow rectangle

```
GS x1 ,y1 ,x2,y2;
```

This is the equivalent to the normal BOX command, and it is used to draw a hollow rectangle, determined by the coordinates of the top left and bottom right-hand corners.

Lines and Outlines

SetLine

Interface instruction: set the style of a line

```
SL pattern;
```

This is identical to the SET LINE command, and is used to design the style of all future line drawing. The pattern for the line style is set by the combination of "dots and dashes" in binary format. For example, this example would set a line style of equal solid and blank components:

```
SLine %1111000011110000;
```

GraphicLine

Interface instruction: draw a line on the screen

```
GL x1,y1,x2,y2;
```

This command draws a graphical line from coordinates x1,y1 to x2,y2, like the normal DRAW command. As usual with Interface commands, the starting coordinates are measured from the BAse position. For example:

```
GLine 0,0,100,100
```

GraphicEllipse

Interface instruction: draw an ellipse or circle

```
GE x,y,radius1,radius2;
```

This simple Interface command is used to draw hollow ellipses or circles. The centre of the figure is set by coordinates x,y relative to the BAse location, and then the height and width of the figure are set by specifying the length of the appropriate radii in pixels. To draw a circle, simply specify the same value for both radii.

Displaying text

Once the surroundings of the requester have been generated, the following commands will be needed to display text in these dialogue boxes. The PRint command is examined in Chapter 9.1, here are some more instructions to affect
the way text is displayed.

PrintOutline

Interface instruction: print hollow text with outline

```
PO x,y,'text',outline colour,text colour;
```

The PrintOutline command is used 'to call up outlined or stencilled text. This is achieved by pasting the same text at slightly different positions, and is most effective with larger typefaces. The parameters are self-explanatory, and
consist of the starting coordinates for the text on screen, the string of text enclosed in single quotation marks and the index numbers of the outline and text colours.

One side effect of the PrintOutline command is that the drawing mode is automatically re-set to transparent, and so a previous opaque setting may need to be changed. This is explained next.

SetWriting

Interface instruction: set the writing mode for text and graphics

```
SW mode;
```

Normally, all text and graphics are drawn over a transparent background, allowing them to merge neatly into the existing display. If you need to set the background colour using an INk command, you must change the mode to "opaque". The mode parameter that affects this command uses a value from zero to 7, and is fully explained under the GR WRITING command in Chapter 6.4. A parameter value of zero will set an opaque mode, and a value of 1 resets it to transparent. Use of the PrintOutline command automatically resets the mode to transparent.

SetFont

Interface instruction: select font to be assigned to text

```
SF number,style;
```

To change the type font used by a previous PRint or PrintOutline command, use SetFont followed by the number of the new font to be assigned to the text, and the style to be adopted by that font. A full explanation of the available styles can be found under the SET TEXT command in Chapter XX.X (5.6).

If only the style is to be changed, the font remains unaffected when a dummy parameter value, such as zero, is used for the font number.

There are four more simple Interface functions that can be used to manipulate text, which are self explanatory.

TextWidth

Interface function: return the width of current font text in pixels

```
width="text" TW
```

TextHeight

Interface function: return the height of current font in pixels

```
height=TH
```

TextLength

Interface function: return the number of characters in a string of text

```
number="text" TL
```

CentreX

Interface function: centre text in the display

```
position="text" CX
```

The CentreX function is used to find the correct centring location by comparing the width of the given text with the value returned by an SX function, like this:

```
SetVar 0,"Hello, this is the text"
PRint 0VA CentreX,0,0 VA,2;
```

VertText

Interface instruction: display vertical text

```
VT x,y,'text',colour
```

The VertText command is used to display a column of vertical text using the specified colour index number, starting at your chosen coordinates.

Labels and Tests

The AOZ Interface also supports a number of program control instructions, allowing simple tests to be performed and jumps to be made to various routines. These controls make it easy to create complex multi-level user interfaces.

LAbel

Interface instruction: create a simple label

```
LA label;
```

The LAbel command is used to define a marker label in an Interface program. This can them be employed as the target destination for a JumP or JumpSub command, which are the Interface's equivalents to the familiar GOTO and GOSUB operations in normal AOZ programs. It can also be used as an entry point for the DIALOG RUN instruction.

Several complete dialogue boxes can be installed into the same definition string, if required.

Unlike conventional AOZ programs which recognise any characters for labels, Interface labels are referred to by numbers only, ranging from zero up to 65535. If a newly defined label already exists, an error message will be generated. Interface labels are defined in the following manner:

```
LA 10; set up label number 10
```

JmP

Interface instruction: jump to an Interface program label

```
JP label;
```

The JumP command transfers control to the Interface instructions that commence with the selected label number.

This label must be defined elsewhere in the Interface program. The JumP instruction cannot be used inside any of the routines held in square brackets of a BUtton command. It is used with the single parameter of the target label,
like this:

```
JumP 10;
```

Interface labels are also used to mark the start of various subroutines, which are entered and left as explained next.

JumpSubroutine

Interface instruction: call an Interface sub-routine

```
JS label;
```

The JumpSubroutine command calls up the sub-routine whose beginning is marked by the specified label number from zero to 65535. Sub-routines may be nested inside one another, with a maximum of 128 calls that can be made from each routine.

ReTurn

Interface instruction: return from an Interface sub-routine

```
RT;
```

An Interface sub-routine must be terminated by a ReTurn command. The Interface program will now re-commence from the command immediately after the initial JumpSubroutine call. If a ReTurn call is encountered out of sequence, an error will be generated.

Interface conditional tests

Constructing a test facility inside the AOZ Interface is relatively simple.

IF

Interface structure: Mark start of conditional test

```
IF expression;[routine]
```

The familiar IF structure is followed by an Interface expression. When the expression results in a value of zero (False), any routine held inside the square brackets will be completely ignored, but if the expression is not zero (True) the bracketed routine will be executed immediately.

The expression is a normal Interface expression, and all values are taken from the stack in reverse order! The routine within the square brackets contains a list of normal Interface commands to be performed if the expression is true.

There is no limit to the size or the number of these commands, and JUmp as well as JumpSubroutine calls can be included. User-defined instructions may be accessed, and these are explained below. It is even possible to include another IF within the square brackets, and providing the number of opening brackets equals the number of closing brackets, all should be well. Here is a very simple example of a conditional test:

```
IF 0VA 1=; if the contents of variable zero is equal to one
[PRint 0,0,'Variable 0 equals 1',5; then print a message]
```

Here is a table of the available testing operators that can be used for Interface conditional tests:

Operator Meaning Notes

= equals Gives -1 if two values are equal, otherwise gives zero
\ not equals Gives -1 if two values are unequal. Do not confuse with /
< less than Gives -1 if the first value is less than the second value

> greater than Gives -1 if the first value is greater than the second value
> & logical AND
> | logical OR

User-defined functions

User-defined Interface commands are treated in exactly the same way as any of the existing instructions, and they can be used to create whole libraries of box definitions, button types and selectors.

UserInstruction

Interface instruction: create a user-defined Interface command

```
UI XX,number of parameters;[instruction definition]
```

To create a new Interface instruction, use the UI command and then specify the pair of capital letters that are to represent the name of the new instruction from now on. For example, SB could be specified to refer to a new command for drawing a ShadowedBox. The new name must not already be used for an existing Interface command.

The pair of identification letters is followed by a list of parameter values. Each user-defined instruction can read up to nine parameter values from the Interface program, these parameters must be separated by commas, and can be entered directly in the command line. They can now be read from the new definition routine using the parameter functions P1 to P9.

Finally, the definition of the user-defined command is specified inside a pair of square brackets. This definition enters an Interface program to be assigned to the new instruction, and it can include anything you wish.

Here is a complete working example of a user-defined instruction that draws a shadowed dialogue box:

TODO - "Update code underneath to say AOZ instead of AMOS"

```
A$=A$+"BA 50,50;"
A$=A$+"SI 160,60; SA 1;"
A$=A$+"SB 0,0,150,50,5,0,5;"
AS=A$+"PO 10,10,'AMOS Professional',2,4; PR 48,20,'Basic',4;"
A$=A$+"BU 1,90,38,50,10,0,0,6;"
A$=A$+"[ShadowBox 0,0,50,10,1,0,4 BPos+; PR 1,2,'Button',6;][]"
A$=A$+"RU 0,7;EXit;"
A$=A$+"UserInstruction ShadowBox,7; create shadowed box with seven parameters"
A$=A$+"[IN P6,0,0; GB P1 P5+,P2 P5+,P3 P5+,P4 P5+; draw the shadow effect"
A$=A$+"1N P7,0,0; GB P1,P2,P3,P4, draw the box at the top]"
D=Dialog Box(A$)
```

The Interface provides a method of arranging your user-defined instructions so that they behave exactly like the built-in graphics operations.

XY

Interface instruction: set graphics variables

```
XY xa,ya,xb,yb
```

The XY instruction loads the internal variables XA, YA, XB and YB with the position of the graphics cursor before and after the new operation.

The xa,ya parameters hold the values to be loaded into XA and YA, and these store the coordinates of the graphics cursor before the current operation is performed. Similarly, the xb,yb parameters enter values to be stored in XB and YB, and they are used to set the position of the cursor after the new instruction as been executed. So after the new command has done its work, the internal variables can b set up accordingly. For example:

```
'An instruction that draws a box and then prints some text in it.
'XB YB will be at the end of the box, and not at the printed text!
'The syntax for this is TextBox x1 ,y1,x2,y2,text
Userinstruction TextBox,5;
[BOx P1,P2,1,P3,P4; PRint P1 ,P2,P5,1; XY P1 ,P2,P3,P4;]
```

User instructions have a few limitations, which should be remembered. Each command is restricted to a maximum of nine parameters, identified as P1 to P9, and any user-defined instruction can call up to a maximum of ten additional UserInstruction commands from the definition routine.

One additional problem can occur if you want to create a new button type with a UserInstruction. You need to save the parameters somewhere safe before you leave your routine, so that they will be readily available when the new button is selected on the screen. This problem only occurs if you are defining a zone inside a UserInstruction, and the problem can be solved by the following instruction.

SetZonevariable

Interface instruction: save a parameter for the next zone definition

```
SZ value;
```

This command is used to set the internal zone variable for the next active zone to be defined. The value can be a number or a string that you want to save, and the Interface stores it in an internal buffer, ready for the new Zone definition. You can now use the ZoneVariable function to poke this value directly into the new zone, so that a "change" routine, held inside the second pair of square brackets, will work as normal. This function is explained lastly..

ZoneVariable Interface function: read a zone variable from the internal buffer area value=ZV This function can only be used inside a change or a draw routine held inside square brackets. It returns the contents of the internal zone variable, and pokes it into your definition for safe- keeping. For example:

```
'This user instruction defines a button made of text only
'MyButton zone,x,y,text
Userinstruction MyButton,4;
[SetZone P4; save parameter four in the internal buffer zone
BUtton P1,P2,P3,P4 TextWidth 16+,TextHeight 4+,0,0,1;
[PRint 0,0,ZoneVariable,1][] zvar permanently installs text into the new button
```

Machine code extensions

The final part of this chapter explains how to add your own machine code extensions directly into the AOZ Interface.

CAll

Interface instruction: call a machine code extension

```
CA address;
```

This instruction has no effect in AOZ.