The manifest

At the root of any AOZ application should be located a file called "manifest.hjson" that contains the various settings for both the compiler and the application.

This file is a simple UTF-8 file, encoded in enhanced JSON format (HJSON) allowing comments and simplifying its display.

The format of JSON a simple text format allowing to define 'properties' in a structured way. I have tried to organize the content of the file in a logical manner, and provide comments to help you modify it. I will also provide in the near future several different manifest configuration to cope with the various applications you might want to create.

I will certainly in the future create (in AOZ of course :) a manifest editor that will present the various options in a simplified graphical way.

Please note that the manifest is constant work in progress. You can play with it, but always keep a copy of a working manifest somewhere!

How to edit the manifest.json file?

The file is a simple text file editable with any source code editor, encoded in UTF-8. The compiler supports the cariage return formats of all platforms, and therefore a manifest saved on a mac will be compilable on a PC and vice-versa.

IMPORTANT

Be careful when you modify values in the manifest: punctuation is important. Strings should be encoded within double quotes. Do not remove commas or accolades. Any illegal change in the file will generate an "Illegal manifest" error during compilation.

The different parts of the manifest

Let's examine the different components of the manifest file.

Version number

```
    // Version of the manifest
    version: "8",
```

The first property contains the version of the manifest, number 5 when writing this doc. This number is crucial and MUST match the number that the compiler expects. If numbers do not match, the compiler will generate an error and refuse to compile.

Each time a new value is added to the manifest (as development progresses), 1 is added to the version number and the previous manifest files must be replaced by the new one. This is the reason why I highly suggest that you keep only ONE manifest file for all your AOZ projects, this file being located in the "default" folder of the amos-2 directory (see next chapter).

Application information

```
    // Information about the application
    infos:
    {
        applicationName: "Hello world!",
        author: "By Francois Lionet",
        version: "Version 1.0",
        date: "Created on the 07/02/2019",
        copyright: "Public domain!",
        start: "main.amos"
    },
```

This simple section contains information about the application, its author, the version, date of creation and copyright.

One field is more important that the other though, it is the "start" field. This field contains the name of the file that contains the Basic source-code of the application. I have as a default, used "main.amos", and this is the reason why you will find the source code of all the demos and examples of the distribution to be called "main.amos'.

But nothing prevents you from changing this name and renaming it to "source.bas" or anything you have in mind, as long as the compiler can find the file that it refers to. Of course, if it cannot find the file, the compiler will generate an error and will not compile.

Compilation mode

```
// Compilation mode
// ----------------
compilation:
{
    speed: "fair",
    syntax: "amosPro",
    emulation: "1200"
    usePalette: true,
    useShortColors: true,
    showCopperBlack: false,
    unlimitedScreens: true,
    unlimitedWindows: true,
    maskHardwareCoordinates: false,
    endian: "big"
},
```

This very important section defines how AOZ tries to emulate, or not, the Amiga computer when running your application.

- speed

  To achieve instructions such as Goto and On Error Goto and real endless loops (like the main loop of a game) in a Javascript application, AOZ has to 'stay in control' of the code executed. This means that the code generated by the compiler is cut in 'chunks' and the hand is given back to AMOS, which eventually gives it back to the browser to avoid the infamous 'unresponsive' Javascript pages as we have all seen once in our lives.

  AOZ compiler tries to be as smart as possible, and groups as many instructions together so that the resulting code stays as fast as possible. Hand is given back to AMOS at the end of loops, for GOTO statements, and at waiting instructions (like Wait Key or Wait Vbl).

Applications can be very different from one another, and the system implemented has to cope with all of them. Unfortunately there is not way for the compiler to 'evaluate' the code of an application and find if it is mostly graphic oriented, or 'data' oriented etc. This is the reason why the manifest contains the 'speed' property, to fine tune the execution of the application in case the default setting does not apply.

The various value for this property are:

- "fair" : the default value that should work for most of the applications. The application is run at the best possible speed while maintaining enough processor time for the browser and the machine to work properly. Processor load in this mode should be (at maximum for heavy processing) around 35%.
- "safe" : some applications do a lot of calculations without a lot of display. Or consist of very small loops turning very fast. The "fair" mode may not work fine for them and the browser may seems stuck, or the keyboard be irresponsive. If this happens to you, select this mode : the application will be a little bit slower and the browser will have more processor time.
- "graphics" : some application are very graphic incentive. The problem with graphical functions such as Paint, Polygon etc. is that they 'take the hand' and need a lot of time to complete, thus corrupting the multitask of the browser and the machine. If you have an application with loops of Plot, Bar, Polygon, Paint etc. use this setting. The application will be slower but the computer will not look as it has crashed.
- "fast" : use this option at the end of development when you are sure that ALL the main loops of your application or game are stabilized by a Wait Vbl instruction. This option reduces the intervention of AMOS to a minimum and concentrates on the application's code, resuting in an execution speed close to the speed of true Javascript (very fast). Warning: if some loops in your application are not stabilized by a Wait Vbl instruction and take several seconds to execute, the browser will appear 'frozen' during that time and keyboard (including Control-C) will be un-responsive.

- syntax (not implemented)

  This property indicates the name of the instruction set with which the application was programmed. It can take several values:

  - "amos1.3" : only keywords present in the 1.3 version of the original AMOS will be supported, all others will generate synatx errors at compilation.
  - "amosPro" : all keywords up to AMOS Professional latest version
  - "aoz" : all keywords up to AMOS Professional AND the new ones specific to AOZ. This is the suggested settings, the new keyword being designed not to interfere with the original ones.
- emulation (partially implemented)

  This property indicate what machine is emulated when your application runs.

  Modern machines provide a lot more processing power than Amigas or Atari ST, and Javascript run hundreds of times faster than 68000 machine language. An original AMOS application where the speed of the main loop is not stabilized by Wait or Wait Vbl instruction would run much too fast to be useable on a modern machine.

  This property is here to indicate two things: the hardware emulation mode (classic or AGA) and the percentage of slow-down to apply to the basic code, so that the speed matches the speed of the machine it was originally programmed on. The different values are:

  - 500 : non AGA, speed as an Amiga 500
  - 600 : AGA, speed as an Amiga 600
  - 1200 : AGA, 1200 speed
  - 3000 : non AGA, 68030 speed

    - 4000 : non AGA, 68040 speed
    - PC : full HD, true colors, no speed limit.
- usePalette

  Modern machine use true color graphics whereas the Amiga and Atari computer used palette based colors. In a palette based system, each pixel contains a number indicating an entry in the system palette which contains the color to display. True color pixels directly contain the value of the color.

  On a programmer point of view, it changes everything, and must be indicated prior to compilation. This property can take two values:

    - true : AOZ screen will use a palette as on the Amiga. Screens will have 16, 32, or 256 colors depending on the emulation. Ink, Pen and Paper commands will ask for a number from 0 to the number of color of the screen.
    - false : AOZ will be in 'true color' mode, screen without palette, ink, Paper and Pen asking for true color definition in the form $RRGGBB.
- useShortColors

  On the Amiga (and on the ST), colors were coded on 12 bits, resuting in a hexadecimal representation with 3 digits : $RGB . Modern machine can display milliions of colors, and therefore need 24 bits to define one color : $RRGGBB. This property can take two values:

    - true : colors will be defined on 12 bits, $RGB. Use this value to be compatible with original AMOS sources
    - false : colors will be coded on 24 bits, $RRGGBB. Use this value when you create a new application.
- showCopperBlack

  On the Amiga, the copper chip was reponsible of the incredible display features of the machine. But the design of the chip also induced some artfacts known as 'black lines'. A black line was visible above every AMOS screen, and the left and right portions of the display near a screen were masking the screen behind them.

  AOZ provides a way to emulate these limitations and somehow reproduce the flaws of the hardware. This property can take two values:

    - true : AOZ will emulate the display as best as possible, one or two black scan line will be visible above each screen, and the left and right portions of the display of screens will be black
    - false : no emulation, screens are independant and have no masking effect
- unlimitedScreens

  Due to hardware limitations, the original AMOS only allowed the user to open 8 screens at the same time. This property can take two values:

    - true : only 8 screens allowed, an Illegal Function Call error will be generated if you try to open new ones.
    - false : no limit in the number of screens.
- unlimitedWindows

  The same kind of limitation was applicable to text windows. This property can take two values:

    - true : only 16 text windows can be opened at the same time, and can only be positionned at a horizontal coordinate multiple of 16
    - false : no limitation of any kind
- maskHarwareCoordinates

The hardware of the Amiga imposed to the programmer some limitation in the numbers. For example, a screen could only be positionned horizontally at multiple of 16 pixels. This property can take two values:

- true : same limitations as on the Amiga
- false : no hardware limitations, position the screens to the pixel

The display setup

This section of the manifest defines how the application is displayed in the browser.

```
// The display setup
display:
{
    width: 1024,
    height: 720,
    background: "color",
    backgroundColor: "#000000",
    scaleX: 3,
    scaleY: 3,
    screenScaleX: 2,
    screenScaleY: 2,
    fps: true,
    fpsFont: "12px Verdana",
    fpsColor: "#FFFF00",
    fpsX: 10,
    fpsY: 16
},
```

- width
- height

  These values contain the width and height of the canvas element defined in the index.html file produced by the compiler. Values are in pixels.

- background
- backgroundColor

  Indicates how to clear the canvas before rendering the AMOS application display. It can take the following values:

  - "color" : wipes the canvas with a specific color, defined in the "backgroundColor" property
  - "transparent" : wipes the canvas with transparency, only the parts generated by the AMOS application will be opaque

- scaleX
- scaleY

  The resolution of the Amiga computer was dependent on the technology of the time, and Full HD was far from being invented. An Amiga screen had the resolution of a normal TV, approximatively 512 x 450 pixels.
  Such resolution would appear very small on today's monitor, and in order to be displayed at a reasonable size on Full HD or even 4K monitors, AOZ has to zoom the display of the AMOS application.

  The scaleX and scaleY property contain the horizontal and vertical scale factor to apply when the application starts. Value of 3 provide a big enough display.

- screenScaleX
- screenScaleY

  The display of the Amiga begin so small for today's machine, drawing artefact due to Javascript anti-aliasing procedures can appear for small screens like lowres screens in 320x200.

  To counteract this problem, AOZ offers the possibility to internally use BIGGER screens. Artefacts are then less visible and the emulation is better.

  These properties must contain an integral multiplier and will not work for floating point numbers. A value of 2 for both X and Y scale provide a good compromise.

  Please note that this value only affects the precision of the display and have no effect at all on the program itself.
- fps
- fpsFont
- fpsColor
- fpsX
- fpsY

  For developers, AOZ provides the possibility to display a FPS indicator on top of the application's display. This counter shows in real time the number of frames per second (FPS) of the display of the application.

  A good game should aim at staying at a constant rate of 60 FPS, and this counter can help find the places where the program slows down.

  - fps : true-> displays the indicator, false-> no indicator
  - fpsFont : "12px Verdana" -> indicates the name of the font to use, HTML font definition format
  - fpsColor : "#FFFFFF" -> indicates the color to use to draw the indicator
  - fpsX : horizontal position of the indicator
  - fpsY : vertical position of the indicator

Sprites and bobs setup

The next set of properties is related to sprite and bobs.

```
// Sprites and bobs
sprites:
{
    collisionBoxed: false,
    collisionPrecision: 1,
    collisionAlphaThreshold: 1,
}
```

- collisionBoxed

  AOZ provides two ways to handle collision detection : unprecise and fast boxed mode, and precise and slower pixel mode. This property can take two values:

  - true : collisions will be detected from the box delimiting the image of each sprite or bob. Although very fast, this method is also very imprecise, specially if the sprite or bob is rotated. If set to true, the next properties have not effect.
  - false : collisions will be detected pixel by pixel. The next properties allow you to define the precision of the detection. Please note that in Chrome, pixel precise collision

detection does not work if the application is not resident on a server (you will get an Internal Error message).

- collisionPrecision

  To detect pixel per pixel collision, AOZ uses what is called a 'collision mask'. This mask is a simple representation of the shape of the image of the sprite or bob. This property allows you to define the scaling factor of the mask relative to the original image. Example of values:

  - 1 : the mask will have the same size as the original image. Use this value for small sprites or bobs. It is suggested to use this value for all emulation of Amiga applications.
  - 0.5 : the mask will be reduced by half. Collision detection will be faster, a little more imprecise but the global shape of the objects will be preserved
  - 0.25 : a smaller value should be used for very large sprites or bobs running in Full HD or 4K applications. This will not affect the detection, will reduce considerabley the memory consumption of the application, and will go 16 times faster.

- collisionAlphaThreshold

  AOZ creates the collision mask from the images stored in the sprites bank, by setting as 'colliding' every non-transparent pixels. Transparency is defined in PNG images as a value from 0 to 255, called the 'alpha' value, 0 being 'transparent' and 255 being 'totally opaque'. This property is a threshold telling AOZ at which level of alpha a pixel should be considered transparent or opaque. Every pixel ABOVE the threshold will be opaque, and transparent below.

  If you produce your sprites with modern tools like Photoshop or Paint.net, such tool leave an anti-aliasising area around the shape. The alpha in such area goes from opaque (close to the actual drawing) to transparent. It is a good idea to set te collisionAlphaThreshold to values like 20 for such images, making sure that the collision happens in the opaque area and not the anit-aliasing area.

Default display settings

The next section of the manifest defines the construction of the default display. The manifest you will find in the distribution emulates the original display of AMOS running on a PAL Amiga, but this can be changed to match your application.

```
// Default settings
default:
{
    // Definition of the default screen, here, the basic AMOS red screen
    screen:
    {
        x: 128,
        y: 50,
        width: 320,
        height: 200,
        numberOfColors: 16,
        pixelMode: "lowres",
        palette:
        [
            "#000000", "#AA4400", "#FFFFFF", "#000000", "#FF0000", "#00FF00",
"#0000FF", "#666666",
            "#555555", "#333333", "#773333", "#337733", "#777733", "#333377",
"#773377", "#337777",
            "#000000", "#EECC88", "#CC6600", "#EEAA00", "#2277FF", "#4499DD",
"#55AAEE", "#AADDFF",
```

```
            "#BBDDFF", "#CCEEFF", "#FFFFFF", "#440088", "#AA00EE", "#EE00EE",
"#EE0088", "#EEEEEE"
        ],

        // The text window in the back of the screen, where you can Print
        window:
        {
            x: 0,
            y: 0,
            font: "8px Arial",
            fontWidth: 8,
            fontHeight: 8,
            border: 0,
            paper: 1,
            pen: 2,
            background: "opaque"
        }
    }
}
```

Screen

This section contains the definition of the default AMOS screen.

AOZ emulates the display of the Amiga if the 'display/emulation' parameter is set to an Amiga machine. Hardware coordinates go from 0 to 512 horizontaly, and from 0 to 450 vertically.

- x
- y
  The coordinates of the default screen as you would enter them in a Screen Display command.
- width
- height
  The width and height of the default screen, as you would enter them in a Screen Open or Screen Display command
- numberOfColors
  The number of colors of the palette. Values can be:
  - 2
  - 4
  - 8
  - 16
  - 32
  - 64 -> the screen will be in half-bright mode
  - 128 -> if emulation is "1200"
  - 256 -> if emulation is "1200"
- pixelMode
  This property is equivalent to the last parameter of the Screen Open instruction, and can take the following values:
  - "lowres' : opens a lowres screen
  - "hires" : as on the Amiga, Hires screen have double information horizontally
  - "laced" : creates an "interlaced" screen, with double amount of pixel vertically. You can combine values, example : "hires|laced" will quadruple the number of pixels on the display.

- palette
  This property contains the default AMOS red screen palette, 32 colors defined in HTML format.

Text window

AMOS creates a default text window (number 0) for each new screen. This window is the place where the Print instruction takes place and covers the entire surface of the screen. The manifest contains the parameter of this window.

```
// The text window in the back of the screen, where you can Print
window:
{
    x: 0,
    y: 0,
    font: "8px Arial",
    fontWidth: 8,
    fontHeight: 8,
    border: 0,
    paper: 1,
    pen: 2,
    background: "opaque"
}
```

- x
- y
  The coordinates of the top left of the window, should usually be left at zero
- font
- fontWidth
- fontHeight
  The name of the font to use. This part of AMOS is still work in progress and is not used for the moment. Unpredictale results if you chaneg the size of the font! :)
- border
  The number of the border (out of the 16 available in AMOS) to use around the text window.
- paper
- pen
  The default values of Pen and Paper, pointing to the corresponding colors in the palette of the screen. Here Paper 1 is red (#FF0000) and Pen 1 is white (#FFFFFF)...
- background
  Work in progress, unused.