AOZ STUDIO

# Magic Book - Livre de Magie

# The Magic Book is a short summary of the User Guide

Herein lie some important instructions so you can quickly find the right ingredients to make your magic spell successful, and bring your creations to life! You will find much more in the User Guide and the AOZ Studio Interactive Search.

## Let it Show!

*The default screen resolution that AOZ is using for graphics is: 1920 x 1080 pixels, and for text: 80 x 25 characters. Imagine screens as a pile of sheets of paper, and as you open them, you place them on top of each other. If one has transparent colours, you will see parts of the screen below it.*

*Unless otherwise specified, all instructions and functions have an effect to the current screen. You need to tell AOZ which screen you are working on with Screen <s>. By default you don't have to open screen 0.*

**Screen s**

Make Screen s active (screen 0 is set by default). **Ex: Screen 1**

**#DisplayWidth: w / #DisplayHeight: h** *(2 instructions here)*

Sets the display Width / Height. **Ex: #DisplayWidth: 1000**

**=Display Width / =Display Height.** **Ex: DW=Display Width**

Returns the value of the width or height of the AOZ display.

**Screen Open s, width, height, colours*, mode, lines, border, tags***

Opens a screen s, specifying pixels width and height, Number of colours of the palette. Optional: mode (lowres, hires…), Number of lines of the text window, border, tags for special properties.

**Screen To Front s / Screen To Back s**

Pulls screen s to the top or the bottom of the stack so it covers the others. (If s is omitted, pulls the current screen to the top.)

**Screen Show s / Screen Hide s**

Makes screen s visible or not (If s is omitted, makes the current screen visible.)

**Cls / Cls c**

Clears the current screen **Ex: Cls** / the one indexed by colour c. (If c is omitted, clears the current screen to the current colour.) **Ex: Cls 0**

## Displaying Texts

You can use the instructions Print or Text to display texts.

**Print ""**

This will print the text in between **" "** or number if no **""**
**Ex : Print "Hello"** Will print Hello as a text
**Ex : Print 2** Will print 2 as a number
**Ex : Print 2+2** Will print 4, doing the calculation

**Print x, y$, z#** (and so on)

This will print each variable in the order listed, starting at the current text cursor position on the current text window.
**Ex : Print "Hello","AOZ";"Studio"** Note the separators **;** and **,**
-When a comma is used as the separator the cursor will be positioned at the next tab (can be modified using Set Tab n). Tab stops are used to align text in columns.
-When a semicolon ; is used the cursor will be just to the right.

**Print Using f$; x, y$, z#** (and so on)

This will print each variable formatted using the format string f$. Certain characters will be replaced with the contents of the variables listed. **Ex: Print Using "+## ~~~ -#,###.#"; 1,"ab",1234.56**
*Result:* + 1 ab 1,234.5 (more details on the User Guide)

**Curs On / Curs Off**

Display or hide the text cursor

**Locate x,y**

Set the Print position, x, y are character (TEXT) positions, not pixels.

**Pen x**

Sets the pen (text) colour to the colour indexed by x in the current screen's palette (See Palette). By default pen is Pen 1 for white, 4 for Red, 5 for Green, 6 for Blue, 0 for Black: **Ex: Pen 4 : Print "Hi"**

**Paper x**

Sets the printing Paper (background) colour to the colour indexed by x in the current screen's Palette. **Ex: Paper 0**

**=Pen**

Returns the value of the Pen colour index. **Ex: x=Pen**

**=Paper**

Returns the value of the Paper colour index. **Ex: Print Paper**

**Text x,y,t$**

Displays t$ at the x,y graphical coordinates on the current screen, using the current graphical font settings. If the x and/or y coordinates are omitted the coordinates will be read from the graphic cursor.

***A combo example using screens with text and graphics:***

```
#googleFont:"syncopate"
// Create Screen 0 :
Screen Open 0,1920,1080,32,0 : Curs Off
// Create Screen 1, transparent paper color, Ink is the Text
colour :
Screen Open 1,1920,1080,32,0 : Curs Off : Set Transparent 0 :
Paper 0 : Ink 1
Set font "syncopate", 70

Screen 0        // Switch to Screen 0
        Actor "magic", image$="magic", X= 0, Y=10
Screen 1        // Switch to Screen 1
        Text 5,90, "WELCOME"
        Print "AOZ"
```

## Drawing Commands

> ***Tips:***
> *AOZ will remember the last point in use, called the graphic cursor.*

**Plot x,y,c**

Draws a pixel at graphic coordinates x, y using the colour indexed by c. (If c is omitted, uses the current Ink colour.)

**Draw x1,y1 To x2,y2**

Draws a line between x1,y1 and x2,y2.

**Ex : Ink 1 : Draw 50,50 To 800, 50**

**Draw To x1,y1**

Draws a line from the graphic cursor to x1,y1.

**Polyline x1,y1 To x2,y2 To x3,y3** (and so on)

Draws a line from the first to the next point, then from there to the next, etc.

**Ex: Ink 1 : Polyline 100,100 To 100,400 To 400,400**

Also: Polyline To x1,y1 To x2,y2 To x3,y3 (and so on)

**Polygon x1,y1 To x2,y2 To x3,y3** (and so on)

Draws lines between each of the specified points to form a polygon, using the current Ink as the fill colour. If the last x, y coordinates do not match the first, a final line will be added going back to x1, y1.

Also: Polygon To x1,y1 To x2,y2 To x3,y3 (and so on)

**Ex: Ink 4 : Polygon To 100,400 To 400,400**

**Box x1,y1 To x2,y2**

Draws a box defined by the corners x1,y1 and x2,y2 using the current Ink colour.

**Bar x1,y1 To x2,y2**

Draws a filled box defined by the corners x1,y1 and x2,y2 using the current Ink.

**Circle x,y,r**

Draws a circle centered at x, y, with a radius of r, in the current Ink colour.

**Disc x,y,r**

Draws a circle centered at x, y, with a radius of r, in the current colour.

**Ellipse x,y,rh,rv, rot, fill**

Draws an ellipse centered at x, y, with a horizontal radius of rh and a vertical radius of rv, rotated by rot, using the current colour.

**Filled Ellipse x,y,rh,rv, rot, fillIndex, lineIndex**

Draws an ellipse centered at x, y, with a horizontal radius of rh and a vertical radius of rv, rotated by rot, using the colour indexed by lineIndex, and filled with the colour indexed by fillIndex.

**Star x,y,r1,r2,points,rot,fill**

Draws a points pointed star centered at x, y, with an inner radius of r1 and an outer radius of r2, rotated by rot, using the current colour.

**Filled Star x,y,r1,r2,points,rot,fillIndex,lineIndex**

Draws a points pointed star centered at x, y, with an inner radius of r1 and an outer radius of r2, rotated by rot, using the colour indexed by lineIndex, and filled with the colour indexed by fillIndex.

**Shape x,y,r1,r2,sides,rot,fill**

Draws a shape centered at x, y, with an inner radius of r1 and an outer radius of r2, rotated by rot, using the current colour.

**Filled Shape x,y,r1,r2,sides,rot,fillIndex,lineIndex**

Draws a filled shape.

**Arc x,y,r,angle1,angle2,rot,fill,ccw**

Draws an arc of a circle centered at x, y, with a radius of r, between angle1 and angle2, using the current Ink colour, and rotated at rot. If fill is true, a cord is drawn between the endpoints, and the enclosed area is filled using the same colour. If ccw is true, the arc is drawn in a counter-clockwise direction.

**Filled Arc x,y,r,angle1,angle2,rot,fillIndex,lineIndex,ccw**

Draws an arc of a circle centered at x, y, with a radius of r, between angle1 and angle2, using the colour indexed by lineIndex, and rotated at rot. A line is drawn between the endpoints, and the enclosed area is filled using the colour indexed by fillIndex. If ccw is true, then the arc is drawn in a counter-clockwise direction.

**Ellipse Arc x,y,r,angle1,angle2,rot,fill,ccw**

Draws an elliptical arc centered on x, y, between angle1 and angle2, using the current Ink colour, and rotated at rot. If fill is true, a line is drawn between the endpoints, and the enclosed arc is filled using the same colour. If ccw is true, the arc is drawn in counter-clockwise.

**Filled Ellipse Arc x,y,r,angle1,angle2,rot,fillIndex,borderIndex,ccw**

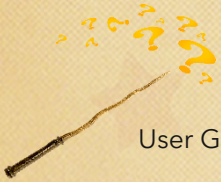
Draws an elliptcal arc centered at x, y, between angle1 and angle2, using the colour indexed by lineIndex. A cord is drawn between the endpoints, and the enclosed area is filled using the colour indexed by fillIndex. If ccw is true, the arc is drawn in counter-clockwise.

**Ink** c

Sets the graphic Ink colour (to the colour indexed by c in the current screen's Palette).

**=Ink**

Returns the current colour index of the graphic Ink.



AOZ have 1400 instructions, please use the User Guide and the AOZ Studio Search bar.

## Set the scene, display the Actor

Actor is a sophisticated instruction to manipulate 2D, 3D images, videos, on the screen. You can check, change and control the behavior of your Actors, manage collisions and so much more. You can do games, great user interfaces, portals, applications,…
Here are the main parameters for the Actor instruction:

- **Name or Index:** Set the Actor by giving a name Ex Actor "A" or Actor "magic", or an index. Ex using the Actor with Indexes (an integer value):

```
For J=0 to 4
   Actor J, Image$="magic", X=10, Y=J*200,
   Endx=1800, duration=1000*J
Next J
   Wait Input
```

- **Type$**: Type of the Actor. "2d" by default, or "3d"
- **X:** Horizontal position of the actor on the screen. 0 by default.
- **Y:** Vertical position of the actor on the screen. 0 by default.
- **Z:** (3D) Depth position of the actor on the screen. 0 by default.
- **StartX / StartY:** The start horizontal/vertical position of the actor's movement.
- **StartZ:** (3D) The depth start position of the actor's movement.
- **EndX:** The horizontal position at the end of the actor's movement.
- **EndY:** The vertical position at the end of the actor's movement.
- **EndZ:** (3D) The depth position at the end of the actor's movement.

- **Duration:** Duration of the actor's movement in milli secs.
  **Ex : Actor "magic", Image$="magic", X=0, EndX=2000, duration=4000 : wait input**

- **Image / Image$:** Index or name of the image. Ex: Image=1 or Image$="magic" with Images located In your *app/resources/images* folder, or already in *the AOZ Drive /AOZ Drive/resources/images*, or using the Load Asset instruction to specify where it is.
- **Video$:** Name of the video located in the *app/resources/images* folder or loaded by Load Asset used like an image for this Actor.
  **Ex : Actor "mycam", Video$="@webcam"** display the webcam1 by default (not in the AOZ Viewer, only on the web browsers).
- **Videoplay:** Play or stop the video. True by default.
- **Videoloop**: Play the video in loop. True by default.

- **Control$:** To direct the actor ("keyboard", "mouse", "joystick").
  **Ex : Actor "m", Image$="magic", control$="keyboard"**
  *By default the arrow keys and A D W S (can be changed).*

- **Transition$:** Assign a movement, by default "linear". AOZ is using the powerful CREATE.JS API, it is a great product with this effects: https://www.createjs.com/demos/tweenjs/tween_sparktable

- **Scale:** The decimal value of the actor size scale. Default: 1.0 (normal size). **Ex :Scale=0.5** (for 50% in X and Y).
- **ScaleX / ScaleY:** The decimal value of the X/Y scale of the actor.
- **StartScale / EndScale:** The scaling Start / End size decimal value.
- **StartScaleX / EndScaleX / StartScaleY / EndScaleY:** The decimal scaling value /Start / End / Width(X) / height(Y).
- **Angle / StartAngle / EndAngle:** Actor's angle / Start / End rotation in degrees.

- **Anglez / Startanglez / Endanglez:** (3D) Modification of the 3D Actor Z angle / Start / End of rotation. If not set, the last angle value is used.
- **Alpha / StartAlpha / EndAlpha:** Decimal value of the opacity / Start/End of the actor. Default 1.0 (totally visible).
- **SkewX / StartSkewX / EndSkewX:** Decimal value of horizontal / Start/End distortion of the actor.
- **SkewY / StartSkewY / EndSkewY:** Decimal value of vertical / Start / End distortion of the actor.
- **Visible:** True/False shows or hides the actor.
- **Enable:** Enables or disables an actor. If False, the Actor will still be displayed on the screen, but the controls, animations and mouse actions will be disabled.
- **Collision:** Enables or disables all collisions with this actor. If False, no collision effects will be applied.
- **LoopMove:** True/False play the movement as a loop.
- **ActionMove$:** "play" or "pause" the movement.
- **Spritesheet$:** The name of the spritesheet to use for actor animation.
- **Anim$:** Name of the animation to play as set in the sprite-sheet or defined with the Actor Animation instruction.
- **Actor Animation** : Create an animation without Spritesheet. Sequence$ is a string with each image name or number separated by a comma. The last value can be "L" to loop, "E" to end, "R" to reverse and loop.
  You can to set a delay in milli secs between 2 images like this: "1:1000, 2,..." to pause 1 sec. between image 1 and 2.
  Actor "Magic", AnimPlay=False (or True) to play or stop

  Example:

```
Actor Animation "gorun", Sequence$="1,2,3,4,L"
Actor "magic", Anim$="gorun"
```

- **LoopAnim:** True/False to play a loop animation or not.

- **Hotspot$ / HotspotX / HotspotY:** Decimal or text value that determines the position of the actor's "hot spot" (see the Hotspot Chapter). **Ex Hotspot$="middle"**
- **HRev / VRev:** True/False horizontal/ vertical mirror effect.
- **LeftLimit or RightLimit:** Value in pixels of the left or right (X) edges of the displacement limit of the actor.
- **TopLimit or BottomLimit:** Value in pixels of the Y edges of the displacement limit of the actor.
  **Ex : Actor "magic", Image$="magic", control$="mouse", TopLimit=100, BottomLimit= 600**

- **LookAt$:** Creates an automatic movement: the actor is looking towards an object, actor or point on the screen.
- **Auto$:** Creates an automatic movement of the actor using the same Control$ properties: offsetX, offsetY, angle, forward and backward.
- **Group$**: Assigns the actor to a custom group. A custom group can be assigned to several Actor. The group can then be controlled like an actor. Example for a group of enemies.

- **OnChange$:** Name of the procedure to be called when the actor undergoes a change (movement, form, ...).
- **OnAnimChange$:** Name of the procedure called for each change of the actor animation.
- **OnCollision$:** Name of the procedure called when the actor collides with another. Note: Another way to simply handle collisions is the Actor Col() function which becomes true (True) if collision.
  **Ex: If Actor Col ("magic", "lucie")=True Then ...**
- **OnLimit$:** Name of the procedure called when the actor reaches one of the edges..
- **Onmouse$**: Name of the Procedure that will be called at each mouse action On the actor.

- **Onmouseenter$**: Name of the Procedure that will be called when the mouse pointer enters over of the actor.

- **Onmouseclick$**: Name of the Procedure that will be called at each mouse click On the actor.
- **Onkeypress$**: Name of the Procedure that will be called at each key pressed.
- **Onkeydown$**: Name of the Procedure that will be called at each key down.
- **Onkeyup$**: Name of the Procedure that will be called at each key up.
- **Oncontrollerconnect$**: Name of the Procedure that will be called at each mouse doubleclick On the actor.
- **Oncontrollerdisconnect$**: Name of the Procedure that will be called at each mouse buttOn down On the actor.
- **Oncontrollerbutton$**: Name of the Procedure that will be called at each mouse buttOn up On the actor.
- **Oncontrolleraxis$**: Name of the Procedure that will be called at each mouse doubleclick On the actor.
- **Oncontrollerdirection$**: Name of the Procedure that will be called at each mouse button up on the actor.
- **Actor X / Actor Y** : Give the Actor position
  **Ex : PosX = Actor X ("magic")**

- **Actor Del ""** : Erase the actor(s).
  **Ex : Actor Del "magic"** erase the actor magic
  **Ex : Actor Del "*"** erase all the created creators

Example (Magic moves with the keyboard):

```
Actor "Magic", X=10, Y=200, Image$="magic",
OnMouse$="MOUSEEVENT", Control$="keyboard",
Oncollision$="MOUSEEVENT"
Actor "Lucie", X=400, Y=200, Image$="lucie",
OnMouse$="MOUSEEVENT", OnMouseEnter$="MOUSEEVENT"

Do
  Refresh  // use it when you move many graphx
Loop

Procedure MOUSEEVENT [EVENT$, BUTTON, INDEX$, INDEX1$,
INDEX2$]
        Cls : Print INDEX$ ;": "; INDEX1$ ;" "; INDEX2$ + "
EVENT$:"+EVENT$ + "  BUTTON:";BUTTON
End Proc
```

## Loops and Loops

**Do ... Loop**

Code is continually processed between Do and Loop unless a flow control instruction interrupts the loop. (Loop branches back to the code immediately following the Do instruction.)

**While** *[condition]* **... Wend**

Runs the code between While and Wend as long as condition is True. When condition becomes False, branches to the code immediately following the Wend instruction.
Wend Jumps back to the associated While instruction.

```
While J<5
    Inc J : Print J    // Inc J increase J of 1, same as J=J+1
Wend
```

**Repeat ... Until** *[condition]*

Same, it is running the code between Repeat and Until as long as the condition, this time at the end of the loop, is true. If condition is False it resumes at the code immediately following the Until

```
Repeat
    Inc J : Print J
Until J>4
```

**For ... To ... Step ... Next**

The famous loop, very useful. Step is an option. Inside the loop, the index (below I) can be used by the program as if it is a normal variable.

```
For J=1 To 10 Step 2
    Print J     // loop for J equal 1 to 10 by step of 2
Next I
```

**Exit / Exit n / Exit If** [*condition*] **/ Exit If** *[condition],* **n**

Similar to Exit, exits a loop immediately, branching to the code just after the associated Loop, Wend, Until, or Next. If n is used, exits n nested loops if condition is True. condition may be any valid Boolean expression or variable.

**If ... Then ... Else**

Is another famous instruction, to do the tests. Else is an option. There are several ways to use it, the simplest of which is: If <condition is true> Then <do this> **Ex : If A=1 Then Print "A equal 1"**

## Music and Sounds

**Shoot / Boom / Bell / Explode**

Play simple predefined sounds. **Ex : Boom**

**SamPlay**

**NEW Sfxr extension Interface of the Sfxr library**, synthesizer of game Sound Effect:

* Sfx Play // play a sound effect without fuss
* Sfx Create // create a new sound effect
* Sfx Del // Delete a sound effect
* Sfx Mutate // Slightly change the sound effect
* Sfx To // Assign a sound effect to a voice
* Sfx Set // Set a property of a sound effect
* =Sfx Json$ // Returns the JSON definition of a sound effect
* =Sfx Get // Return the value of one of its properties
* =Sfx Get$ // Return the value of one of its string properties

## Need More Input! Keyboard Commands

**Input x$**

Returns in the variable here x$ the text typed on the keyboard. Optionally you can display a text before a semicolon like this:
**Ex: Input "name?";name$**

**=Inkey$**

Return the key if pressed otherwise an empty car. "". For certain keys, such as the function keys, AltGr,… use the ScanCode function.

**=ScanCode**

Returns the scan code (ASCII standard). You can use the Keyboard Tester accessory to determine the scan code.

```
Do
  A$ = Inkey$ : B= ScanCode
  If A$<>"" then Print A$,B   // If A$ is empty ("") loop continues
Loop
```

## Analog/Digital Gamepad/Joystick

**=Gamepad Axis ( g, axis )**

Position of the analog/digital axis on gamepad g, between -1 (-100%) and 1 (100%). 0 = center (0%). Depends on the web browser and the gamepad/ joystick. (Use the Joystick Tester App).

**=Gamepad Axes ( g )**

Returns the number of axes available on gamepad g.

**=Gamepad Button ( g, b )**

Returns true when button b on gamepad g is pressed.

**=Gamepad Buttons ( g )**

Returns the number of buttons available on gamepad g.

**=Gamepad Connected ( g )**

Returns true when gamepad g is connected.

**=JLeft ( j, lock )** **=JRight ( j, lock )**
**=JUp ( j, lock )** **=JDown ( j, lock )**

Returns true if joystick number j is pointed in the specified direction. If lock is true, waits until specified position is released.

**=JUpLeft ( j, lock )** **=JUpRight ( j, lock )**
**=JDownLeft ( j, lock )** **=JDownRight ( j, lock )**

Returns true if joystick number j is pointed in the specified directions. If lock is true, waits until both directions are released.

**=Fire ( j, lock )**

Returns true if the main fire button on joystick number j is pressed. If lock is true, waits until fire is released before another shot is fired.

**=Joy ( j )**

Returns full state of Joystick j in binary form. That means that each digit (bit) represents one digital joystick functions, as follows:

| Bit Value | Meaning |
|---|---|
| 1 | Joystick is in up position |
| 2 | Joystick is in down position |
| 4 | Joystick is in left position |
| 8 | Joystick is in right position |
| 16 | Fire button is pressed |

```
Do
   J = Joy ( 0 ) : Locate 0,0 : Print Bin$( J, 5 ), J
   Refresh
Loop
```

The digital joystick default commands (change with JoyKey) are JUp(), JDown(), JLeft(), JRight(), and Fire(), they support keyboard emulation.

## And the Mouse?

**=X Mouse  =Y Mouse**

Returns horizontal and vertical position of the mouse coordinates.

**=Mouse Key**

Returns the status of all the buttons on the mouse as a bitmap. 1 indicates the left button is pressed, 2 left, 3 both, 4 central button.

```
Do      // try pressing more than 1 button
  Print Mouse Key, X Mouse, Y Mouse
Loop
```

**Limit Mouse x1,y1 To x2,y2**

Limit the mouse to the rectangular area x1,y1 and x2,y2.
**Limit Mouse** Restores mouse movement to the full screen.

**Limit Mouse x1,y1,width,height**

Limit the mouse movement to x1,y1, width pixels wide and height pixels tall.

**Change Mouse <name of the image** (without extension)**>**

You can use an Image as the mouse pointer. PNG only, with a maximum size of 128x128px. Put the image in your application folder: resources/assets/mouse.
**Example: Change Mouse "01mouse"**

### Using the mouse with examples:

1°) check where your mouse pointer is using zones:

```
Reserve Zone 2 : Line Width 10  // reserve 2 zones, set the line width
// draw and set 2 zones:
Ink 1 : Pen 1
Box 80,400,1800,200 : Set Zone 1, 80,400,1800,200
Box 80,700,1800,200 : Set Zone 2, 80,700,1800,200
Do  // Loop to check where the mouse is located
 A= Mouse Zone       // will return the zone Number (0= out of zones)
 Locate 1,5 : Print A   // Display the Number
Loop
```

2°) check if user click and where on a big Actor:

```
Actor "forest", Image$="forest", Y=50, OnMouse$="MOUSEEVENT"
Wait Key   // so ne program do not end until you press a key

Procedure MOUSEEVENT [ INDEX$, EVENT$, X, Y]
   Cls : Print "ACTOR:"; INDEX$ ; "   EVENT$:" ; EVENT$ ;"   "; X;Y
End Proc
```

## Wait for me!  Timer and Wait Commands

**Wait Click**

Pauses the application until a mouse button or screen has been pressed.

**Wait Input**

Pauses the application until a mouse button has been pressed or the screen has been touched, or a keyboard key has been pressed.

**Wait x**

Pauses the application for x seconds. **Ex Wait 2**
In Amiga mode, time is measured in Vbl cycles.

| Command | Amiga Pal | Amiga NTSC | AOZ |
|---|---|---|---|
| Wait 1 | 1/50 second | 1/60 second | 1 second |
| Wait 0.5 | 1/100 second | 1/120 second | 0.5 second |

**Refresh (or Wait Vbl)**

Synchronize the display. **It is needed within the display loops**.
Example of a loop with 2 animations. Try without the Refresh.

```
Do
    score=score+1
    UI Progress "barre", x=10, y=200, width=1900, height=50, value=score/20
    Actor "magic", Image$="magic", X=score
    Refresh
Loop
```

# Avoid Spaghetti Code!

## Goto and Gosub

*It is good programming practice to avoid using many Goto. Abuse of this instruction may lead to difficult to follow and to maintain "spaghetti code", better use Procedures or Gosubs.*

## First the Labels

A label identifies a "jump point" in the program. It is a position it can "jump" to using Goto, Gosub, Then, Else, On, etc.
Labels may contain letters and numbers, but must start with a letter. Optionally labels may be computed. A label definition must be the first item on a line of code and have a colon **:** at the end of it. When jumping to a label, the **:** is omitted. Better with an example :

```
MyLabel:              ' once set here you may use it like this:

Goto MyLabel          ' Jumping to MyLabel via Goto     (note no : )
Gosub MyLabel         ' Jumping to MyLabel via Gosub    (note no : )

' Dynamic "computed" labels:
Gosub Chr$(Asc("A")+n)        ' If n=0, label="A", If n=1, label="B"
```

## Line Numbers

If a label starts with a Line numbers it work identically to labels, except that the colon at the end is omitted.

```
100 Print "Hello."          ' Typical use of line number
Gosub x*10                  ' Computed line number
Goto 350                    ' Jumping to line number 350
```

**Goto** and **Gosub** branches to the specified label or line Number and continues executing the code there.
Gosub is different: if the code reaches a **Return** the execution resumes just after the Gosub. Gosub always works with Return.

**Procedure**

A procedure is a block of code that you can run by calling it by a name you choose:

```
Myproc [3]       // calling the procedure
Myproc [10]      // calling again the same procedure
Procedure Myproc [A]  // with the parameter in [] set here in A
    Print A*2          // as many lines of code before the End Proc
End Procedure     // declare the end, to return after the call
```

An other example:

```
Print Myproc$[3]

Procedure Myproc$[a]  // input the a value in the procedure
S$=str$(a*2)                // Str$() convert a number in a text
End Procedure[S$]        // output the S$ value from the procedure
```

**Pop Proc**

Exit the procedure without running the remaining code

**Exit App**

Quit an application and close the window

## Variables, Constants, and Data Types

AOZ is an extension of the famous BASIC language, like Ada, Pascal, SQL,… it is cases insensitive, IE the variable Flag is the same as flag.

Variable names may contain letters, numbers, underscores_ By default a variable is an integer otherwise you have to declare the variable types:

- **Integer** (whole numbers): X=68000 (Default)
- **Float** (real numbers): Y**#**=-123.45 (note the #)
  **Ex : Temp=19.5 : KO Temp#=19.5 : OK**
- **String** (alphanumeric): Z**$**=**"**Fred@3**"** (note the $)
  Strings must be surrounded by double quotation marks.)
  **Ex : NOM="AOZ" : KO NOM$="AOZ" : OK**
- **Hexadecimal** X=$03D0 (note the $)
- **Binary** Y=%0110 (note the %)
- **Dim** With manually set dimensions Dim A (10, 20).
  Automatic **Ex : Dim A$() : A$( 1000 ) = "Hello!" : Print A$( 1000 )**
  Or mix of both. Dim TOTO@ ( , 100 ).

# LOCAL BACKUP AND DATABASE

The local memory of browsers (example Chrome) can store data in the form of Cookies, but be careful, as long as the browser's cache memory is not cleared.

## Saving a file or data in cache memory.

**Open In / Open Out / Close**

1. Open for writing with: Open Out channel number, file name
2. Save with: Print #Open channel number, data
3. Close an open channel: Close Open channel number

Example:

```
Filename$="MyFile.txt"
Input SA$  // text to save

Open Out 1, Filename$
Print #1,SA$
Close 1

Open In 1, Filename$
Input #1, RD$
Print "READ ";RD$
Close 1
```

## Using a database.

AOZ includes instructions for Firebase (hosted at Google) and for SQL (hosted at AOZ Studio or elsewhere). Summary for SQL:

- **DB Database:** select a database
- **DB Table:** select a table
- **DB Read:** copy the values of the record into your AOZ variables
- **DB Write:** copy the content of variables into the selected record
- **DB New:** selects a new empty record. (To be used before a new DB Write.)

– **DB Search All:** searches all records and selects the first one
– **DB Search:** searches for records according to a simple search criterion and selects the first one
– **DB Search Sql:** searches for records according to search criteria written in SQL language and selects the 1st
– **DB First:** selects the first record resulting from the search
– **DB Last:** selects the last record resulting from the search
– **DB Next:** selects the next record resulting from the search
– **DB Previous**: selects the previous record resulting from the search
– **DB Pointer:** retrieves the identifier of the selected record
– **DB Point On:** selects a record whose identifier is known

Example of a program that writes and reads:

```
name$="AOZ": score=987654
DB Write "name$, score"

DB Read "name$, score"
Print name$, score
```

Note: if the name of the database and/or the table is not defined, AOZ creates default ones. See User Guide.

AOZ includes 1400 instructions, this book is only an extract.
For more please consult all the tricks on: doc.aoz.Studio
Support is on DISCORD: https://fr.aoz.studio/faq

AOZ STUDIO

V1.0 Dec.22