TODO- "this chapter was skipped, to convert?'

Advanced Control Panels

This chapter reveals the true power of the AOZ Interface, and deals with the creation of advanced control panels, dialogue channels, edit zones and sliders.

The last chapter explained h w simple requesters and dialogue boxes that wait for the user to make a selection, and then return straight back to the main AOZ program with the result. But this only uses a fraction of the system. In actual fact the Interface is capable of running a dialogue box completely in the background, exactly like an AOZ menu.

A control panel can be displayed on screen permanently, and each button selection can be read as it happens, directly from the main program, without any interruption to that program whatsoever. This means that a computer game can be busy performing its calculations while the user enters new values into a dialogue box.

In order to access these features, a little preparation must be made in advance, in order to exploit dialogue channels.

Dialogue channels

DIALOG OPEN

instruction: open a channel to an Interface program

```
Dialog Open channel number,Interface string
Dialog Open channel number,Interface string,nvar,buffer
```

The DIALOG OPEN command opens a "communication channel" to the new program, and loads it with a list of Interface commands. If there are any problems, an appropriate error message will appear, and mistakes can be located using a simple call to the error function EDIALOG, which is explained below.

The parameters for a DIALOG OPEN instruction are given as follows: Firstly, the number of the channel to be opened, starting from 1. Providing that there is enough memory, you may open as many channels as you wish. A string should be specified next, containing one or more Interface programs to be initialised. If this string contains several programs, each routine should begin with a LAbel instruction and end with an EXit command.

Normally AOZ provides enough space to hold up to 17 different values in every Interface channel (0 VA to 16 VA). If more channels are needed, this array can be expanded via an optional nvar parameter, and each extra variable will take up four bytes of memory. There is a final optional parameter that allocates bytes for an internal memory buffer used by Interface programs. This array holds all of the information that is required to display dialogue boxes and selectors on screen. As a default, 1k is reserved for each channel that has been defined, but if the requirements are very complex, this value may have to increase. An error message will appear automatically if the current allocation is too small.

Note that the DIALOG OPEN command only initialises the communication channel, and it does not start the program running or generate any graphics on the screen. To accomplish this, the following function is used.

DIALOG RUN

function: run a dialogue box from an open channel

```
button=DIALOG RUN(channel number)
button=DIALOG RUN(channel number,label,x,y)
```

As its name suggests, DIALOG RUN executes an Interface dialogue program from a specified channel, previously opened by a call to DIALOG OPEN. This Interface program will now run in the background, leaving the main AOZ program to continue from its next instruction, providing that the Interface program does not contain a RunUntil instruction.

The use of an Interface RunUntil command will force the Interface program to behave exactly like the original DIALOG BOX routine, so it will take complete control of the system and only return when the user clicks on an exit button, or aborts by pressing [Ctrl]+[C]. In this case, the value held by "button" will contain the number of the last button pressed by the user.
The parameters for a DIALOG RUN function are very simple. First give the number of a channel that is currently open, then define the label in an Interface command string from which the program is to start. If a label is not specified, the Interface program will commence from the first routine in the list. Optional x,y coordinates can also be set to position the control panel on screen, and all graphics coordinates will now be measured from this location. It is important to note that if a BAse command is included in the program, these new x,y coordinates will be completely ignored! Also note that optional coordinates can be given without specifying the label parameter, as long as the appropriate commas are included. For example:

```
Dialog Run(1,,x,y)
```

Here is a simple working example:

```
A$=A$+"BA 50,50;SI 180,60;SA 1 ;IN 5,0,0;GB 0,0,170,50;"
A$=A$+"PO 10,10,'Simple Keyboard',2,4;"
Rem Define two quick return buttons
A$=A$+"BU 1,10,28,24,10,0,0,1;[IN 0,BP 2*,0;SW 1;PR 4,2,' 1 ',4;][BR 0;NW;]"
A$=A$+"BU 2,70,28,24,10,0,0,1;[IN 0,BP 2*,0;SW 1;PR 4,2,' 2 ',4;][BR 0;NW;]"
A$=A$+"EXit;"
Dialog Open 1,A$ : Rem Open a channel to the Interface program in A$
R=Dialog Run(1) : Rem Run the program
Rem Read each keypress as it is made
Repeat
    P=Dialog(1) : Rem Check for button selection explained later
    If P<>0 Then Play P*10,10
Until Inkey$<>""
Dialog Close
```

DIALOG CLOSE

instruction: close one or more dialogue channels

```
Dialog Close
Dialog Close channel number
```

This command closes one or more dialogue channels on the screen.

Interface programs are terminated and removed from memory immediately. If the Interface programs contain an SA command, the original background areas will be neatly pasted back onto the display.

Any active channel can be shut down with this command, by specifying a channel number. If the channel number is omitted, all of the current Interface channels will be de-activated. Please see the DIALOG FREEZE Command below, for something a little less drastic!

EDIALOG

function: find an error in an Interface program

```
position=Edialog
```

Whenever an error occurs in an Interface program, its position can be found with a quick call to the EDIALOG function. The relevant section of the Interface string will be displayed on screen, enabling you to discover what has gone wrong. In practice, the most common mistakes are caused by missing or wrongly-used semi-colon characters!

Here is a small error handler that may be useful if included in your own programs:

```
On ErrOr Goto TRAP: Rem Add this line before a DIALOG OPEN command
...: Rem The rest of your program goes here
TRAP: Print Mid$(DB$,Edialog,80) : Wait Key : End : Rem Error handler
```

Testing an active zone

DIALOG

function: return status of an open dialogue box

```
button=Dialog(channel number)
```

This function provides a simple method of testing whether or not an option from a control panel has been selected.

Simply specify the number of the open channel that is to be tested.

After this function has been performed, one of the following values for the tested button will be returned:

- <0 A negative value means that the current channel is inactive, either because it has not been
  assigned to a dialogue box, or because the user has left the box via an exit button.
- =0 A value of zero indicates that there has been no user-input since the previous test.

- > 0 If a positive value is returned, it indicates the number of the last button that was selected by the user. In the case of edit zones, a value will only be returned when the [Return] key is pressed.

Once the value of the contents held by the DIALOG function has been checked, it will be re-set to zero.

Here is an example of a simple check:

```
Do
    D=Dialog(1)
    Exit If D<0
    If D>0
        On D Gosub BUTTON1,BUTTON2
        Wait Vbl
    Endif
Loop
```

RDIALOG

function: read the status of a zone or button

```
button=RDIALOG(channel number,button number)
button=RDIALOG(channel number,button number, object number)
```

This function is used to read the position of a particular button or selector. Specify the number of an open Interface channel, and then give the number of the button or zone to be tested. There is also an optional parameter which can be specified to check one of several objects that have been assigned to a current zone number. If this object number is omitted, then the status of the first object defined in the current zone number will be returned. Object numbers are arranged according to the order in which they have been defined in the Interface program, so the first button has an object number of zero, the second will be read as 1, and so on.

The result returned by the RDIALOG function depends on the type of zone being scrutinised. In the case of a simple button, its current value will be given, but there are special numerical edit zones (explained later) which will return a new value entered by the user. If a text zone is checked in this way, a result of zero will given, and the RDIALOG$ function should be used instead. This is explained next.

RDIALOG$

function: return text string entered into an edit zone

```
text string=RDIALOG$(channel number,zone number)
text string=RDIALOG$(channel number,zone number,object number)
```

Use this function to return a string of text Assigned to a zone. If the selected zone does not contain any text, an empty string will be presented. Please see the Interface EDit command for more details.

Accessing a variable array

Variables that have been assigned to an active Interface channel can be read and modified directly from the main AOZ program.

VDIALOG

function: assign or read an Interface string

```
Vdialog(channel number,variable number)=value
value=Vdialog(channel number,variable number)
```

This function can be used to either read or change the variables in any active Interface program. The active channel number is selected, followed by the number of the variable you are interested in. The value refers to the new integer value that has been selected for this variable.

VDIALOG$

function: assign or read an Interface value

```
Vdialog$(channel number,variable number)=string$
string$=Vdialog$(channel number,variable number)
```

The VDIALOG$ function is exactly the same as the previous function, except that it works with strings rather than numbers.

Specify the channel number and the variable number, and then string$ holds a new AOZ string to be stored in the Interface variable array.

There now follows a detailed examination of some of the additional zone commands which make the AOZ Interface so special!

Editing zones

The AOZ Interface allows for both text and numbers to be input into screen zones.

EDit

Interface instruction: create a text edit zone

```
ED zone number,x,y,width,max,'string',paper,pen;
```

The EDit instruction opens a zone for text input on the screen. This zone can be selected with the mouse, and then a string of characters may be typed in using all the normal line-editing functions. The text cursor may also be positioned directly, by clicking the mouse pointer on the required new position.

The EDit parameters begin with the zone number, from 1 upwards. The x,y coordinates set the position of the text input line relative to the coordinate base, and will be added to the BAse setting to give the actual position of the line.

If the resulting x-coordinate is not an exact multiple of 16, it will be rounded down to the nearest 16 pixels.

The width parameter sets the width of the text edit window, and is specified in the number of characters to be accommodated, rounded up to the nearest even number. Next, the maximum length of the text string must be given in characters, and an area of this size will be reserved in the channel buffer.

The 'string' parameter enters a string of characters that will be initially loaded into the text edit zone. If a default string is not required, simply use a pair of single quotation marks with nothing in them, or use a zero. Finally, set the colour index numbers for the paper and pen to be used for the characters on screen.

If the size of the string is larger that the physical size of the zone, the string will scroll automatically as more characters are typed in. If the [Return] or [Tab] key is pressed within the edit line, the Interface will jump to the next EDit zone, if it has been defined. When the final zone is reached, all keyboard input will be directed to the dialogue buttons, and the next [Tab] press will return you back to the original editing zone.

When the dialogue box is drawn, the first edit zone of the Interface program will be activated automatically. Please note that order of activity follows the order in which the edit zones were declared in the Interface program, and not the number of the zones.
The contents of an edit zone can be read directly from the main AOZ program, using a simple call to a RDIALOG$ function. Also, if the [Return] key has been pressed, the number of the selected EDit zone will also be available from the DIALOG function.

Characters are entered in a new text zone window with the number z+1000. If the program is to display some text while the zone is in use, the default screen window should be re-activated with a WINDOW 0 instruction. After the text has been displayed, the edit window can be handled by a line like this:

```
Window Z+1000: Rem Z is the number of the EDit zone
```

Here is a working example of a text input zone:

```
A$=A$+"BA 50,50;SI 200,60; SA 1; set things up"
A$=A$+"IN 5,0,0; GB 0,0,200,50; PR 16,10,'Enter your name human!',2;"
A$=A$+"BU 1,150,40,50,10,0,0,16;"
A$=A$+"[1N BPos 4+,0,0; GB 0,0,50,10; PR 2,0,' Quit',2;][BQ]"
A$=A$+"ED 2,16,25,14,14,",0,2; define an edit zone"
A$=A$+"EXit,"
Dialog Open 1,A$ : Rem Open a channel
R=Dialog Run(1) : Rem the Interface program
Repeat
    D=Dialog(1) : Rem Wait until Return Key is pressed
Until D<0
Print "Hello ";Rdialog$(1,2)
```

Dialog Close

The numerical equivalent of a text zone is examined next.

DIgit

Interface instruction: create a numeric editing zone

```
DI zone number,x,y,width,value,flag,paper,pen;
```

The DIgit instruction creates a special edit window for entering numbers. It is very similar to the previous EDit command, except for the fact that only the digits from zero to 9 may be entered, and anything else will be completely ignored by the Interface system.

Please refer to the EDit command to see how the zone number is specified first, followed by the x,y-coordinates that set the position of the zone on screen. The width of the window is then set, and it is rounded up to the next even number, in exactly the same way as with an EDit instruction.

The value parameter defines what the initial default value displayed in the edit window is to be. The flag parameter indicates whether the default is to be shown in the window, with a zero setting to make the default value invisible, and any other setting meaning that it will be displayed as normal. Finally, the paper and pen parameters are set by the colour index numbers to use for the input numeric characters.

The DIgit value can be read from the main AOZ program with the RDIALOG function.

Here is a simple working example:

```
A$=A$+"BA 50,50; SI 200,90; SA 1;"
A$=A$+"IN 5,0,0; GB 0,0,200,80;PR 16,10,'Enter your name human!',2;"
A$=A$+"BU 1;150,70,40,10,0,0,16;"
A$=A$+"[IN BPos 4+,0,0; GB 0,0,50,10; PR 2,0,' Quit',2;][BQ;] KY 13,0;"
A$=A$+"ED 2,16,25,14,14,",0,2; text edit zone"
A$=A$+"PR 16,40,'How old are you?',2;"
A$=A$+"DIgit 3,16,50,4,30,0,0,2; numerical edit zone"
A$=A$+"EXit;"
Dialog Open 1,A$ : R=Dialog Run(1)
Repeat
    D=Dialog(1) : Rem Wait for the user to enter the last item
Until D<0
Print "Hello ";Rdialog$(1,2) : Print "You are ";Rdialog(1,3); "years old"
Dialog Close
```

Sliders and Selectors

One of the most powerful features of the Interface system is the facility to create a wide variety of sliders and selection boxes in AOZ programs. The next part of this Chapter provides a detailed examination of the commands that make this possible.

HorizontalSlider

Interface instruction: create an animated horizontal slider bar

```
HS zone number,x,y,width,height,position,trigger,total,step;[changes]
```

The HS command draws an animated horizontal slider bar on the screen.

It is similar to the AOZ HSLIDER command, with the additional facility of allowing the slider's position to be determined directly using the mouse. All of the animation is handled by the Interface automatically.

Here is an explanation of the HS parameters, in the order that they are set.

The zone number is simply the number of the new zone that is to be defined. Next the width and the height of the slider is set, and since this is a horizontal bar being created, it is sensible for the width to be greater than the height!

The position parameter is a simple number, ranging from zero to "total", and it determines the default position of the slider's trigger, which is the active part of the horizontal bar, somewhere in the middle, that is dragged to the left or right by the mouse.

Total is the parameter which defines the maximum value that will be returned by the slider.

Allowable positions range from 1 to "total", with each step representing a movement of the trigger, in pixels.

The step parameter controls the distance that is to be moved whenever the user clicks on the background area of the slider. The bar will scroll slowly towards the current mouse position in units specified by the number of pixels given in the "total" parameter above. Once the actual pointer has been reached, it will cycle back and forth by a single step.

Finally, a list of Interface commands is given within a set of square of brackets. This [changes] parameter will be called up whenever the slider is moved on the screen.

After a selector has been activated, its position can be read from the main AOZ program using the RDIALOG function, as explained earlier in this Chapter. The new position will only be reported to the main program after the user releases the left mouse button.

The colours used by the slider bar are zero for the background area, 4 for the unselected bar and 3 for the selected bar, which is normally flashing. Here is a working example of a horizontal slider in action:

```
B$=B$+"BA 112,50; set base coordinates to the screen centre"
B$=B$+"HSlide 1,0,0,100,16,0,1,100,1;[] define a one hundred position slider"
B$=B$+"EXit;"
Dialog Open 1,B$ : Rem Open a dialogue channel to the slider
D=Dialog Run(1) : Rem Run the program held in B$
Curs Off : Centre ""
Rem Read the slider
Repeat
    D=Dialog(1) : Rem See if slider has been selected
    If D<>0 Then Locate 14,20 : Print "Position ";Rdialog(1,1);" ";
Until Inkey$<>""
Dialog Close
```

VerticalSlider

Interface instruction: create an animated vertical slider bar

```
VS zone number,x,y,width,height,position,trigger,total,step;[changes]
```

The VerticalSlider command generates a working vertical slider, as demonstrated by the standard AOZ file selector. The parameters are exactly the same as for the HSlider command, and the position of the slider can be read by a call to the RDIALOG function, like this:

```
position=RDIALOG(channel number,zone number)
```

Once slider bars have been created, they can be used to produce attractive selector boxes. These allow you to scroll through a list of items directly on the screen, and select them individually using the mouse, again, as demonstrated by the AOZ file selector.

Reading arrays

In order to generate an Interface selector, a method is needed of grabbing an entire list of items from the AOZ main program. A simple function is provided for transferring complete arrays into an Interface routine.

ARRAY

function: load the address of an array into a program

```
address=ARRAY(list$(0))
```

The ARRAY function returns the address in memory of the first item in the specified list$ array. This string can contain any data at all, but if the array is to be accessed from an Interface program, each element in the array must be of exactly the same length.

After it has been returned, the address may now be installed into an Interface program using the VDIALOG function, as follows:

```
Vdialog(1,0)=Array(V$(0)) : Rem Loads 0 VA (channel 1) with the address of V$(0)
```

Once installed like that, the address can be accessed from an Interface program using the ArrayRead function.

ArrayRead

Interface function: read an element from an AOZ array

```
item=address,element number AR
```

This function returns the specified item from a normal AOZ array. The element to be returned must have been previously loaded into an Interface variable from the main program. The two parameters that must be specified are the address which holds the location of the first item in the array, followed by the number of the item in that array which is to be returned. Obviously, if the number of the specified element is higher than the dimension of the array, an error will be generated.

ArraySize

Interface function: return the size of an array

```
size=address AS
```

The ArraySize function is used to return the number of elements in an AOZ array, stored at the specified address. The address is the location of the array in memory which has been loaded from the ARRAY function.

Displaying items on the screen

After an item array has been entered into the Interface, it can be displayed on the screen using the powerful ActiveList command.

ActiveList

Interface instruction: display an active list window

```
AL zone number,x,y,width,height,address,index,flag,paper,pen;[changes]
```

This command displays an active window for an AOZ string array. Each string can be individually selected with the mouse, and returned to the main program by the RDIALOG$ function. An ActiveList command can be linked to a set of slider bars or an edit zone, so that the bars move the list up and down through the array, and the edit zone changes the value that has been selected on screen.

The parameters for this instruction must be given in the following order:

- The zone number to be allocated to the selector, followed by the x,y-coordinates to set the position of the selection box on screen. Because a normal window is used for this display, the location of the x-coordinate will be rounded down to the nearest multiple of 16. If the coordinate base has been set to a new value with a BAse instruction, this will be added to the x-coordinate before it is rounded down, so the final screen coordinate will always be an exact
  multiple of 16.
- The width and height of the window are specified next, in the number of characters required.
  These parameters are followed by the address of the string array to be displayed in the selector box, and this array must have been previously defined in the program. The address of the array can now be grabbed with the ARRAY function from the AOZ main program, and loaded into an appropriate Interface variable with VDIALOG. It can then be entered directly into the AList command. This system is explained in detail below.
- The next parameter is the index number of the first item to be displayed in the selector box. If this number goes past the end of the array, the selector will be filled with blank lines.
- The flag parameter is a bit-map that is used to trigger a range of useful features, as follows:
  - Bit 0 If this is set to one, each string will be preceded by a number representing its position in the array. The count normally starts from zero, but this can be changed as explained next.
  - Bit 1 is only active if the automatic numbering system has been turned on by setting Bit 0 to one. If this is the case, then the number count will start from one rather than zero, if Bit 1 is setto one.
  - Bit 2 changes the way the selector reacts to the mouse pointer. If it is set to zero, each line will react to the mouse immediately, with no need for a click of the mouse button. If Bit 2 is set to one, then items must be selected explicitly with the left mouse button.
- The paper and pen parameters are the colour index numbers for the text in the item.
- Finally, the square brackets hold an Interface routine that determines any changes to be executed every time one of the items is selected.

Note that the text is displayed using a normal AOZ window defined by the number z+2000. Particular care should be taken if you are writing to the screen while reading a selector. Return to the normal screen before printing, with a WINDOW 0 command, then use a line like WINDOW Z+2000 to move the cursor back to the active window. If this is not done, text will be printed inside the selection area!

As with all of these techniques, there is a ready-made working example of an active window, available for examination:
TODO - "Change code below to AOZ"

```
Load "AMOSPro_Tutorials:Tutorial/Interface/Sliders.AMOS"
```

InactiveList

Interface instruction: display an inactive list window

```
IL zone number,x,y,width,height,address,index,flag,paper,pen;
```

To display a window containing items in an array that are not for selection by the mouse, the InactiveList command is used in much the same way as an ActiveList instruction except that there is no [changes] parameter. Inactive windows are useful for generating simple lists on the screen.

Before using the AList or IList commands, some advance preparation should be carried out, using the following steps:

First of all, define a string array to hold the items in memory, using a line like this:

```
Dim ITEM$(100)
```

Next, load the items into the array. These items may be anything you wish, such as the commands for an adventure game, or a set of filenames on a disc. The one factor that must be ensured is that each item must have exactly the same number of characters. Use spaces to pad items as necessary.

The third step is to enter the Interface program into a string, and include an ActiveList command. The following example line would create zone 1 at coordinates 10,10, with 15 lines of 30 characters each, taking the address of the item array from 0 VA.

```
AL 1,10,10,30,15,0 VA,0,0,0,1;[]
```

Now open an Interface communication channel with a DIALOG OPEN command, and grab the address of the first character in the item array to be loaded into an Interface variable.

```
Dialog Open A$,1
AD=Array(ITEM(0)) : Vdialog(1,0)=AD
```

Finally, call the Interface program with DIALOG RUN, like this:

```
R=Dialog Run(1)
```

Creating a selector

The ActiveList command is very useful, but it is not intended to operate in isolation. To create a working selection box, an appropriate set of sliders, buttons and edit zones need to be linked up, as explained in the next part of this Chapter.

ZonePosition

Interface function: return the status of a zone

```
value=ZP
```

ZonePosition returns the value of the current zone's status. In fact it is simply an expanded version of the BP function, with the difference being that it can be used in any active zone, including buttons.

ZoneChange

Interface instruction: change the status of a zone

```
ZC zone number,new data;
```

ZoneChange is a simple expansion of the BC instruction. It is used within the "changes" square brackets to link a number of zones together into one compound object. After the ZC command, specify the number of the zone to be affected, followed by some new data that is to be fed into the zone.

The new data depends on the type of zone that is to be affected, and there are four ossibilities:

- Buttons. The data holds a new value for the status of the button position. If this is different from the current position, the button will be re-drawn immediately.
- Edit zones. If a text zone has been created with EDit, the new data should contain a new string of characters that is to be displayed in the box. Similarly, if the zone was defined using Digit, the existing number will be replaced by a newly specified value.
- Sliders. The data is used to move the slider bar on the screen, exactly as if it had been selected by the user.
- Active lists. In this case, the selection window is scrolled up or down the string array.

The ZoneChange command can be used to great effect with the ZonePosition function, to link the positions of various items in the dialogue box. For example, if a slider bar has been assigned to zone 1, and an active list to zone 2, you would be able to scroll through the list with the slider using a line like this:

```
VSlide 1,16,16,8,64,0,1,12,1; create a twelve position slider
[ZChange 2,ZPosition;] set zone two to position of changed slider
```

The commands inside the square brackets are executed automatically whenever the slider is moved on the screen, and they copy the new slider position straight into the active list command and move the list through the selector window.

Here is a step by step procedure for generating a selection box on screen:

- Firstly, create a list of components needed for the selector. This could begin with a vertical scroll bar, where 0 VA holds the address of the item array already set up in the main program, and AS returning the number of items:

```
VSlider 1,x,y,width,height,0,8,0 VA AS,1;[]
```

Next an ActiveList is defined for the selection window, using a line like this:

```
AList 2,x+width,y,20,8,0 VA,0,0,0,4; set up the item list
```

Finally, a text edit zone is created to hold the current item on screen, like this;

```
EDit 3,x,y+height,width/8,width/8,0,0,2;
```

Now the scroll bar can be linked to the item display in the ActiveList window, as already explained.

```
VSlider 1,x,y,width,height,0,1,0 VA AS,1; create a twelve position slider
[ZChange 2,ZPosition;] set zone two to slider position
```

Lastly, the ActiveList is linked with the EDit command, so that it loads the selected item into the editing zone, as follows:

```
AList 2,x+width,y,20,8,0 VA,0,0,0,4; set up the item list
[ZChange 3,0 VA ZPosition ARray;] load the edit zone with selected item
```

Remember that 0 VA stores the address of the array, ZPosition holds the item number and ARray is used to get the item. This will return the value of the selected item straight to the EDit zone. If you need to transfer the edited version back to the selector, the appropriate array item needs to be loaded directly from the main AOZ program. Please see the DIALOG UPDATE command for a detailed explanation.

It is sometimes necessary to link up a number of zones in sequence, and relative values instead of absolute zone numbers can be used for this purpose.

ZoneNumber

Interface function: return the number of a zone

```
number=ZN
```

The ZoneNumber function is used to return the number of the current zone. It is intended to be used in conjunction with the ZoneChange command, like this:

```
[ZChange ZNumber 1+ZPosition] loads the next zone with existing position value.
```

Controlling a selector from the main program

DIALOG UPDATE

instruction: update a zone

```
Dialog Update channel number,zone number[,param1][,param2][,param3]
```

This instruction enables AOZ programs to force the Interface to re-draw a zone on the screen. It is especially useful for selectors, which often need to interact directly with the main program. These can include file selectors that need to read a new search path and update the file list, as well as EDit routines that must load a selection window with a new value entered by the user.

The DIALOG UPDATE parameters are given in the following order: first the channel number of an active dialogue channel to be updated. This is followed by the number of the zone to be affected. There are also three parameters held inside their own set of square brackets, and the effect of these parameters varies, depending on the type of the zone. Parameter I affects any of the different zone types, whereas Parameters 2 and 3 affect active lists and sliders only. Here is a table of the possibilities:

Parameter 1

```
- Button Enters a new status position
- Active List Sets the number of the first string displayed
- Slider Moves the slider
- Digit Replaces the existing number zone
- Edit Inserts a new string into the edit zone
```

Parameter 2

```
- Active List Changes the currently selected string
- Slider Changes the size of the slider window
```

Parameter 3

```
- Active List Chooses the last element of the array that can be selected.
Normally this parameter is equal to the size of the array, but it can be
restricted for certain applications.
- Slider Changes the "total" parameter
```

DIALOG FREEZE

instruction: stop dialogue channel input

```
Dialog Freeze [channel number]
```

This command is used to freeze all input from one or more active dialogue channels. The number of a single dialogue routine that is to be suspended is specified in square brackets. If this number is omitted all current channels will be frozen.

DIALOG UNFREEZE

instruction: re-activate a frozen dialogue channel

```
Dialog Unfreeze [channel number]
```

Use this instruction to re-activate one or more dialogue channels from the point at which they were frozen.

DIALOG CLR

instruction: clear a dialogue box

```
Dialog Clr channel number
```

The DIALOG CLR instruction erases all zones and shuts down the dialogue box completely, leaving the specified channel open. The Interface program can now be re-run from the beginning, using a further call to the DIALOG RUN command. As always, if the background area has been saved using the SA option, it will be restored to its original position.

HyperText

The impressive interactive Help system is one of the most user-friendly aspects of the AOZ system, allowing detailed information to be called up using single mouse key- presses from the Editor. More impressive than this is the fact that the Help system was entirely written in AOZ, and it is a typical example of the Interface in action!

The final part of this chapter explains how similar systems can be created using the HyperText feature.

HyperText

Interface instruction: open an interactive text window

```
HT zone number,x,y,width,height,address,line number,buffer,paper,pen;[changes]
```

The HyperText command opens a simple window on a piece of text held in memory. This text can include optional words or phrases that may be selected directly by the user. Each option returns a specific value to the main program, which can then be used to jump to some additional text, or call up an appropriate routine from the main program.

The zone number is given as usual, followed by x,y-coordinates. The x-coordinate will be rounded down to the nearest multiple of 16.

Next the width and height parameters are specified to define the size of the text window that will hold the required characters. This window is very similar to a window produced by AList, and is relatively crude compared to the fancy Help screen display. However, when border images and sliders are added, things begin to take on a true AOZ appearance.
Text is listed in windows numbered 3000+z, and you can write to these windows from the main program if necessary, using a WINDOW command like this:

```
Window 3001 : Rem Print to text window assigned to zone 1
```

The next parameter to be given is the address of the text in memory. Unlike the AList command, HyperText expects an address of a Memory Bank rather than an array. This bank should already be loaded with some text in standard Ascii format, and each line should be separated byby chr$(13)+ chr$(10) for PC format. Text must be terminated with a single chr$(0) character.

The line number parameter holds the number of the first line in the text to be displayed in the window.

The buffer parameter sets the maximum number of buttons on any one screen line. The Interface requires 8 bytes for each zone, so if the window is 25 lines high, and ten button's are required on every line, you will need 25*10*8 or 2000 bytes for the buffer area. If you simply want to display some normal text, use a value of zero for the buffer parameter instead.

The paper and pen parameters are set as usual, to determine the colour of the text on screen, and the background window will be filled with the paper colour when it is initially drawn.

The square brackets hold an Interface function that will be called up whenever the mouse is clicked on the HyperText zone. If the zone returns a number, it will now be available directly from the ZPosition function.

Creating some HyperText

The text can include a number of active zones if required. These are defined using curled brackets, and their are two alternative formats, as follows:

```
{[value]highlighted text}
```

or

```
{[value,paper,pen]highlighted text}
```

The square brackets contain a value which will be transferred to the program when the item is selected. This can beeither a number or a part of the text, up to 64 characters long. If the value is a number, such as {[1000]...}, the mainprogram will load it into an RDIALOG function, and it can now be read directly from a [changes] routine using theZPosition function. If a string is entered, such as {[Hello everybody]...}, the main program will grab it with RDIALOG$ instead, and the ZPosition function will return a value of zero.

Supposing an automatic jump to a specific page of text is required. The HyperText command would look like this:

```
HText 1,0,0,40,20,text_address,0,4,0,3;
[ZChange 1,ZPosition]
```

The entry in the text would -he defined along the following lines:

```
X> {[10] Goto line 10}
{[100] Goto line 100}
```

After the return value has been set up, the paper and pen colours of the button can also be included. If these are not defined, the ink and paper colours will be swapped over when the active highlighted word is displayed on screen.

Finally, the word or sentence to be highlighted is given. This can be anything from a single character up to an entire line of text, and it will be displayed at the current cursor position automatically.

The zone definition is now closed with a closing curled bracket.

If an error should occur, or if there are too many button zones on the same line, the zone will be printed in simple Ascii, and the [1 characters will be visible on the screen.
To scroll the window with a slider bar, the number of lines in the current piece of text need to be known. This allows you to set the "total" value to the actual size of the text, and then scroll the text by a single line.

The HyperText instruction places this size in the internal ZoneVariable, so all that needs to be done is define the slider after the text window is opened. For example:

```
SetVar 1,0; variable number one holds the position in the text
HText 1,0,0,38,20,0 VA,1 VA,10,0,1;[]
SetVar 2,ZVar; load the number of lines into variable number two
VSlider 2,38 8*,0,8,20 8*,1 VA,40,2 VA,1; use this to set the total value
[SetVar 1,ZPos; ZChange 1,1VA:] move hypertext window to new position
```

If a HyperText window has been defined as a separate screen, it can be physically manipulated using a ScreenMove command.

ScreenMove

Interface instruction: move a screen linked to mouse pointer

```
SM;
```

This simple instruction is used to automatically drag the screen when the mouse pointer is moved, and it should be called as part of a [changes] routine to position the screen whenever a particular item is selected. In this final example, a button is set up that generates the scroll bar used by the Help window.

```
BUtton 6,24,0,SX,10,0,0,0;[][ScreenMove;]
```