TODO - "to be converted. not in use atm"

AOZ Interface

Section 9 of this User Manual is devoted to the AOZ Interface. You should be warned that this is a very advanced feature of the system and will take a while to fully understand, even for experts! Unfortunately there can be no short-cuts with such a technical subject, so please persevere.

Introducing the Interface

Imagine the possibilities of including all of the features of the AOZ Editor inside your own programs, and that they can be controlled directly from the screen via icons, buttons and selectors!

How about a state-of-the-art graphics adventure, with selection boxes for the commands and an interactive inventory on the screen. Or a set of direction keys that display all of the available movements from the current location. Simple simulations could be transformed into complex fantasy worlds, arcade games could have interactive hi-score tables, and animations could be conjured up by the actions and reactions of the player.

Perhaps you are interested in more serious applications. Then imagine an animated data base, complete with intelligent dialogue boxes, or powerful calculators that appear above your program listings on demand. Even the humble File Selector could be transformed beyond recognition!

The Interface is directly available from your AOZ, so anything the Editor can achieve, you can do too! What's more, you have instant access to all of the original Editor messages and graphics, allowing an effortless matching of your own programs to the existing AOZ style.

There is no need to rely on the default settings, and graphics can be designed from scratch, using 16, 32 or even 64 colours. A powerful Resource Creator is provided on the Accessory Disc, allowing images to be grabbed from any IFF picture, and immediately assigned to your icons, buttons and requesters.

And if that hasn't whetted your appetite, see all of this theory in practice now, by running the following ready-made example program:
TODO - "Update Address below to AOZ"

```
Load "AMOSPro_Tutorial:Tutorials/Interface/Full_Example.AMOS"
```

The need for the AOZ Interface

Experienced AMOS programmers will know that the features demonstrated in the above example program could also be created using standard AMOS screen zones, so why is a separate Interface language required?

In fact, the traditional system may well be useful, but it has many limitations.

The SET ZONE command creates a rectangular testing zone around any area of the screen, and this area can be identified if it comes under the mouse pointer, via the MOUSE ZONE function. This is a powerful system, as demonstrated by the Disc Manager and Object editor, but these are massive programs involving huge amounts of work, even for the most experienced of

programmers. If you need to generate similar features in your own programs, much time and effort would have to be
expended.

All Interface programs are entered into strings.

These strings are brought to life with the standard AOZ functions DIALOG BOX or DIALOG OPEN.

Each Interface instruction consists of a pair of capital letters. Similarly to AMAL, anything in lower case letters is completely ignored, allowing commands to be expanded and customised for readability and recognition. REM comments can be freely included in Interface listings, as long as no capital letters are used.

Each command is terminated by a semi-colon character. A colon can also be used for this purpose. Each command must be isolated from its neighbours in this way. Spaces are not enough.

Here is a simple example of an Interface command:

```
EXit;
```

EXit

Interface instruction: Leave Interface and return to AOZ
EX;

The EXit command must always be the last instruction in an Interface program. It is used to return to the more familiar environment of AOZ programs, by informing I let the system know that the last line of an Interface program has been reached. If the EXit command is omitted, a syntax error will be generated when the
routine is tested or run.

Variables and numbers

The AOZ Interface provides a standard way of performing calculations, and saving the results into variables.

All Interface programs have their own list of variables, stored in an internal array. These variables have exactly the same format as ordinary AOZ numbers.

Each variable is referred to by an index number, from zero upwards. However, instead of reading an array in the conventional way, like this:

```
VARIABLE(index number)
```

Interface variables must have the item number placed first, like this:

```
index number VARIABLE
```

VAriable

Interface Function: Return value held by an index

```
value=index number VA;
```

VA will return the value associated with the item number index, making it available for your Interface program. As a default, up to 17 variables can be used at one time, but this number can be increased via the DIALOG OPEN command, which is explained later.

Setting a variable

Set Variable

Interface Instruction: Set an Interface variable

```
SV index number,value;
```

Setting a variable for the AOZ Interface is very easy. Simply give the command, followed by the number of the variable to be changed and the value to be entered. Interface variables are not limited to numbers, and complete strings of characters may be assigned as necessary. For example:

```
SetVar 0,42: this loads the value forty two into item number zero.
SetVar 1,'The answer is'; this loads a string into item number one.
```

The use of quotation marks in Interface programs is very important. You are recommended to use single quotation marks as above, instead of the conventional double quotes. This will avoid any confusion when Interface commands are typed into an ordinary AOZ string.

Numbers may also be entered in Hexadecimal or Binary notation, if preferred. For example:

```
SetVar 2,$2A;
SetVar 3,%101010;
```

PRint

Interface Instruction: Print contents of a variable to screen

```
PR x,y,number,ink;
PR x,y,'text',ink;
```

The PRint command is used to print the contents of a variable. After the command, the target coordinates should be specified, followed by the variable number or the string of characters to be printed. The final parameter is a colour index number, which determines the ink colour to be used. The following example prints a message at coordinates 10,10 using colour 2. Note the use of single quotation marks, which is explained above.

```
PRint 10,10,'Message',2;
```

The next example would print the contents of 1 VA at the coordinates 0,100 in ink colour 2:

```
PRint 0,100,1 VA,2;
```

The hash character # can be used as a special function, which converts a number into a string. It is similar to the normal AOZ STR$ function. When using this, the following syntax must be observed:

```
PRint 0,110,1 VA #,2;
```

Expressions

AOZ Interface expressions have been carefully optimised for speed, which is why they appear very different from the standard system.

All calculations are performed in reverse, with operations and functions after the numbers. So a normal expression like 1+2 is entered as 12+ for an Interface program. Similarly:

```
6*9 becomes 69*
8/2 is 82/
6-3 is generated by 63-
```

These expressions can be entered as part of a normal Interface command. Supposing you want to add one to the variable zero (0 VA). This could be achieved by using a line such as:

```
SetVar 0,0 VA 1+;
```

To perform more complex calculation's some theoretical understanding is needed. Every time a value is entered into an expression, it is laced on the top of a list of numbers known as a stack. Supposing a stack is represented by this:

```
3210
```

The stack grows from the bottom upwards, so 3 sits on the top and the zero is sitting at the bottom. Whenever the Interface encounters a number or a string, it is pushed straight to the top of the current stack. But operators are treated differently. When an operator or a function is spotted, the Interface will execute it immediately, grabbing the required values directly from the stack. After the calculation has been performed, the result is replaced back onto the current stack.

This process continues until the Interface reaches the end of the expression.
Here is a theoretical calculation of one of these expressions, in this case how the result of 21 is evaluated from the expression 23+ 4*1+

```
23+4*1+=54*1+
54*1+=20 1+
20 1+=21
```

That expression is evaluated strictly from left to right. The 2 and the 3 are first placed onto the stack and are then added together by the plus sign, giving a result of 5. The 4 is now loaded onto the stack to be multiplied by the item immediately below it, which is the 5 that resulted from the last operation. This gives a value of 20, and this new value is loaded onto the stack to be added to the item below, generating a final value of 21.

It is important to realise that the stack is only retained during the current calculation. It has no permanent existence whatsoever! This means that after the expression has been calculated, exactly one value must be left in the stack,
otherwise a syntax error will be generated.

Here is a table of the available arithmetic operators:

Operator	Example	Result	Explanation
+	12 + 3	15	add two values together
-	23- -1	22	subtract second value from value

beneath in the stack * 23* 6 multiply two values from the top of the stack / 62/ 3 divide a value by the value above it in the stack NEg 2NEg -2 toggle value from positive to negative ! "H" "ello"! "Hello" Add two strings together #42# "42" convert a number into a string MIn 6 49 MIn 6 give minimum of two values MAx 6 9 MAx 9 give maximum of two values.

A separate set of logical operators that can be used to perform tests is explained in Chapter XX.X (9.2).

Resources

Each Interface program has access to a number of objects, known as resources. These resources contain a set of images to be used for background effects, as well as a list of messages for titles or interactive buttons. There is a Resource Editor program within the Accessory package, which can be installed into a permanent memory bank ready for instant use. As a default, the Interface grabs the internal resources assigned to the Editor, allowing a complete range of useful images to be employed in your own programs. Resources are examined in detail in Chapter XX.X (9.4).

Calling an AOZ Interface Program

DIALOG BOX

function: display dialogue box on screen

```
button=Dialog Box(Interface$)
button=Dialog Box(Interface$,value,parameter$,x,y)
```

To display a requester or dialogue box, the DIALOG BOX function is used to handle your Interface commands from the specified Interface string. This dialogue now waits for either an appropriate button to be selected, or until a specific period of time has elapsed. The system then returns to the AOZ main program, returning the value of the button. The dialogue box can be quit at any time by pressing [Ctrl]+[C]. If this is done, a value of zero will be returned.

The Interface$ parameter is a normal AOZ Professional string, containing the Interface program. This may be followed by various optional parameters:

The optional value parameter contains a value that is loaded straight into the internal variable array. It can then be accessed using the VA function, from the dialogue box, or requester.

Parameter$ holds an optional string parameter which will be forwarded to the Interface program. It will be saved as item 1 of the variable array (1 VA).

Finally, the optional coordinates x,y are given, to position the dialogue box on screen. These coordinates may be overridden by a BAse command inside the Interface program, and this is explained later. Here are some examples:

```
A$=A$+"SetVar 1,'The answer is', set variable one to a message"
A$=A$+"SetVar0,42; variable zero is loaded with forty two"
A$,A$+"PRint 0,100,1 VA,2; print the message"
A$=A$+"PRint 0,110,0 VA #,2; print the answer"
A$=A$+"EXit; leave the interface program"
Print Dialog Box(A$)
B$=B$+"PRint 0,0,1 VA,2; PRint 0,10,0 VA #,2; EXit;"
D=Dialog Box(B$,42,"The Answer")
D=Dialog Box(B$,42,"The Answer",100,100)
```

Please note that if the Interface program is to wait for user input, a RunUntil command must be included before the final EXit, otherwise the dialogue box will jump directly to the main AMOS Professional program after the last Interface program instruction. RunUntil is explained in detail below.

The DIALOG BOX facility is only intended for simple requesters. To control the dialogue directly from an AOZ program, the DIALOG OPEN and DIALOG RUN commands must be used instead. These are fully explained at the beginning of Chapter XX.X (9.3), which is devoted to advanced control panels.

Creating a simple requester

If a large requester with a lot of graphics is to be generated, it can be very difficult to keep track of all of the coordinates. You can simplify things enormously by entering all of the coordinates relative to the top left-hand
corner of the requester box.

BAse

Interface instruction: set coordinate base for dialogue box

```
BA x,y;
```

The BAse command sets the reference point for all future coordinate calculations, and is used by simply setting the screen coordinates of the new origin. The default coordinate values are 0,0.

If this command is called more than once, the new BAse setting will replace the previous one. This coordinate reference point can also be set using the x,y parameters with the DIALOG BOX command, as explained earlier.

Here is a working example:

```
A$=A$+"BAse 50,50; set the coordinate base"
A$=A$+"INk 5,0,0; Graphic Box 0,0,150,50; draw a filled bar in ink five"
A$=A$+"PrintOutline 5,10,'AMOS Professional',2,4; print message in outline text"
A$=A$+"PRint 45,20,'Basic',4; display message in normal text"
A$=A$+"EXit; leave the interface program"
D=Dialog Box(A$) : Wait Key
```

Saving the background graphics

The simple drawing operations used in these examples would destroy any existing graphics on the screen. But AMOS Professional dialogue boxes should wink into place over an existing display, and then return the screen to its original state after use. The Interface includes commands that do exactly that!

SIze

Interface instruction: define the size of graphics to be saved

```
SI width,height;
```

The SIze command sets the size of the dialogue box on the screen, and prepares the Interface system to save the background graphics. The location of this screen area is set by giving the number of pixels for the width of the zone, followed by the number of pixels for its height, measured relative to the reference point already set by the BAse command.

SAve

Interface instruction: save background under the dialogue box

```
SA block number;
```

This is used to save the graphics defined by the previous BAse and SIze commands. The area that is saved into memory will be restored to the screen after the Interface program reaches its final EXit instruction. Give the SAve command, followed by the number of a memory block to be used for the graphics. If this has already been defined, it will be replaced by the new definition.

Note that each new control panel can save its own background area independently. Blocks will be re-drawn in reverse order when the dialogue panel is removed. If there is insufficient memory, an error will be generated when the program is initialised.

Supposing a dialogue box is to be positioned from coordinates 50,50 to 210,110. The following sequence of instructions could be used:

```
A$=A$+"BA 50,50; set the coordinate base"
A$=A$+"SI 160,60; SA 1; save area under dialogue box"
A$=A$+"IN 0,0,0; GB 5,5,155,56; IN 5,0,0; GB 0,0,150,50; draw a fancy box"
A$=A$+"PO 5,10,'AMOS Professional',2,4; PR 45,20,'Basic',4; print messages"
A$=A$+"EXit; quit interface program"
D=Dialog Box(A$) : Wait key
```

Unfortunately, when that example is run, the graphics will be removed from the screen immediately after they have been drawn! This is because DIALOG BOX executes the entire Interface program in a single burst, and jumps back to the main AOZ program as soon as it is completed. In order to use a dialogue box, the Interface
must be instructed to wait for an event of some sort. To make that example work properly, read on!

Waiting for an event

RunUntil

Interface instruction: run until conditions are satisfied

```
RU delay,flags;
```

This command runs an Interface program from the first command, and activates all buttons and sliders that have been defined in the dialogue box. The box now waits on screen, until a specific set of conditions have been satisfied.

These conditions can be anything from pressing [Ctrl]+[C], selecting a [Quit] icon, or simply hitting a key. The requester may also be displayed for a specific amount of time. If the original screen contents have been saved using the SAve command, they will be restored to normal as soon as the program returns to AOZ.

The RunUntil command is ideal for use with dialogue boxes that do not need to interact with the main AOZ program.

The delay parameter holds the interval time for the dialogue to remain on screen, specified in 50th of a second. If this value is greater than zero, the box will automatically exit after the chosen period has elapsed, with no n ed for
any user input. On the other hand, a value of zero will wait patiently for a user response, or until the conditions specified in the flag values have been met.

The flag parameter is a simple bitmap, with a value of 1 in the relevant position activating the feature, and a zero disabling it. Here are the settings:

- Bit 0 clears the keyboard buffer before running. This is similar to a CLEAR KEY command.
- Bit 1 ignores any accidental mouse key presses before the dialogue box is drawn.
- Bit 2 exits whenever a key is pressed from the keyboard.
- Bit 3 quits when the user clicks on one of the mouse keys.

Here are some example settings:

```
RUn 500,%1111; display box for five seconds or until mouse click or key press
RUn 0,0; wait for quit button explained later
```

You can now display a crude requester on the screen, by changing the last working example as follows:

```
A$=A$+"BA 50,50; set the coordinate base"
A$=A$+"SI 160,60; SA 1; save area under dialogue box"
A$=A$+"IN 0,0,0; GB 5,5,155,56; IN 5,0,0; GB 0,0,150,50; draw a fancy box"
A$=A$+"PO 5,10,'AMOS Professional',2,4; PR 45,20,'Basic',4; print messages"
A$=A$+"RU 0,%0110; wait for either a mouse click or a key press"
A$=A$+"EXit;"
D=Dialog Box(A$)
```

Requesters serve a useful purpose, but real control panels need to employ buttons, icons and slider bars. The AOZ Interface makes this very easy, and offers a large selection of options.

The last part of this Chapter deals with simple button commands Advanced control panels will be explained in Chapter 9.3. All these commands are used to display an object on the screen which is automatically assigned its own zone, and can then be manipulated in a variety of ways.

Interface buttons

BUtton

Interface instruction: define an Interface button

```
BU number,x,y,width,height,setting,minimum,maximum;[][]
```

The BUtton command defines a simple Interface button, which can then be selected directly using the mouse. This button system is very flexible, and can be used to generate dozens of different button types in a few simple lines of code.

These buttons can be read in several ways. If the RUn command has been used in the program the button value that has been selected will be returned immediately by a DIALOG BOX or DIALOG RUN command. Alternatively, a background dialogue box can have its buttons read directly from an AOZ main program, which is explained later.

Here are the BUtton command parameters, in order:

- After the BUtton instruction, the number of the button is specified, from 1 upwards. If several buttons are to be linked together, they can be assigned the same number without causing any problems at all. Each button can be individually tested by the RDIALOG function in an AOZ main program.
- The x,y coordinates hold the position of the button, relative to the BAse setting of the dialogue box.
- The width and height parameters determine the size of the rectangular test zone that is to enclose the new button, and are given in pixels as usual.

Setting refers to an initial value setting for the new button, starting from zero. This indicates the actual appearance of the position of the button on screen. In the case of a simple ON/OFF button, a value of zero could indicate OFF,
while 1 would indicate an ON setting. Normally, the value of this setting will increase by one every time the button is "clicked", but this may be changed directly, within your new button definition.

The minimum and maximum parameters are used to set the range of the allowable button settings. If the button exceeds the maximum limit, it is automatically set back to the minimum setting.

After these parameters have been specified, you are ready to draw the button on the screen. This is defined using a simple list of Interface commands enclosed in square brackets. These instructions can feature anything, including
any of the Interface graphics operations. The semi- colon in front of the first square bracket is essential!

The drawing routine is first called when the button is initialised, and it is performed again every time the button is activated by the user.

At the beginning of the routine, the coordinate base is moved to the specified x,y position, and the Size is set to the specified width and height. So all drawing operations are relative to the start position of the current button. This
means that there is no need to know the final location of the button on the screen, so the button may be moved around by simply changing the original coordinate values in x,y. The same drawing routines can also be used for several different buttons.
There is a second set of square brackets, which can remain empty, or contain optional commands. These are used to define any "change" routine, to be called every time the button is released.

Note that there is no semi-colon after this final set of brackets. if one is included a syntax error will be generated when the program is run. Beware of making this very easy mistake!

Whenever a button is selected by the user, the AOZ Interface performs the following sequence of actions:

- The button setting is incremented by one, and this new setting is compared to the values held in the minimum and maximum parameters.
- The commands held in the button drawing routine are now executed on screen.
- The system waits for the mouse key to be released
- If it has been defined, the "change" routine is now performed. This can change the setting value of any button on the screen, allowing you to link several buttons together. If the "change" commands move one or more buttons to a new setting, they will be updated with an appropriate call to the relevant drawing routine.

Here are some examples of button definitions:

```
BU 1,80,38,50,10,0,0,1; set position and size of button number one
[PR 1,2,'Button 1',6;][] draw button using a simple print command
BU 2,10,38,50,10,0,0,1;[PR 1,2,'Button 2',7;][]

### **_ButtonQuit_**

__Interface instruction__: trigger an exit button
```

BQ;

```
There are special buttons featured in the Editor requesters, such as [Ok] and
[Cancel], which are used to leave the dialogue box the moment that they are
selected by the user. These "Exit" buttons are created using the Button Quit
instruction. This command can be placed inside the "change" brackets in order to
trigger a forced exit from the dialogue box after the button has been selected.
For example:
```

BU 1,80,38,50,10,0,0,1 ;[PR 1,2,'Button 1',6;][BQ;]

```
## Drawing a button

Buttons can be drawn in a variety of styles, using any of the Interface graphics
commands. A full list of all these commands is fully explained in Chapter XX.X
(9.2). Here are some typical options:

Text buttons can be created by enclosing the required text with a box or a
filled bar, as follows:
```

BUtton 1,20,16,50,10,0,0,1;
[INk 4,0,0;GraphicSquare 0,0,50,10; PRint 1,2,'Button',6;][]
BUtton 2,120,16,40,1,0,0,1;
[INk 1,0,0; GraphicBox 0,0,33,10; PRint 1,2,'Quit',4;][ButtonQuit;]

```
Vertical buttons can be drawn using the Vertical Text command, like this:
```

BUtton 3,135,35,10,42,0,0,1;
[INk 0,0,0; Graphic Box 2,2,12,42; INk 6,0,0; Graphic Box 0,0,9,39;
VText 0,0,'Hello!',2;][]

```
Graphical Icons can be generated using a packed image held in the Resource bank.
They can be displayed with a simple call to the UNpack command, as follows:
```

[UNpack 0,0,1]

```
Complex buttons may be produced using a packed picture as the background, along
with any combination of the Interface text and graphics commands. The background
can also be generated from a whole line of images, using the powerful LIne
command. Here is a ready- made example to examine:

TODO - "Update adrress below to AOZ"
```

Load "AMOSPro_Tutorials:Tutorial/Interface/Simple_Requester.AMOS"

```
## Changing a button

The ability to call an Interface routine whenever the status of a button is
changed allows a range of useful effects to be generated. The key to these
effects is held by three Interface button functions, BPosition, BReturn and
BChange.

### **_BPosition_**

__Interface function__: return the setting inside a button definition
```

setting=BP

```
After a normal button is created with a BUtton definition, the position of the
current setting can be read straight from the drawing routine, using a simple
call to the BPosition function. BPosition returns a value from the minimum to
maximum, depending on the current setting of the button. It can be used to flick
the image of the button ON or OFF as part of the drawing routine, so that there
is one image for the selected button, and a different appearance for the
original version.

This can be used in a number of ways. Here are some examples for changing the
colour of text or background, using the INk command:
```

BUtton 1,50,38,50,10,0,0,1;
[INk 0,0,0; GraphicSquare 0,0,50,10; PRint 1,2,'Button',BPosition+5;][]

```
display highlighted text in a new ink colour
```

BUtton 2,90,38,50,10,0,0,1;
[INk BPosition 1+,0,0; change the background of the button when selected
GraphicBox 0,0,33,10; PRint 1,2,'Quit',4;][ButtonQuit;]

```
As mentioned earlier, another possibility is to display a packed image from the
resource bank.

The value of BPosition can be use4 to choose between two pictures representing
ON and OFF in the following way:
```

BUtton 3,180,38,50,10,0,0,1
[UNpack BPosition 13+;] toggle images 13 and 14 depending on bposition value
[] no change routine required in this case

```
The same technique can be used to generate buttons with a number of different
positions. All that is required is to specify the appropriate values of the
setting, minimum and maximum parameters in your button definitions, then test
BPosition as part of the main drawing routine. Here is a ready-made example
using this system:

TODO - "Update Address below to AOZ"
```

Load "AMOSPro_Tutorials:Tutorial/Interface/Button_Types.AMOS"

```
### **_BReturn_**

__Interface instruction__: change the setting of a button
```

BR new setting;

```
This instruction can only be used inside a dialogue change routine, which is
held in the second pair of square brackets, as explained earlier. BReturn moves
the button to the specified new setting, and changes the value of BPosition. It
then calls up the original drawing routine to update the display. If this
includes the BPosition function, the effect can be seen immediately.

This feature can be used to return the button to its original state, after the
mouse key is released. If it is omitted, the button will stay in its current
setting until it is chosen once more. Here is a schematic example:
```

BUtton 1,160,100,64,16,0,0,1;[drawing routine...][BReturn 0;]

```
### **_BChange_**

__Interface instruction__: change the setting of any active button
```

BC number,new setting

```
An alternative way to change the setting of a button is to make it behave like a
radio button. Here, only one of a group of buttons may be switched on at any
time, and as soon as it is "on" the other related buttons automatically appear
to be switched "off".

The BChange command is used to alter the setting of any active button on the
screen, simply by specifying the number of the button or buttons that are to be
changed, followed by the new setting required.

If many buttons are to be affected by this technique, the process can be
simplified by assigning them all to the same identification number. Note that
BChange has no effect on the current button, even if it is specifically given in
the
instruction. To change the setting of the current button, the BReturn command
should be used instead.

Here is an instant working example of every button type:

TODO = "Update Address below to AOZ"
```

Load "AMOSPro_Tutorials:Tutorial/Interface/Working Buttons.AMOS"

```
## Keyboard short-cuts

Any of your buttons can be optionally assigned to an equivalent control-key
combination from the keyboard.

### **_KY_**

__Interface instruction__: set keyboard short-cut
```

KY ascii,shift

```
The KY command requires two parameters. The first is the Ascii code of the
chosen key (scan codes can be entered by simply adding 128 to their value). The
shift parameter is a bitmap which allows a test to be made for one or more
control keys, and the default value is zero. The full list of possibilities is
explained under the KEY SHIFT function in Chapter XX.X (10.1).

Once the keyboard short-cut has been defined, the button may be activated by
pressing the appropriate key. This will click the button on screen and release
it immediately. Here are some example settings:
```

KY 13,0 this is the return key
KY 128 76+,0 this tests the up arrow key which has scan code 76

```
### **_NoWait_**

__Interface instruction__: specify a quick release button
```

NW;

```
It is often necessary for buttons to take effect immediately, without waiting
for the mouse key to be released. The NoWait command is provided for this
purpose, and it is normally used with a BQuit instruction, or with routines that
need to interact directly with an existing AOZ program. It is used like this:
```

BUtton 1,120,16,40,10,0,0,1;[In 1,0,0; GB 0,0,33,10; PR 1,2,'Quit',4;][NoWait,BQ;]

```

```