Retro Sprite Workshop ver1.4

User Manual

Application and Documentation was Created by TCFS

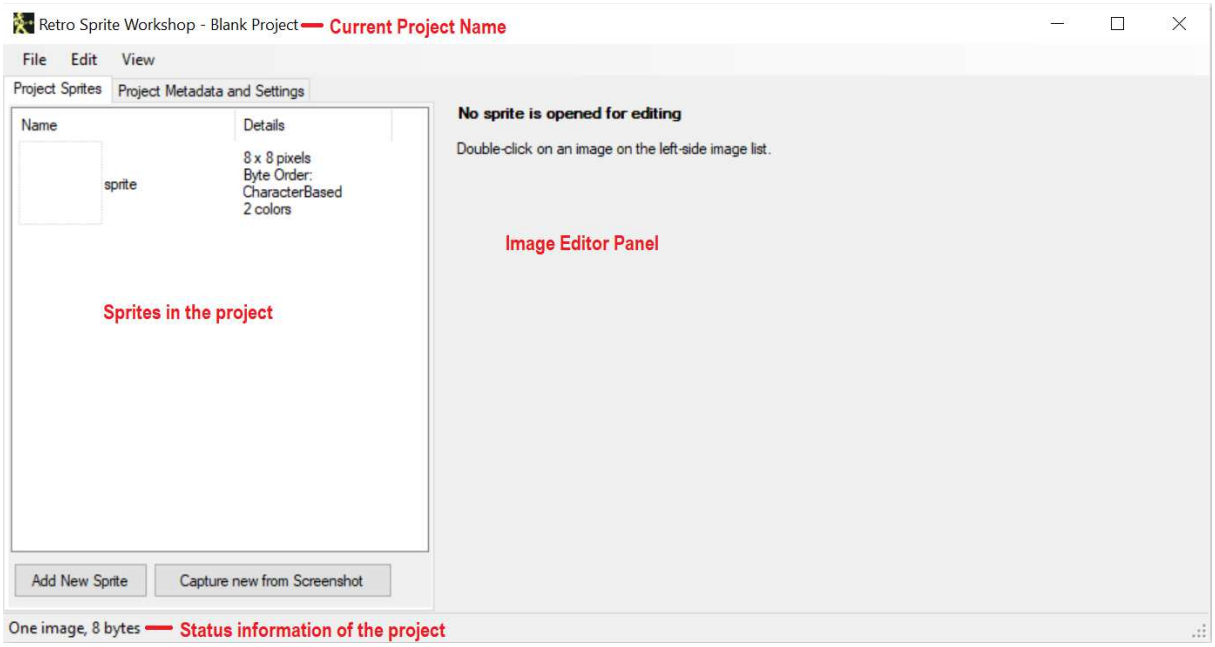
This application is a professional tool for retro game developers, especially for Commodore 64/128 or Commodore 264 series. This program produces output formats which can be used in external development tools, such as *Macro Assembler*.

The usage of this program assumes that the user is experienced in VIC and TED graphics, knows the difference between high resolution (1 bit per pixel, 2 colors) and multicolor (2 bits per pixel, 4 colors) and the pixel arrangement of the color modes.

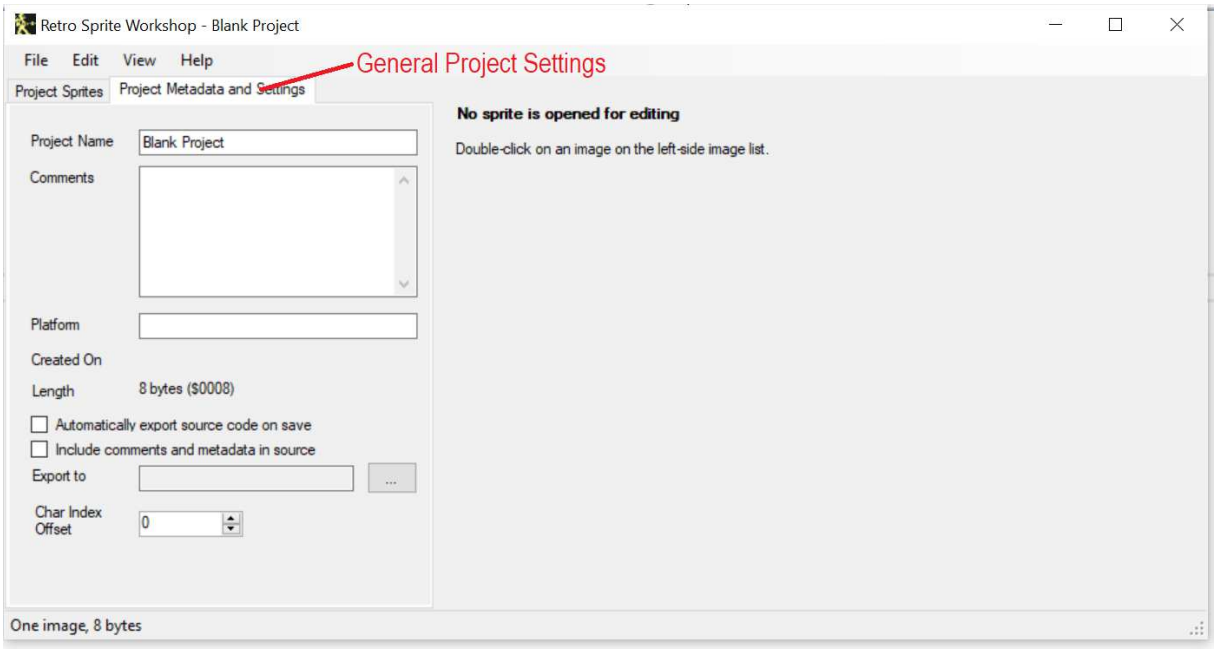
What you can see after start

The application starts with a default, empty project. Everytime when you want to create a new, empty project, just restart the application or click on **Create Project** in the **File** menu.

This image below shows the screen of an empty project.

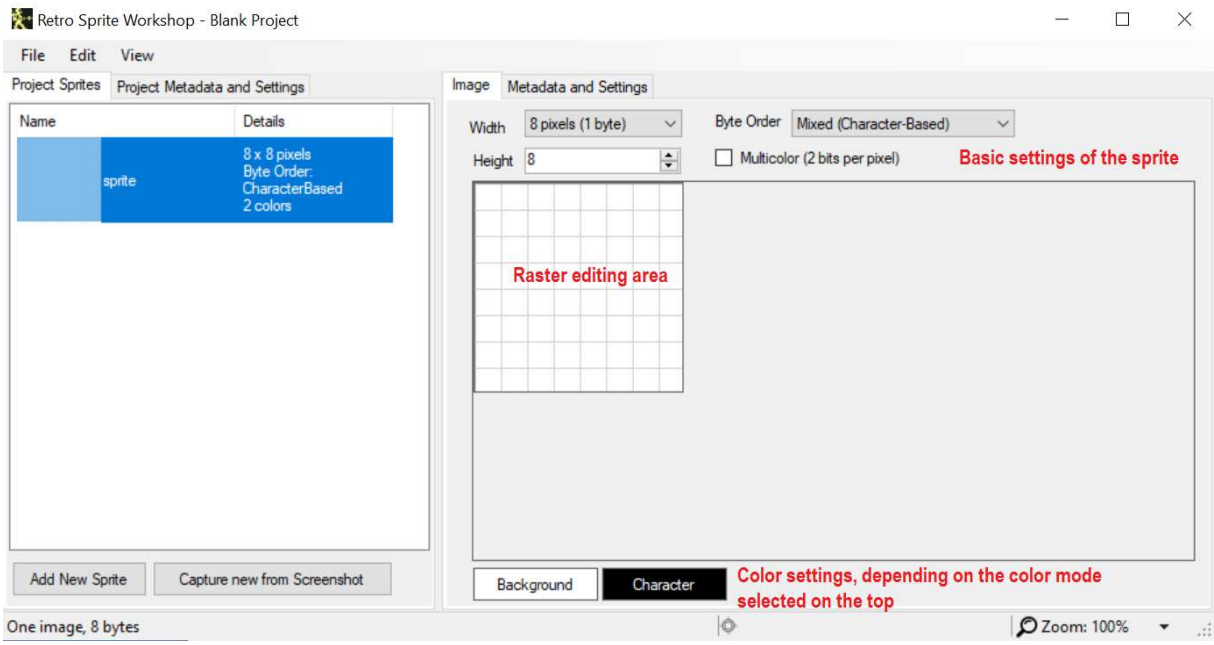


You can swith between the images and the project metadata page using the tabs on the upper left corner of the working area.



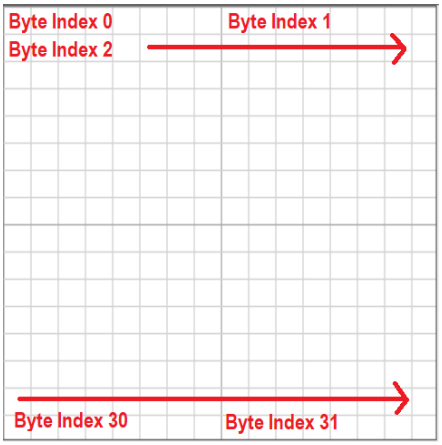
The blank project contains one sprite with no pixels. To open the sprite for editing, double click on its name on the left-side list.

On the top of the **Image Editing Panel**, the basic settings can be changed related to this sprite. The width and height is about the dimensions, the **Multicolor** switch changes between 2 colors and 4 color mode. The **Byte Order** selection controls the rendering of the byte stream in case of exporting the sprite for external development tool, for example *Macro Assembler*.



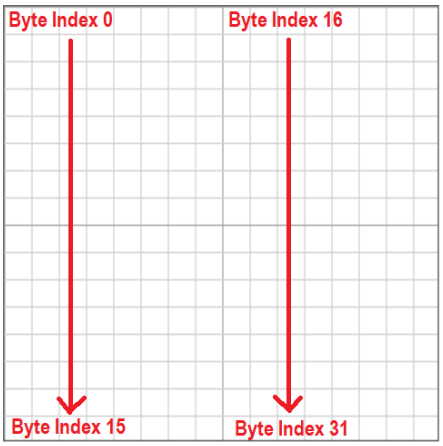
Understanding of the Byte Order Setting

Horizontal



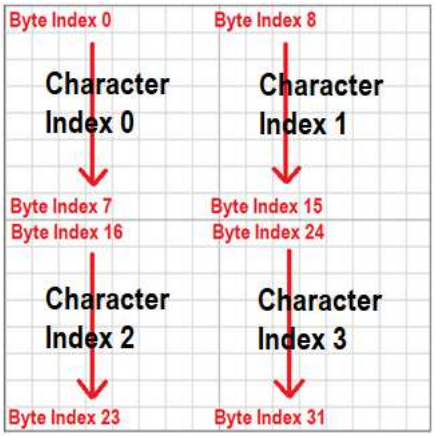
This setting is suitable for the Commodore 64/128 hardware sprites.

Vertical



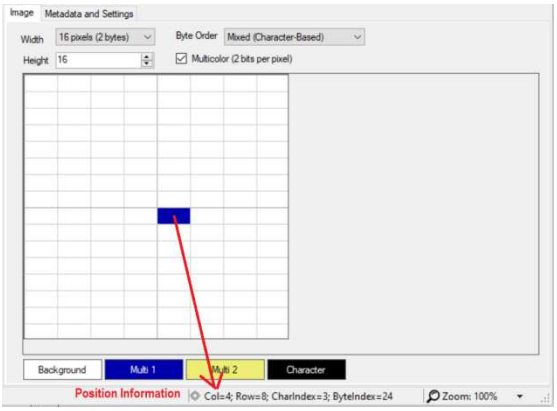
This arrangement is the optimal data structure to create images for software sprites.

Mixed (Character-Based)



Best selection for rendering images used for character screens.

Working with Byte and Character Indexes during editing



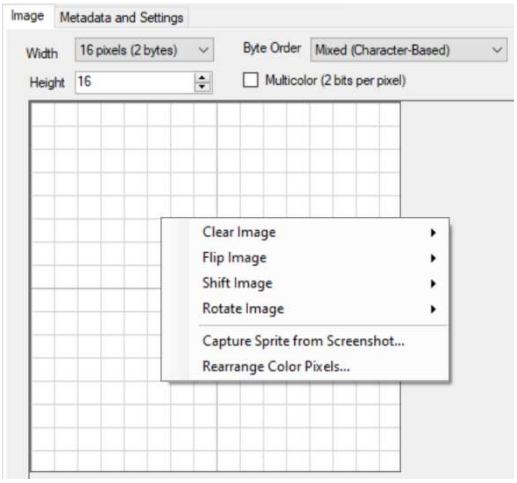
When you are moving the mouse over the Raster Editing Area, the status bar shows different information, based on the current **Byte Order** setting. The position information shows the x and y coordinates within the image and the Byte/Character indexes, where the zero index is always the top left corner of the image.

Drawing Sprites

Please note that this tool is not a professional image editing tool. Do not expect extended graphic functionality, such as shading, layering or masking.

The image area provides basic retouching functionality to put and remove pixels. Just click on the grid to cycle through the available colors.

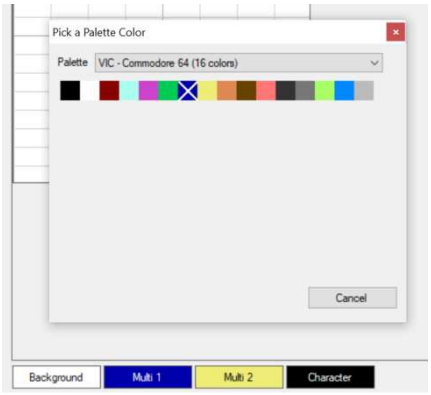
Right-clicking on the editing area will open a context menu with basic image manipulation possibilities, such as flipping/mirroring, shifting and rotating.



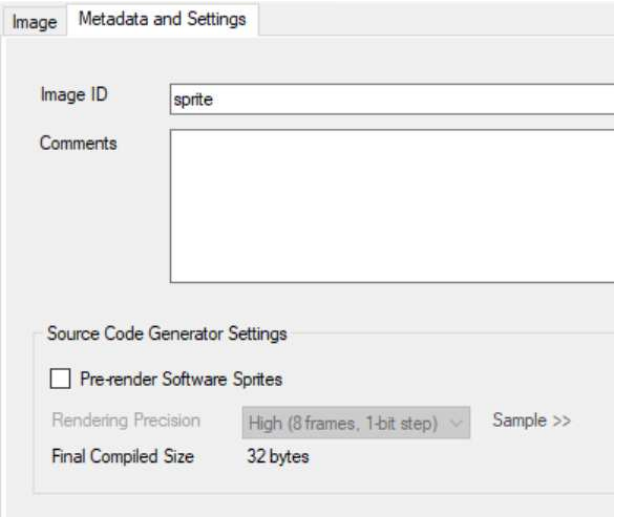
The **Rearrange Color Pixels** option is a common way to refactor your images when the color pixels should be swapped, for example exchanging *Multi1* (01) color bits with the *Character* (11) color bits. Rearranging the pixels is not for changing the colors of the pixels. That option physically rearranges the bit patterns of the image regarding to the settings.

Changing the rendering colors

Below the Raster Editing Area, two or four color buttons are visible, depending on the **Multicolor** setting of the sprite. Click on the appropriate button and the color selector will allow you to pick a color from a palette. Please select a platform-specific palette for best color match. This color selection will help you to visualize how your sprite will look like on the target platform, when the appropriate color registers are set.



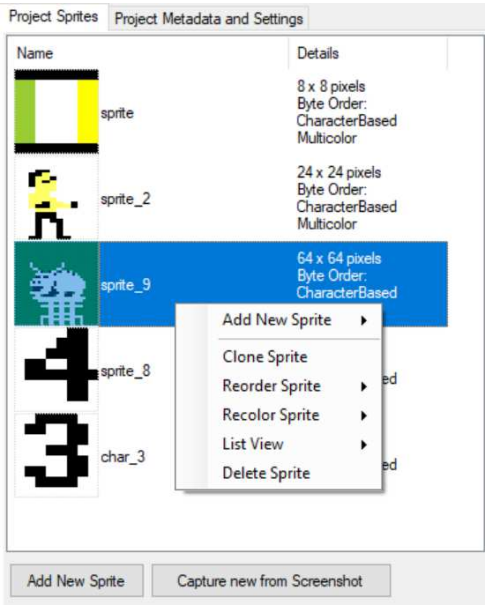
Sprite Medatada and Settings



The second tab of the **Image Editing Panel** is the metadata page which allows you to enter additional information about your sprite.

When the **Pre-render Software Sprites** option is turned on, the export functionality will generate the bitwise-shifted frames of the sprite with a given precision, configured on the **Rendering Precision** combo box. The **Final Compiled Size** data shows, how many bytes will the sprite occupy in the memory of the target platform. This value is dinamically recalculated when you require prerendering with different precisions. The **Sample >>** link will open a small animation showing the prerendered sample frame regarding to the selected color mode and precision. Please note that in case of **Multicolor** (2 bits per pixel) mode sprites, the prerendering precision will be halved due to color limitations, where 1-bit step is not available).

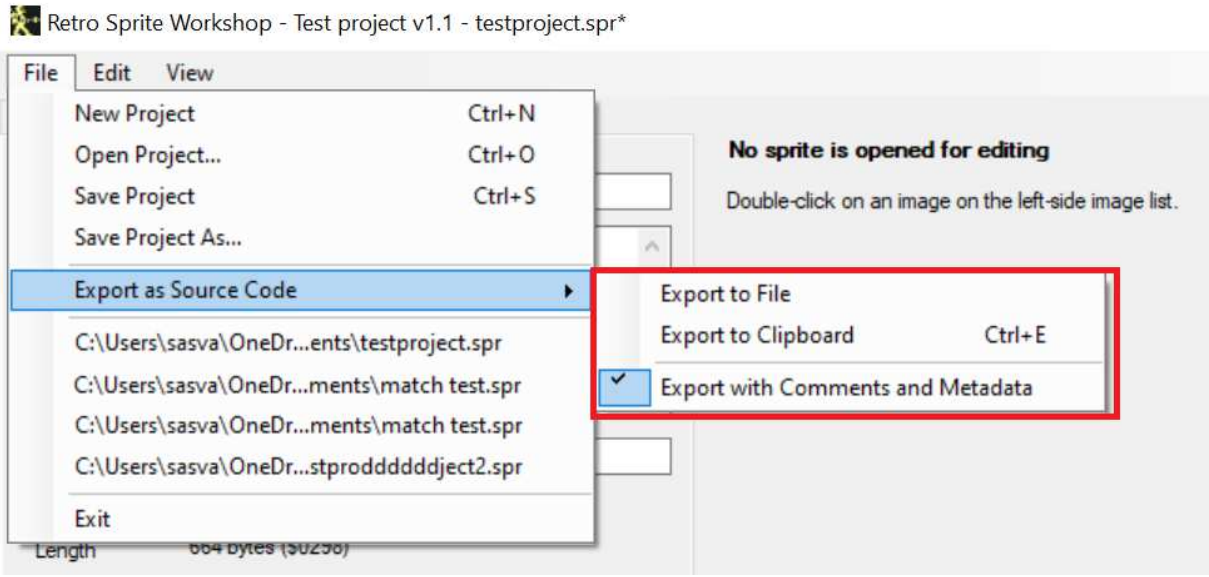
Manipulating the Project



Right-clicking on the project image list will allow to select from several options related to the project. These options can be found in the **Edit** menu of the top menu bar as well, when this image list is visible on the screen. Holding the Ctrl or Shift key while clicking will select multiple images at once. Some operations (**Delete**, **Reorder**, **Clone**, **Recolor**) will be applied on all selected images when multiple sprites are selected. Reordering the image list is important, because the final byte stream will be rendered from the first sprite to last in order, regarding the **Byte Order**ing of each individual sprite.

Interoperability with External Development Tools

Please open the **Export as Source Code** options from the **File** menu. The **Export to File** option will allow to browse a target file and gives the possibility to select the output format.



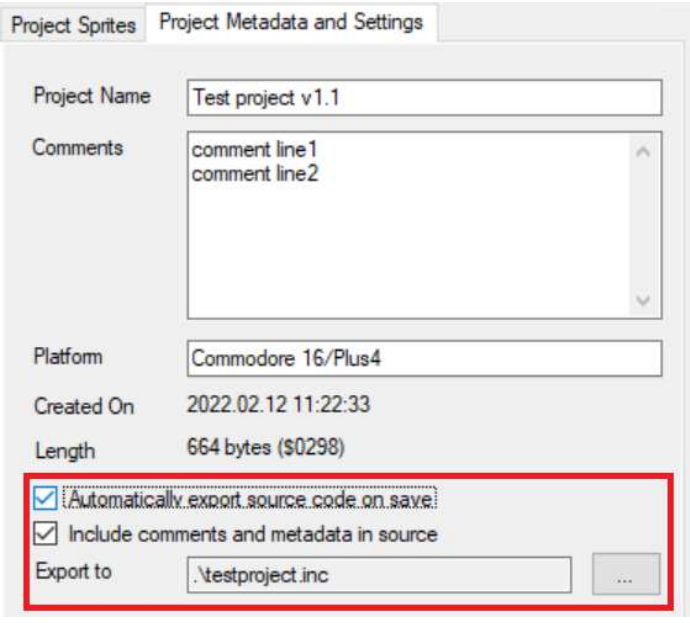
The .bas extension will export the data in *BASIC* language format with decimal numbers in DATA lines.

The .inc format is a *Macro Assembler* (or compatible) formatted file where the bytes and the sprite labels are rendered respectively.

When the **Export with Comments and Metadata** option is turned on, the output file will be more verbose, containing sprite comments from metadata.

Using the **Automatically export source code on save** option on the **Project Metadata and Settings** page, each time when the sprite project is saved, the preferred export file is also placed to the selected path. By default the export file is placed into the directory where the project file is opened from. Click on the … button next to the path will allow to specify the target directory and format.

The exported files can be processed using the `include` command, provided by the used assembler or BASIC compiler.



Compatibility settings

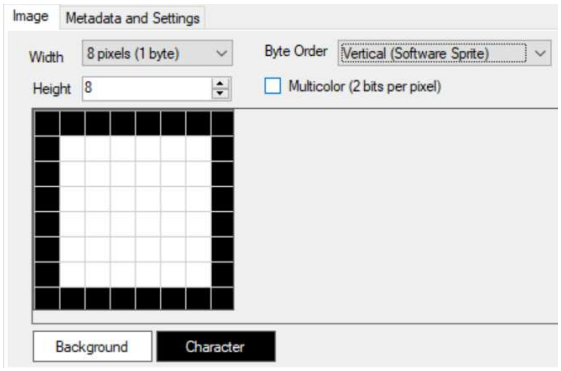
When your external assember/compiler requires different syntax as the code generator provides by default, go to the **Project Metadata and Settings** tab and click on **Advanced** button. This opens a dialog box where you can adjust the code generator settings. For example the line comments syntax can be changed from `;` to `//` when your compiler needs different notation.

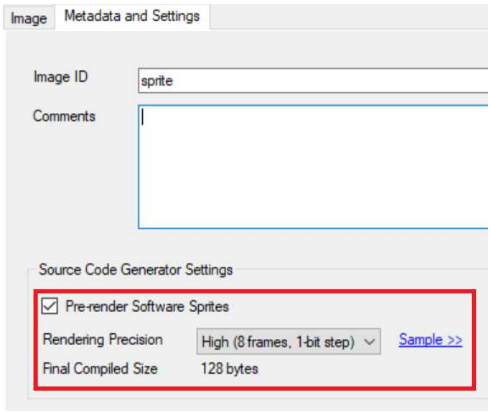
Example export of a sprite project in *Macro Assembler* format. (The **Export with Comments and Metadata** option was turned on.)

```
; ==================== Retro Sprite Workshop Project (c:\file
; Generated On:       2022. 02. 23. 22:27:45
; Project Name:       Test Project
; Project Comments:
; Target Platform:    Commodore 16/Plus4
; Project Created On: 2022.02.12 11:22:33

; ==================== Sprite 1 ========================
; Sprite ID:       sprite_38
; Sprite Comments: Captured from Screenshot (1.png)
;                  PosX=80; PosY=64
; Dimensions:      8 x 8 pixels
; Color Mode:      Multicolor (4 colors, 2 bits per pixel)
; Byte Order:      CharacterBased

sprite_38_WIDTH_PX    EQU 8
sprite_38_HEIGHT_PX   EQU 8
sprite_38_BIT_PER_PX  EQU 2
sprite_38_NUM_FRAMES  EQU 1
sprite_38:
          BYT $00, $00, $00, $00, $00, $00, $00, $00

; ==================== Sprite 2 ========================
; Sprite ID:       sprite_10
; Sprite Comments: Captured from Screenshot (1.png)
;                  PosX=144; PosY=112
; Dimensions:      8 x 8 pixels
; Color Mode:      Hi-Res (2 colors, 1 bit per pixel)
; Byte Order:      CharacterBased

sprite_10_WIDTH_PX    EQU 8
sprite_10_HEIGHT_PX   EQU 8
sprite_10_BIT_PER_PX  EQU 1
sprite_10_NUM_FRAMES  EQU 1
sprite_10:
          BYT $3f, $3f, $c0, $c0, $fc, $fc, $c0, $c0
```

Pre-rendered software sprites

Source sprite (8x8 pixels, 1 bit per color mode)





Exported output of the sprite above

```
; ==================== Retro Sprite Workshop Project  ======================
; Generated On:       2022. 02. 24. 15:51:55
; Project Name:       Blank Project
; Project Comments:
; Target Platform:
; Project Created On: 2022.02.12 11:22:33

; ==================== Sprite 1 ========================
; Sprite ID:       sprite
; Sprite Comments:
; Dimensions:      8 x 8 pixels
; Color Mode:      Hi-Res (2 colors, 1 bit per pixel)
; Byte Order:      Vertical
; Pre-rendering:   High8Frames

sprite_WIDTH_PX    EQU 8
sprite_HEIGHT_PX   EQU 8
sprite_BIT_PER_PX  EQU 1
sprite_NUM_FRAMES  EQU 8
sprite_frame1:
              BYT $ff, $81, $81, $81, $81, $81, $81, $ff, $00, $00, $00, $00, $00, $00, $00, $00

sprite_frame2:
              BYT $7f, $40, $40, $40, $40, $40, $40, $7f, $80, $80, $80, $80, $80, $80, $80, $80

sprite_frame3:
              BYT $3f, $20, $20, $20, $20, $20, $20, $3f, $c0, $40, $40, $40, $40, $40, $40, $c0

sprite_frame4:
              BYT $1f, $10, $10, $10, $10, $10, $10, $1f, $e0, $20, $20, $20, $20, $20, $20, $e0

sprite_frame5:
              BYT $0f, $08, $08, $08, $08, $08, $08, $0f, $f0, $10, $10, $10, $10, $10, $10, $f0

sprite_frame6:
              BYT $07, $04, $04, $04, $04, $04, $04, $07, $f8, $08, $08, $08, $08, $08, $08, $f8

sprite_frame7:
              BYT $03, $02, $02, $02, $02, $02, $02, $03, $fc, $04, $04, $04, $04, $04, $04, $fc

sprite_frame8:
              BYT $01, $01, $01, $01, $01, $01, $01, $01, $fe, $02, $02, $02, $02, $02, $02, $fe

; ==================== Retro Sprite Workshop Project END ======================
```

Please note that each individual rendered frame is labeled correctly for later reference from the code. Each frame contains a „bitwise shift right” version of the previous one. The original 8x8 pixels image became 16x8 (2 bytes width) with an additional column of bytes to make room for the shift operation.

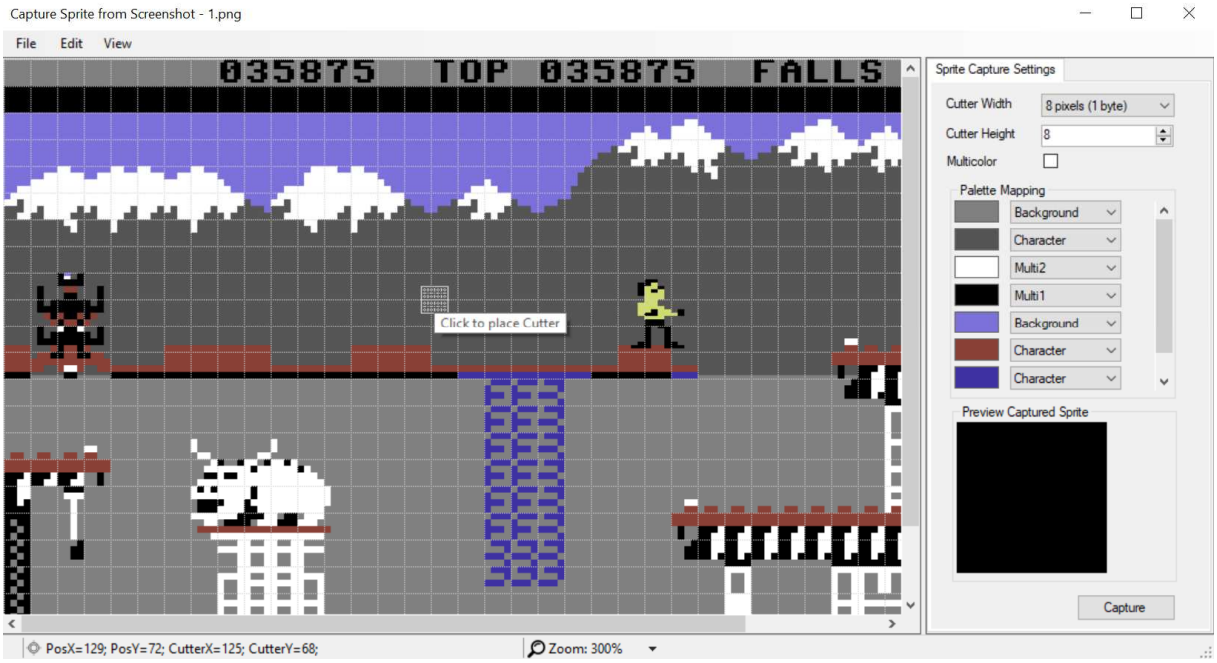
Capturing sprites from emulator screenshot

When you are porting existing program from one environment to another (for example from Commodore 64 to Commodore Plus/4) the capture tool is a powerful solution to grab the graphic elements. Instead of figuring out how the graphic elements are stored in the memory (checking emulator memory snapshots), you can capture the graphic elements right from the emulator screenshot.

When you get a standard 320x200 pixels image from an emulator screen (sample images can be found here: https://mozai.com/writing/brucemap/) Please make sure that the picture is not anti-aliased or filtered/compressed. The best effect is a pixel-to-pixel representation of the original screen in a lossless image.

To add new sprite to your project from a screenshot, click on **Capture new from Screenshot** button ont he project image page. (Or **Edit** menu -> **Add New Sprite** -> **Capture Sprite from Screenshot** command). The capture tool will appear and offer possibility to open an image from file or use the image currently available on the Clipboard.

When the proper image is loaded, the screen will look similar to this:

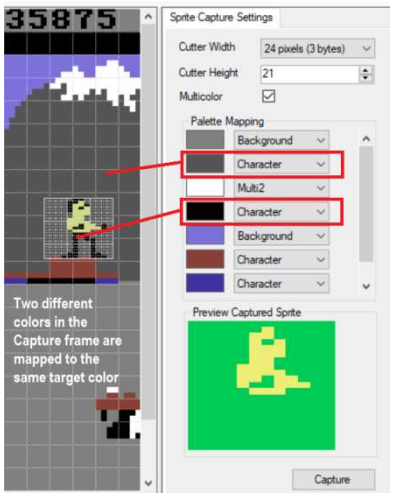


On the right side, the **Sprite Capture Settings** are available. Here you can specify what kind of image element you want to capture. The **Palette Mapping** list contains all colors available in the screenshot. One-by-one you should define the behavior of each colors of the screensot. The **Preview Captured Sprite** area is the final representation of your captured image.

After an image is loaded, you can drop the Capture Frame (Cutter) wherever you want ont he screen. Later, you can drag and drop the frame to a different location. After setting the size and the position of the capture frame, the preview are shows the final result. When necessary, adjust the **Palette Mapping** to match the colors correctly.

It is strongly recommended to zoom the screenshot to higher ratio (above 500%) to make fine positioning of the **Capture Frame** available.

Strange results after dropping the Capture Frame



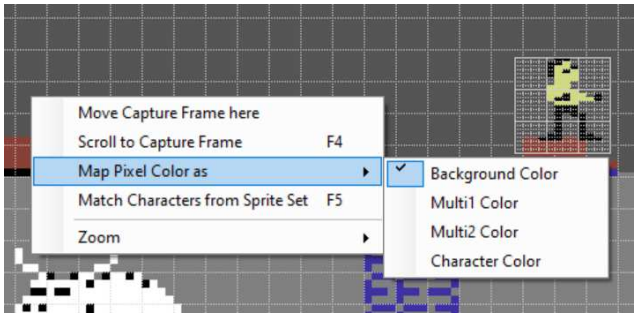
This is an example of the bad mapping.

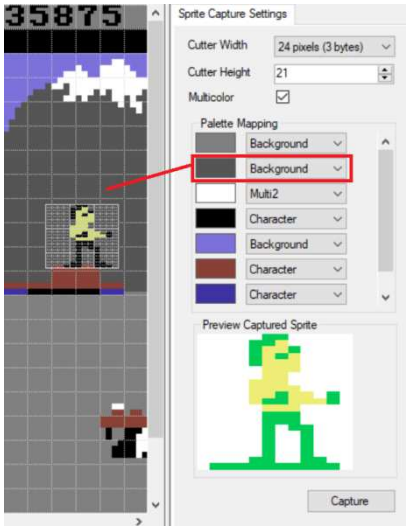
The most common reason for this kind of result is a wrong color mapping between screenshot image and target pixel roles.

Two different colors in the Capture frame are mapped to the same target color and the resulting sprite was generated incorrectly. The green area consists of the same color and we lost the possibility to distinguish the leg from the background.

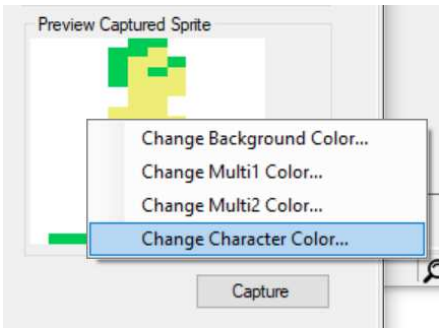
The role of the dark gray color is specified incorrectly. It is configured as character color, but definitely that color is a background color from the point of sprite's view. When that color is defined as Background, the preview image will be rendered again, showing the proper image.

When you are not sure where is the given color in the **Palette Mapping** list, you can map the color directly from the image. Carefully right-click on a pixel in the screenshot and select the **Map Pixel Color as** menu item from the context menu, then select the appropriate role.



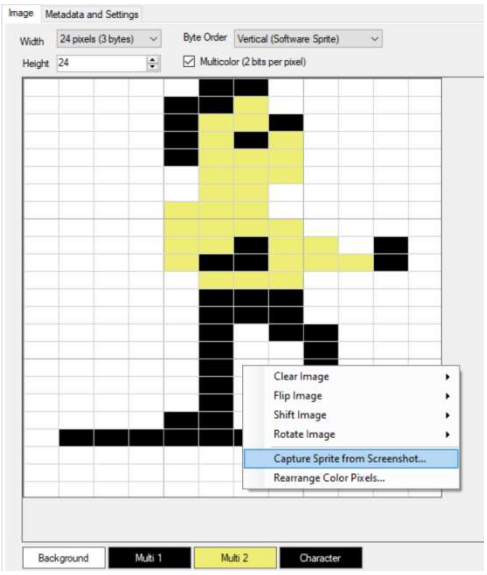


Ok, the sprite looks better now, but the colors are not matching in the captured image. The reason is that the image was rendered correcly and the pixels are set regarding to their real roles, but the sprite colors should also be set tomake a lifelike rendering. Right click on the preview area and set the character color accordingly.



When you pick a black color, the sprite will look similar to the screenshot. Keep in mind that the sprites are containing „palette color indices" from 0-3 and their real colors are provided by the proper TED or VIC registers.

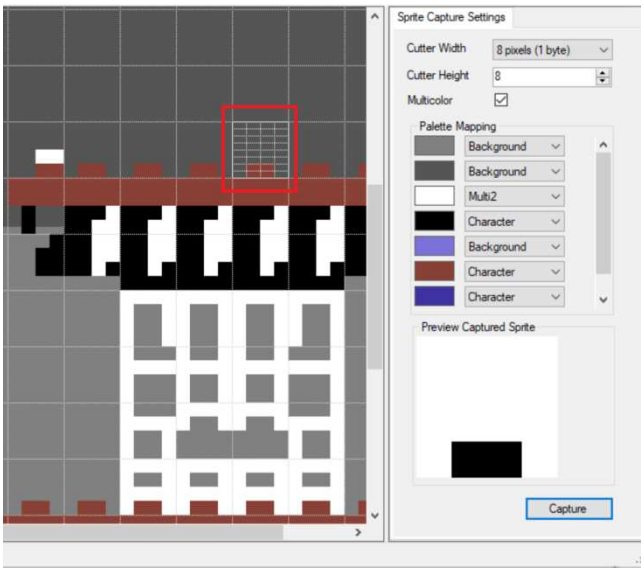
When the preview looks correct, click on the **Capture** button. The Capture window will close and you will get back to the sprite editor to retouch and finalize the sprite when necessary. When you need to capture a new sprite, the Capture tool will appear with the same image and settings loaded.



You can initiate capturing from the editing window of an existing sprite. Using the popup menu from the Raster Editing Area, the Capture tool will be opened using preset size and color mode settings from the current sprite.

Supporting character mode screen and Building Character Set

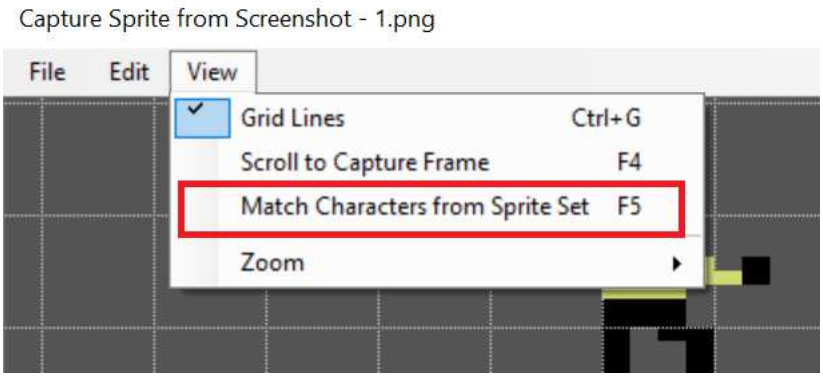
The Capture tool offers functionality to build Character Sets and build character screens using the loaded screenshot.



For building a Character Set , define a 8x8 pixels **Capture Frame** size (VIC and TED supports 8x8 pixels characters) and capture an image exactly from a character boundary (The grid lines can be turned on and aff using *Ctrl+G* or using the **View** menu **Gridlines** command.) Please be careful, characters captured off the character boundaries will prevent the function working properly. When this sprite is captured and avaiable in the project image list, the Capture tool is able to match the presence of the existing 8x8 sprites from your sprite collection.

To display matched characters, go to the **View** menu and click on **Match Characters from Sprite Set** command or press the *F5* key.

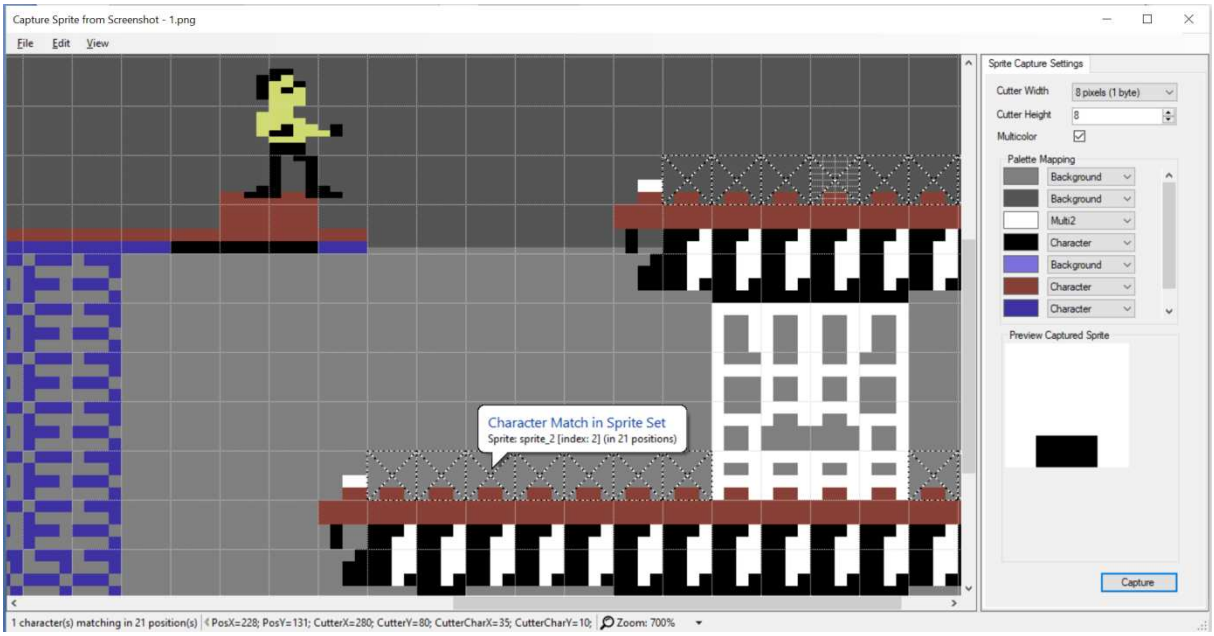
Warning: Using this calculation-intensive functionality might degrade performance on slower computers.



When the matching against existing sprites feature is turned on, the screen will mark the matched items with a crossed character section. The status bar on the lower left corner summarizes the number of matched characters and the positions for statistical purposes. When you hover the mouse over a marked character, a popup bubble will display which sprite is found on that position and how often that character is used on the screen. Please keep in mind that this tool relies on the current **Palette Mapping** settings. When the palette mapping colors are set incorrectly, characters cannot be matched (or not all characters are available for matching).

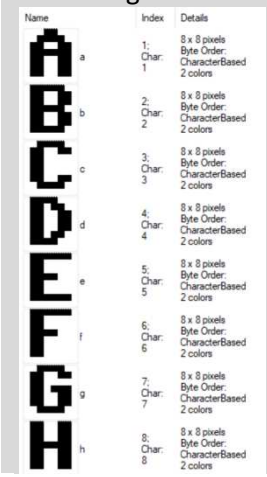
The „**[index: xx]**” indicator in the bubble shows the index of the matched sprite. It is recommended to not mix bigger images within the 8x8 sprites in sequence. When your project sprite set starts with

the characters and later the bigger sprites, the character matching will provide you proper information about the character codes found. When your character set finally assembled to the begin of a character set on the target platform, the index 0 means the @ character, the index 1 is for letter A and so on.



Recommended practice to reference character codes in source code

When you are using this project to define a character set, containing readable letters



…instead of referencing hardcoded character codes like

```
game_over_text
        BYT $07 ;letter g
        BYT $01 ;letter a
        BYT $0d ;letter m
        BYT $05 ;letter e
        BYT $20 ;space
        BYT $0f ;letter o
        BYT $16 ;letter v
        BYT $05 ;letter e
        BYT $52 ;letter r
```

...use dynamic references like

```
game_over_text
        BYT g_char_index
        BYT a_char_index
        BYT m_char_index
        BYT e_char_index
        BYT space_char_index
        BYT o_char_index
        BYT v_char_index
        BYT e_char_index
        BYT r_char_index
```

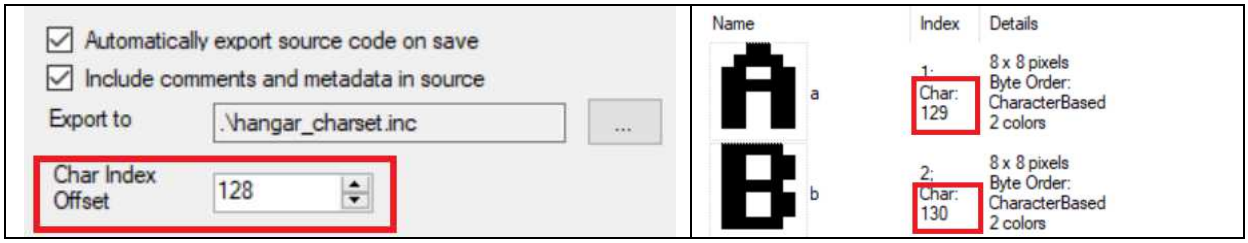
The ***_char_index*** suffix for the sprite name equals to the **Char:** value from the Index column of the image list. When you dynamically relocates the characters in your set (for example adding new characters before the letter „A”) your code will reference the proper character code during compilation.

In case you are combining multiple projects into one character set, the project included later into your code would restart the character indexing which brings negative effect to the character referencing.

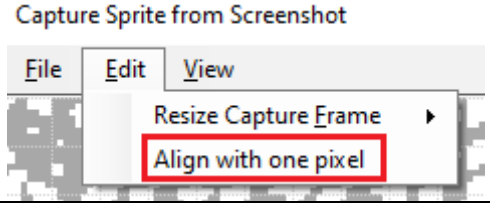
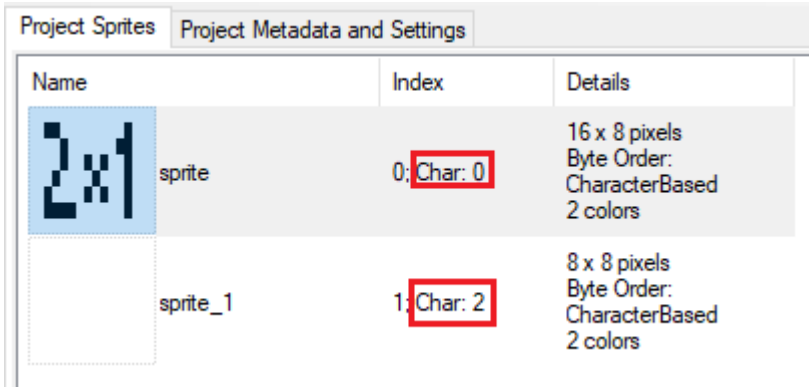
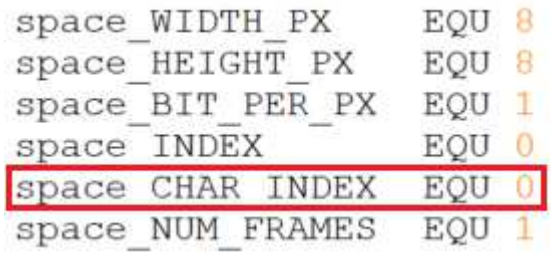
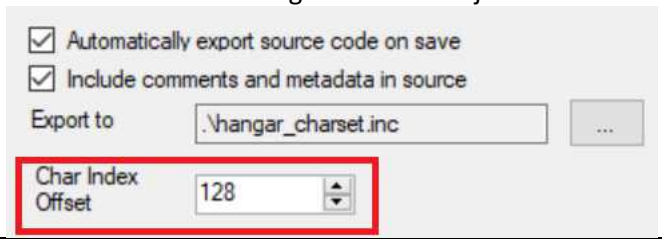
```
;lower chars
#INCLUDE "characters 0-127.inc"
;upper chars
#INCLUDE "characters 128-255.inc"

;wrong reference, char index will be misaligned
character_example
    BYT char_in_upper_charset_char_index
```

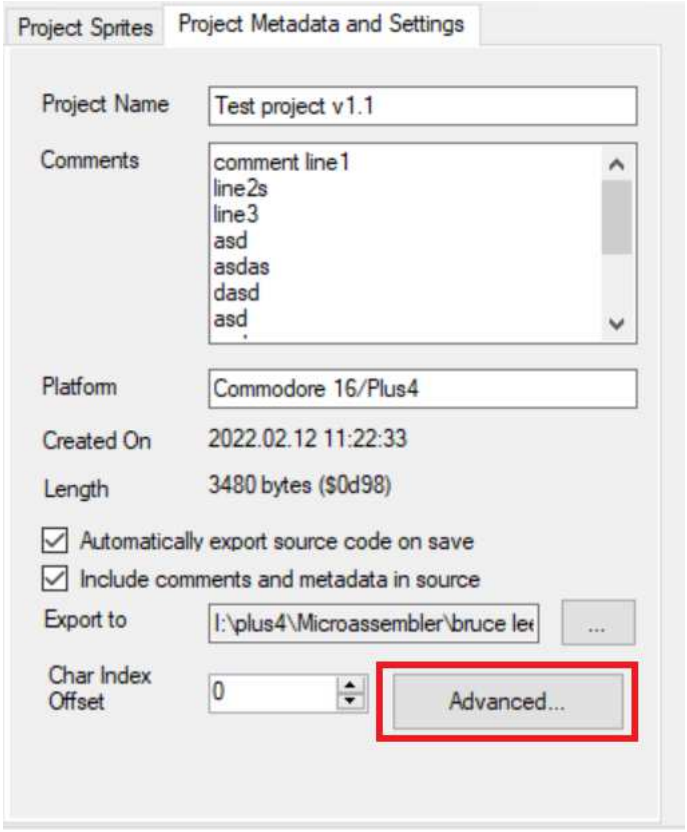
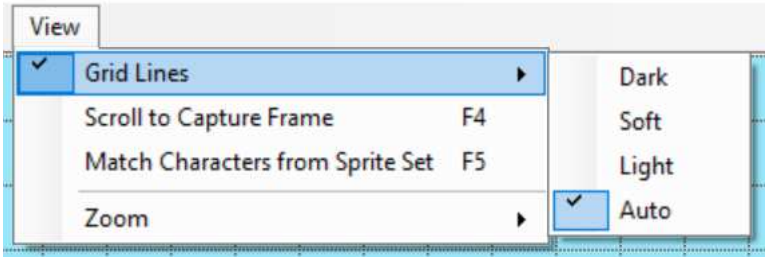
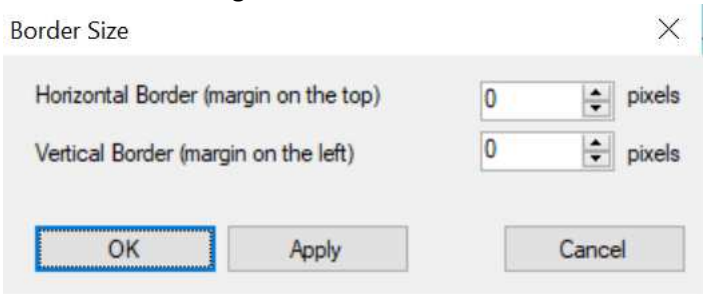
Due to the character index counter starts with 0 for each project, the upper chars part will start also with 0 by default. When you are referencing the first character from upper chars by ***_char_index***, you will get value 0 instead the expected 128. To avoid this, the „`characters 128-255.prj`” file should be configured to start character indexing with 128.

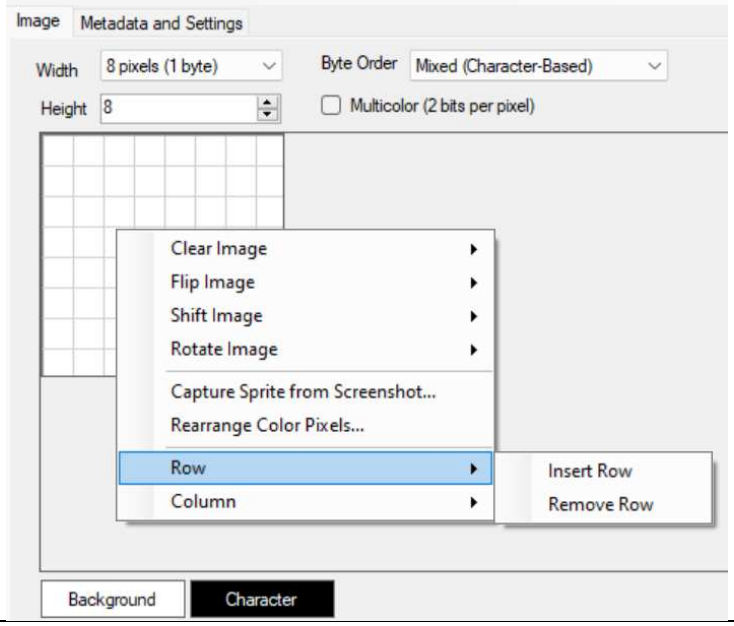
To do that, use the **Char Index Offset** settings (on the **Project Metadata** and **Settings** tab) to control the initial value of character index setting. This change will immediately recalculates the **Char:** value on the image list view.



Version History

ver0.9	Initial Release
ver0.9a	Added new menu item to **Sprite Capture** tool to make one-pixel horizontal alignment for supporting multicolor sprite cutting 
ver1.0	Added **Char Index** information for GUI and for exported file format. This shows the exact character code of the image within a character set.  This data also included in assembly export format. 
ver 1.1	Copy/Paste support for moving sprites from one project file to another
ver 1.2	**Char Index Offset** settings added to Project Metadata tab page 

ver 1.3	Code Generator compatibility settings on **Project Metadata** tab  Added **Grid** and **Cutter** color scheme selector for better visibility  Added **Border Size** setting to support screenshots opened directly from emulator containing borders 
ver 1.3	Added image manipulation commands to the Sprite Editor area context menu: Insert/Remove Row/Column



Retro Sprite Workshop application for Windows was developed by TCFS in 2022 for internal, advanced development purposes. The author holds no responsibility for any data- or hair loss caused by this product.

The above mentioned „*Macro Assembler*" is a product of Alfred Arnold, Stefan Hilse, Stephan Kanthak, Oliver Sellke and Vittorio De Tomasi