DATlib

Made in France.
Deal with it.

Rev 0.2

Table of contents

About DATlib

DATlib is a library designed for the NeoBitz/NeoDev environment.
It is designed to replace libvideo and libinput from the original kit.

Its goal is to provide easy functionality trough base elements (scroller, picture and animated sprite) which you are prone to use in your software, and allow better performance than basic libraries while writing less code.

Tools are also improved over standard ones to allow more colors (no longer limited to one palette per object), auto animation support, smaller data…

Combined tools and library allow easier coding while providing better performance and easy syncing between vblank and sprites update, greatly reducing tearing issues.

Concepts and preliminary notices

First of all, there is a quick demo program provided in the archive with source code, which you can explore and play along with to get familiar with the library and tools, or use as a stepping stone for your project.

DATlib will occupy about 10KB of the system ram.

The main outline of how this library works is that graphic updates are queued into buffers (also called draw lists / command queues) that are processed during vblank. There is currently three command queues: tiledata queue (SC1 buffer, VRAM 0x0000 – 0x6FFF operations), sprites control queue (SC234 buffer, VRAM 0x8000- 0x85FF operations) and palette jobs queue (palette ram operations). This means you can -for example- update a sprite position anywhere in your code, it will automatically be synced with and updated during next vblank.

Many components of this library evolve around the concept of base sprite and base palette:

Base sprite designates the starting sprite to use for said element. As an example a 4 tiles width picture with base sprite set to 10 will therefore use sprites #10 #11 #12 #13 to display.

Base palette is basically the same concept as base sprite, applied to color palettes.

It is currently up to the user to manage sprites and palettes to make sure no overlaps occurs across different elements.

Installation

Requirements: main NeoBitz dev package is required, make sure it is correctly installed and set up.

Copy the designated files from the archive to their destination directory.

File	Destination
lib/DATlib.h	from your NeoDev install directory, m68k/include/
lib/input.h	from your NeoDev install directory, m68k/include/ (backup previous version if you want to keep it for some reason)
lib/libDATlib.a	from your NeoDev install directory, m68k/lib/
bin/Animator.exe bin/BuildChar.exe bin/NeoTools.dll bin/CharSplit.exe bin/Framer.exe	Put those in a known path folder like m68k/bin/, or create a new directory and add it to your path variable.

DATlib is designed to supersede libvideo and libinput, make sure your remove those from your linker options in your project makefile and add DATlib library (remove `-lvideo` and `-linput`, add `-lDATlib`).

IE:

```
LIBS = -lvideo -linput -lprocess -lc –lgcc
```

becomes

```
LIBS = -lDATlib -lprocess -lc -lgcc
```

Add `<input.h>` and `<DATlib.h>` in your program includes.
If you use BuildChar to convert your data into tilemaps (most likely you will), also add “externs.h” to your project.

Use the provided `common_crt0_cart.s` and `crt0_cart.s` file for your project, replacing older ones. (`common_crt0_cd.s` and `crt0_cd.s` for CD projects).

Features

Input management

As standard `libinput` is dropped when installing DATlib, input defines are provided in the new `input.h` include file. Values can be read with the `volMEMBYTE()` macro.

Support is provided for mahjong controllers as well as 4P adapters, check library reference section for available defines.

Example:

```
#include <input.h>


BYTE p1;

p1=volMEMBYTE(P1_CURRENT);      /* get current status of P1 */

if(p1&JOY_A) {                  /* button A pressed ? */
      ...
}
```

Program loop

Using the library requires using a defined program flow for everything to work together.

While there are many ways to arrange code and use functionalities, here is a sample, basic program loop:

```
initGfx(); //initialize library components

/* initialize scroller, pictures etc… */

SCClose(); //we done initializing
while(1) {
      wait_vblank(); // wait vblank
                     // screen updates will occur during vblank

      /* do stuff */

      SCClose(); //we done updating stuff
                 //loop for vblank sync and screen update
}
```

Provided graphics types

DATlib provides three base graphical elements that should fulfill most needs:

picture

- simple picture type
- allows display, positioning and flipping of static pictures
- when setting picture position, you are setting top left pixel position
- uses picture tile width sprites (ie: 64px width picture = 4 tiles width = 4 used sprites)

scroller

- type used to display a scrolling plane
- 8 way scrolling ability
- no map size limit
- uses 21 sprites, regardless of plane dimensions

animatedSprite

- provides support for animated sprites
- allow display, positioning, flipping and animating sprites
- animation system supports repeats and animation linking
- up to 65536 animations, unlimited animation steps
- used sprites depends on currently displayed frame, good practice is to plan enough sprites to fit the widest frame

Note: Animated sprites uses a different way to position themselves. Each frame location is relative to a fixed reference point. This is due to the nature of animations, often using a set of frames of different sizes and alignments (to avoid encasing a few pixels in a large picture frame, saving space and CPU time). Positioning operations on animated sprites refer to positioning the reference point. Flipping animated sprites is done around the reference point.

Vblank handlers

Vblank handlers are interrupt handlers provided by the library, required for proper operation. Those have to be set up as your vertical interrupt (IRQ2) vector.

DAT_vblank

Standard vblank handler.

Operation:

- sets job meter to red
- process tiledata queue
- process sprites control queue
- process palette jobs queue
- sets job meter to orange
- resets draw lists, updates frame counter
- checks and process debug dips
- acknowledges IRQ, kicks watchdog, calls `SYSTEM_IO` (BIOS)
- sets job meter to green
- returns

Note: Job meter colors are only updated under select circumstances, see debug dips section.

DAT_vblankTI

Vblank handler with timer interrupt support.

Operation:

- Load base and reload timing values (if timer interrupt enabled)
- Branch to `DAT_vblank` for standard operations

Note: When using timer interrupts, requested LSPC mode must be written to the `LSPCmode` variable (WORD). This is due to the LSPC mode hardware register being manipulated to set timer values, therefore needing a reference value of requested settings to preserve them. If using standard vblank handler you can ignore the `LSPCmode` variable and write directly to the register.

Timer interrupt

Base functionality is provided for timer interrupts, allowing to change one or two VRAM value on every (or select set of) scanline.

To enable timer interrupt functionalities:

- set `DAT_vblankTI` as your vblank IRQ vector
- set `DAT_TIfunc` as your timer IRQ vector

Notes: make sure you set variable `TinextTable` (DWORD) to 0 before enabling IRQ when using timer interrupt. This is done in the default init code, but make sure to keep it if customizing files. Timer interrupt related code uses the USP register, make sure you code doesn't conflict.

Using timer interrupt:

To work with timer interrupt you need to prepare data in a WORD table, storing VRAM address and data combos.

Format for the data table is:

- VRAM address n (1 WORD)
- VRAM data n (1 WORD)
- VRAM address n+1 (1 WORD)
- VRAM data n+1 (1 WORD)
- (etc…)
- end marker (2 WORD, 0x0000 0x0000)

For correct behavior it is required to use two alternating tables. One table for currently displaying frame, another one to prepare data for next frame.

Timer IRQ function must be set up with `loadTIirq()` prior to use.

Timer IRQ is available for single and dual data writes for each triggering. See `loadTIirq()` section.

Startup timer interrupt:

- set base and reload timers
- put pointer to data table for next frame in the `TinextTable` variable

Stop timer interrupt:

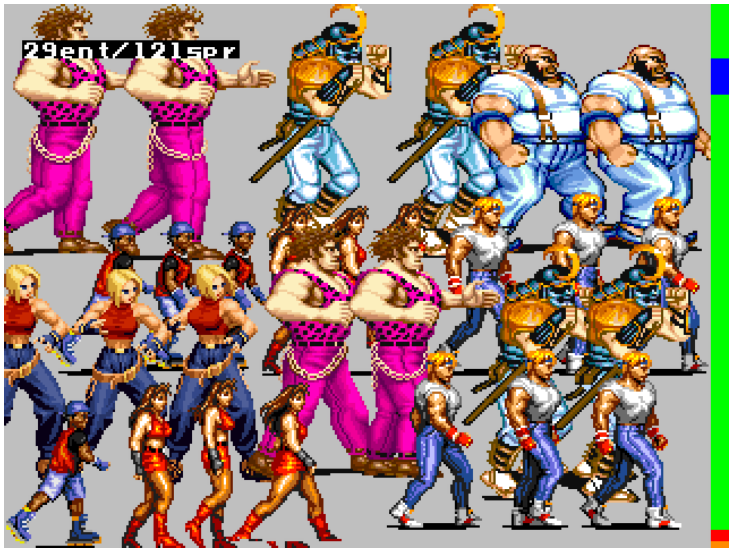
- set `TInextTable` to null (0)

Notes: When data last value is processed, the timer interrupt will be disabled for the rest of the frame until next vblank. This avoids triggering unnecessary IRQ, as they are CPU consuming. Default timer values are provided for first raster line triggering and each line repeat: `TI_ZERO` and `TI_RELOAD`. Timer interrupt will be disabled if TinextTable is null. Timer interrupt will be disabled if first table entry is end marker.

Job meter

Base job meter support is provided by the library.

Job meter allows basic profiling of your code, by having a visual representation of how much CPU time is used. Using different colors lets you observe CPU usage of every procedure, allowing targeting of things to optimize.



Job meter example:

Green color: free CPU time
Blue color: animation procedures
Red color: vblank sprites updates
Orange color: post vblank `SYSTEM_IO`

Note: Setting job meter colors during active display will issue a pixel of said color on screen (on real hardware). This is an issue with the hardware that can't be avoided, therefore make sure to use job meter in debug builds only.

Debug dips

Some of DATlib features are enabled through debug dips. Enable dev mode into bios then set the requested dips to 1.

- debug dip 2-1
  Enable vblank job meter color updates.
  Vblank interrupt will color draw queues as red job, and post jobs, like `SYSTEM_IO`, in orange.

- Debug dip 2-2
  Displays current raster line # when draw queues are done being processed.

- Debug dip 2-3
  Displays a rough usage meter for SC1 and SC234 buffers

- Debug dip 2-4 ~2-8
  Unused / reserved future use.

Sprite Pools

Sprite pools are an alternate way to handle sprites rendering. It consists of a reserved sprites batch which is then used to display assets.

It differs for the classic, “allocated” draw mode by many ways:

- Sprite tilemap/position data is written during active frame, alleviating vblank load
- Sprite tilemap/position data is fully rewritten every frame
- Removes the need to manage baseSprite from assets, they are drawn in the order they are submitted
- Submit order drawing allows for easier sprites sorting/priority change.
- No baseSprite management means less sprite loss, when current frame is smaller than the reserved space

Base operation sketch

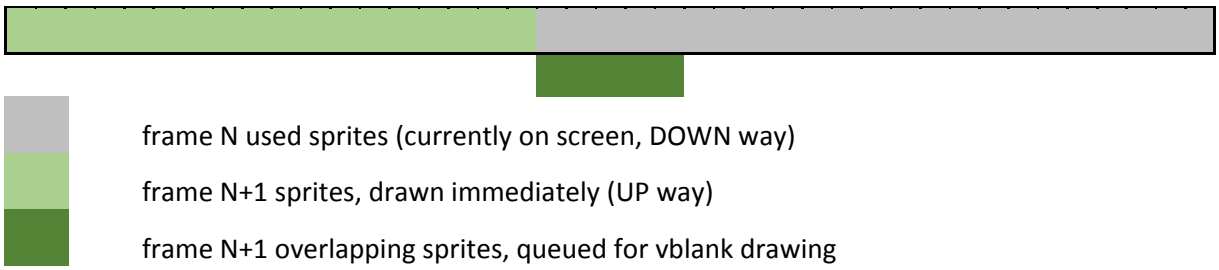
A spritePool entity must be initialized providing a pool size (# of sprites) and a starting position for this pool (baseSprite). Pool size should be aimed at twice the size of an average scene. If an average frame requires 80 sprites, allocate 160.

To draw into the sprite pool, user must submit an array of pointers to aSprite entities, followed by a null pointer end marker.

Drawing in the sprite pool alternates way every frame (`WAY_UP/WAY_DOWN`). When going UP, pool uses sprites from pool start toward pool end, when going DOWN, from pool end toward pool start. User must supply the top or bottom end of the pointer array, to fit needs.

Tilemap and X position data is written into vram during active display, Y position is updated during vblank.

In case of heavy load, it is possible the sprite needs overlaps with the currently used sprites from previous frame. In this case overlapping sprite needs are queued for update during next vblank:



This provides a failsafe and user transparent operation in most scenarios, only way to currupt visuals would be for a single frame to exceed the total size of the sprite pool.

Note: As **s**prite pools are designed to update VRAM during active frame, this feature isn’t interrupt safe (using it alongside timer innterrupt can corrup VRAM info).

Tools

Buildchar

Command line tool used to convert your graphics elements into tiles, tilemaps and palettes.

Input

- `chardata.xml`

  Contains description of assets to include into tile data.

  Example `chardata.xml` file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<chardata>
   <setup>
         <starting_tile fillmode="dummy">256</starting_tile>
   </setup>

   <scrl id="ffbg_b">
         <file>gfx\ffbg_b0.png</file>
         <auto1>gfx\ffbg_b1.png</auto1>
         <auto2>gfx\ffbg_b2.png</auto2>
         <auto3>gfx\ffbg_b3.png</auto3>
   </scrl>

   <pict id="ffbg_c">
         <file>gfx\ffbg_c.png</file>
         <flips>xyz</flips>
   </pict>

   <sprt id="bmary_spr">
         <file>gfx\bmary.png</file>
         <flips>xyz</flips>
         <frame>0,0:4,7</frame>
         <frame>4,0:4,7</frame>
         <frame>8,0:4,7</frame>
         <frame>12,0:4,7</frame>
         <frame>16,0:4,7</frame>
         <frame>20,0:4,7</frame>
         <frame>24,0:4,7</frame>
         <frame>28,0:4,7</frame>
         <frame>32,0:4,7</frame>
         <frame>36,0:4,7</frame>
         <frame>40,0:4,7</frame>
         <frame>44,0:4,7</frame>
   </sprt>
</chardata>
```

Nodes details:

- `<setup>`
  Contains general settings:
  - `<starting_tile>`
    Defines starting tile # (decimal). Used to leave blank tiles at the beginning of the char.bin file, useful if you need room to fit things like a character font at the beginning of the tileset. Additional parameter fillmode (none/dummy) defines if skipped tiles are to be filled or not.
  - `<charfile>`
    Defines output character file name. Optional, defaults to “char.bin”.
  - `<mapfile>`
    Defines output tilemaps data file name. Optional, defaults to “maps.s”.
  - `<palfile>`
    Defines output palettes data file name. Optional, defaults to “palettes.s”.
  - `<incfile>`
    Defines output include file name. Optional, defaults to “externs.h”.
- `<scrl>`
  Used to declare a scroller
  - id (attribute)
    Literal name the scroller will be referenced by in C code.
  - `<file>`
    PNG file of the display area.
  - `<auto1> to <auto7>`
    Additional pictures when using auto animation features.
- `<pict>`
  Used to declare a picture
  - id (attribute)
    Literal name the picture will be referenced by in C code.
  - `<file>`
    PNG file of the picture.
  - `<flips>`
    Flip modes wanted for this picture (optional).
    X = horizontal flip
    Y = vertical flip
    Z = horizontal & vertical flip
- `<sprt>`
  Used to define an animated sprite
  - id (attribute)
    Literal name the animated sprite will be referenced by in C code.
  - `<file>`
    PNG file containing all animation frames.
  - `<flips>`
    Flip modes wanted for this animated sprite (optional).
    X = horizontal flip
    Y = vertical flip
    Z = horizontal & vertical flip
  - `<frame>`
    Defines a frame, format is: top,left coordinate:width,height
    Unit is tile (16px)
    See Framer tool section to easily set up frames

About input files format:
Picture files used in chardata.xml must be PNG format, 32bppArgb. Define transparency by pink color (#ff00ff), or simply use transparency. Size must be multiples of 16.

About colors:
There is no limits color wise, as long as each tile is transparency + 15 colors max, you can use pics with hundreds of colors.
If your file is rejected for using too many colors per tile, erroneous tiles will be shown in a reject.png file.

About ID:
Each declared entity will generate an extern C object named <id>, as well as a palettes object named <id>_Palettes.

Output

- `char.bin`
  Your tile data, linear binary output.
  Convert to cart or CD format if needed by using the CharSplit tool.

- `maps.s`
  Tilemaps data, add to makefile to compile and link into your project.

- `palettes.s`
  Palettes data, add to makefile to compile and link into your project.

- `externs.h`
  Extern definitions of your data. Include into your C program to use data.

Mixing auto4 and auto8 tiles

It is possible to mix up auto4 and auto8 tiles on the same file when using auto animation. To do so, use the supplied auto4 marker tile (auto4_tile.png) on your <auto4> file to designate an auto4 tile.

Tile distribution across mixed up files is as follow:

	<file>	<auto1>	<auto2>	<auto3>	<auto4>	<auto5>	<auto6>	<auto7>
Auto4	Tile #0	Tile #1	Tile #2	Tile #3	End marker	---- Ignored data ----		
Auto8	Tile #0	Tile #1	Tile #2	Tile #3	Tile #4	Tile #5	Tile #6	Tile #7

Charsplit

Command line tool used to convert raw character data issued by buildchar to either cart or CD format files.

Usage:

```
charSplit [input_file] <options> [output_file_prefix]

Options:
        -cart   Ouput to cart format ([output_file_prefix].c1 & .c2)
        -cd     Output to CD format ([output_file_prefix].cd)
```

Example:

```
charsplit char.bin –cart game
```

will convert split char.bin into game.c1 and game.c2 files for cart system use.

Framer

Tools used to delimit animated sprites frames.
Each animated must be assigned a set of frames before being processed by the buildchar tool.

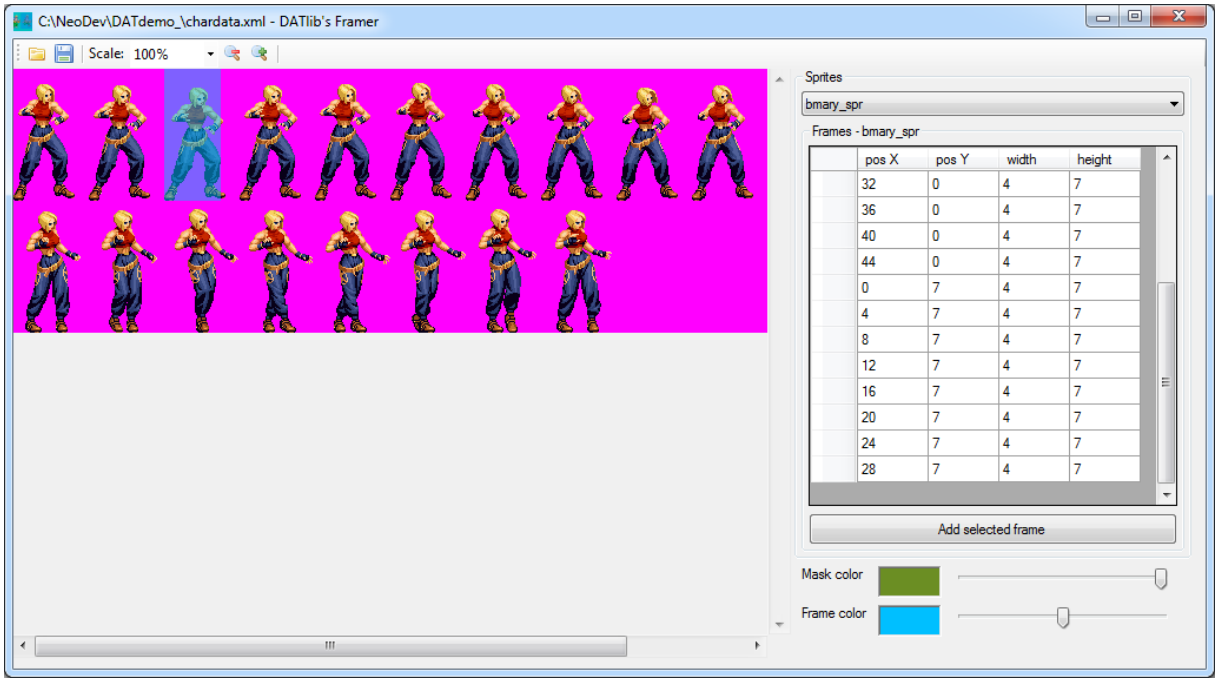
Input

- `chardata.xml`
  Click the open button and select the xml file containing reference to your animated sprites assets.

Output

- `chardata.xml`
  Click the save button to update xml file with the new/updated frames.

Usage:



Framer is very straightforward to use. Open your xml file, then select the animated sprite you want to work with from the drop down menu.
If the xml file already contains data, existing frames will be listed to be updated/removed.
To add a new frame, simply select it by clicking and dragging mouse, then click the add button, or press the space bar.
When done, click save to update the xml file, which is then ready to use with buildchar for processing.

Animator

Tool used to animate animated sprites.
Each animated sprite you process with the buildchar tool must be assigned at least one animation with the animator tool for proper compilation and linking of your project.

Input

When defining an animated sprite, buildchar will output a subfolder containing frames cutouts. Open this folder in Animator.

- `animdata.xml`
  This is your save file regarding this animation. If found inside folder, animator will load it.

Output

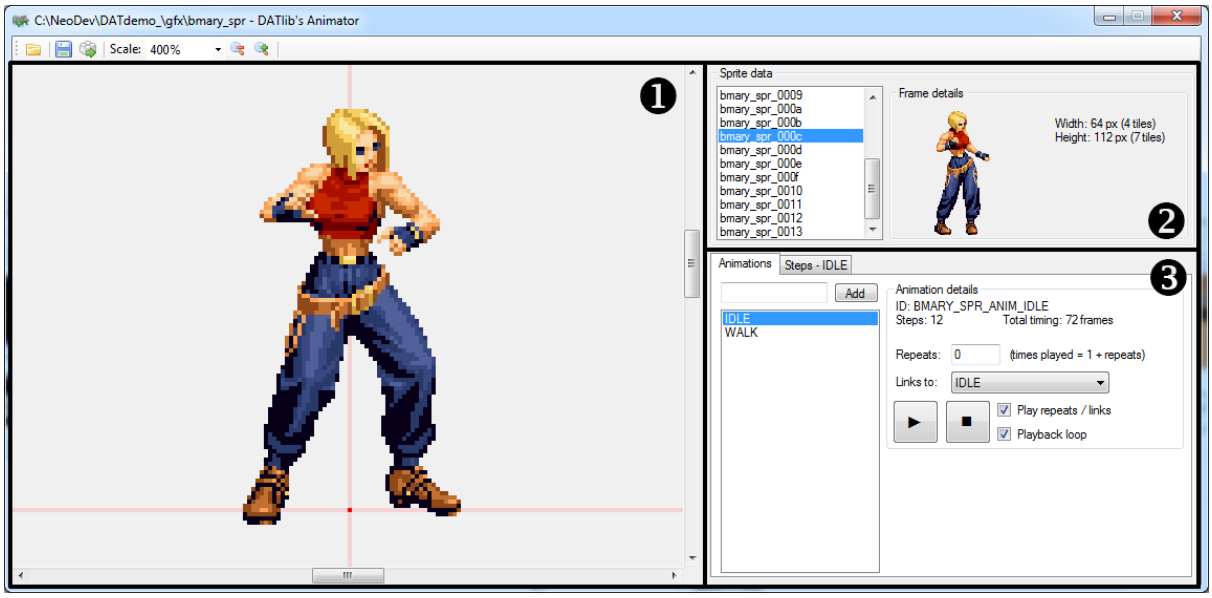
- `animdata.xml`
  This is your save file regarding this animation. Hit save button to save your work.

- `<id>_anims.s`
  Animations file, this should already be an include in your maps.s file by buildchar tool.

- `<id>.h`
  Contains animations C defines, should already be included in your externs.h by buildchar tool.

Using Animator:

Main window is divided into 3 areas



❶ Preview area

This area allows you to visually align frames and preview animations. Change scale for better viewing. Reference point is visualized by the intersection of the two red axes. When setting position of animated sprites in your code, you are setting the position of this reference point.

❷ Sprite and frames data area

This area will list and provide a quick preview of all the frames you created using the framer and buildchar tools, making sure you exported them correctly.

❸ Animations area

This is the main section to edit and check animations

Adding an animation: Input animation name in text field and press the [Add] button, the new animation will appear in the animations list.

Edition an animation: Select the animation you want to edit in the animations list. Input repeat count and link data for selected animation. Repeats are the number of times the animation will be repeated after initial play. Link allows you to branch to another animation once current animation is done displaying (including repeats). You can link an animation to itself to create a loop. If no link is selected the last animation frame will remain of screen after animation is done.

From there on, double click on frames in frames list to add animation steps.
You will have to input frame position for each step (X & Y field, or arrow buttons) as well as step timing (T field). Timing is the number of display frames the selected step remains on screen. Mod all steps checkbox will allow you to edit all steps at the same time.
You can adjust steps order or delete steps by using buttons under the steps list.

Animations IDs

Each animation created with the animator tool will generate a C define that can therefore be used when setting animations.

Format is `<id>_ANIM_<animation name>` (all uppercase).
As an example, building animations named WALK and IDLE for animated sprite defined with ID “mycharacter” will generate `MYCHARACTER_ANIM_WALK` and `MYCHARACTER_ANIM_IDLE` defines.

Exporting data

Use the [Export] button in the toolbar to export animation data into your project for compilation/linking.

Keyboard shortcuts

Shift + arrow keys: move currently selected step
Space bar: start/stop current animation playback

Library reference

Library defines

Flip modes defines

Flip modes used for graphical elements to define orientation.

Flip modes	
FLIP_NONE	unflipped
FLIP_X	horizontal flip
FLIP_Y	vertical flip
FLIP_XY	horizontal and vertical flip
FLIP_BOTH	horizontal and vertical flip

Job meter defines

Colors used for job meter

Colors	
JOB_BLACK	black color
JOB_WHITE	white color
JOB_RED	red color
JOB_GREEN	green color
JOB_BLUE	blue color
JOB_PURPLE	purple color
JOB_CYAN	cyan color
JOB_YELLOW	byellow color
JOB_ORANGE	orange color
JOB_PINK	pink color

Input related defines

Defines used to read controller data and check button presses.
All data registers are byte size.

Hardware registers	
P1_HW	hardware controller port 1 (negative logic)
P2_HW	hardware controller port 2 (negative logic)

Bios registers	
P1_STATUS	player 1 status
P1_PAST	player 1 previous frame data
P1_CURRENT	player 1 current data
P1_EDGE	player 1 active edge data
P1_REPEAT	player 1 repeat data
P1_TIMER	player 1 repeat timer
P2_STATUS	player 2 status
P2_PAST	player 2 previous frame data
P2_CURRENT	player 2 current data
P2_EDGE	player 2 active edge data
P2_REPEAT	player 2 repeat data
P2_TIMER	player 2 repeat timer
P1B_STATUS	player 3 status
P1B_PAST	player 3 previous frame data
P1B_CURRENT	player 3 current data
P1B_EDGE	player 3 active edge data
P1B_REPEAT	player 3 repeat data
P1B_TIMER	player 3 repeat timer
P2B_STATUS	player 4 status
P2B_PAST	player 4 previous frame data
P2B_CURRENT	player 4 current data
P2B_EDGE	player 4 active edge data
P2B_REPEAT	player 4 repeat data
P2B_TIMER	player 4 repeat timer
PS_CURRENT	current select/start data
PS_EDGE	active edge select/start data

Controller types (status byte value)	
CTRL_NOCONNECT	not connected
CTRL_STANDARD	standard controller
CTRL_EXPANDED	expanded controller (4P mode)
CTRL_KEYBOARD	keyboard
CTRL_MAHJONG	mahjong controller

Controller positions

JOY_UP	lever up
JOY_DOWN	lever down
JOY_LEFT	lever left
JOY_RIGHT	lever right
JOY_A	A button
JOY_B	B button
JOY_C	C button
JOY_D	D button
P1_START	player 1 start button (select/start register)
P1_SELECT	player 1 select button (select/start register)
P2_START	player 2 start button (select/start register)
P2_SELECT	player 2 select button (select/start register)
P1B_START	player 3 start button (select/start register)
P1B_SELECT	player 3 select button (select/start register)
P2B_START	player 4 start button (select/start register)
P2B_SELECT	player 4 select button (select/start register)

Mahjong controller related

P1_JONG_A_G	player 1 mahjong data, A-G buttons
P1_JONG_H_N	player 1 mahjong data, H-N buttons
P1_JONG_BTN	player 1 mahjong data, action buttons
P2_JONG_A_G	player 2 mahjong data, A-G buttons
P2_JONG_H_N	player 2 mahjong data, H-N buttons
P2_JONG_BTN	player 2 mahjong data, action buttons
JONG_A	A button
JONG_B	B button
JONG_C	C button
JONG_D	D button
JONG_E	E button
JONG_F	F button
JONG_G	G button
JONG_H	H button
JONG_I	I button
JONG_J	J button
JONG_K	K button
JONG_L	L button
JONG_M	M button
JONG_N	N button
JONG_PON	PON button
JONG_CHI	CHI button
JONG_KAN	KAN button
JONG_RON	RON button
JONG_REACH	REACH button

Library variables

General variables

DWORD	DAT_frameCounter	frame counter
DWORD	DAT_droppedFrames	dropped (skipped) frames counter
DWORD	SC1[760]	draw list for tilemap data
WORD	*SC1ptr	pointer to tilemaps data draw list
WORD	SC234[2280]	draw list for sprite control
WORD	*SC234ptr	pointer to sprites control draw list
DWORD	PALJOBS[514]	palette jobs queue
WORD	*palJobsPtr;	pointer to palettes jobs queue

Timer interrupt related variables

WORD	LSPCmode	requested LSPC mode
DWORD	TIbase	timer interrupt timing to first trigger
DWORD	TIreload	timer interrupt reload timing
WORD*	TInextTable	pointer to data table to use next frame
WORD	Tivalues0[256]	timer interrupt data space 0
WORD	TIvalues1[256]	timer interrupt data space 1

General purpose components

MEMBYTE, MEMWORD, MEMDWORD

Direct memory access macros.

Syntax

MEMBYTE(*address***)**
MEMWORD(*address***)**
MEMDWORD(*address***)**

Explanation

Macros that can be used to directly access a memory address or hardware register.
Available for byte, word and dword operation.

Ex:

```
i=MEMWORD(0x3c0006);      /* reads LSPC mode register into i */

MEMBYTE(0x300001)=1;      /* kicks watchdog */
```

Note: 68000 requires even addresses when operating on word and dword (long) data. Read/write operation at an odd address for a word/long will crash the CPU.

Return value

N/A

volMEMBYTE, volMEMWORD, volMEMDWORD

Direct memory access macros, volatile declaration.

Syntax

volMEMBYTE(*address***)**
volMEMWORD(*address***)**
volMEMDWORD(*address***)**

Explanation

Macros that can be used to directly access a memory address or hardware register.
Available for byte, word and dword operation.
Theses macros are defined with the volatile keyword.

Ex:

```
i=volMEMWORD(0x3c0006);   /* reads LSPC mode register into i */

volMEMWORD(0x300001)=1;   /* kicks watchdog */
```

Note: 68000 requires even addresses when operating on word and dword (long) data. Read/write operation at an odd address for a word/long will crash the CPU.

Return value

N/A

palJobPut

Writes to palette jobs queue. Macro.

Syntax

palJobPut(

BYTE *number*	Destination palette number (0-255)
BYTE *count*	Number of palettes to write
WORD* *data* **)**	Pointer to palette data start

Explanation

Macro allowing user to put palette jobs on palettes queue.

Return value

N/A

SC1Put

Writes to the tilemap data draw queue. Macro.

Syntax

SC1Put(

WORD *addr*	Destination address in VRAM
BYTE *size*	Tile count
BYTE *pal*	Base palette
WORD* *data* **)**	Pointer to tilemap data

Explanation

Macro allowing user writes into the tilemap data draw queue (VRAM sprite control block 1).

Maximum size is 32 tiles.

Return value

N/A

SC234Put

Writes to the sprite control draw queue. Macro.

Syntax

SC234Put(
WORD *addr* Destination address in VRAM
WORD *data* **)** Data

Explanation

Macro allowing user writes into the sprite control draw queue (VRAM sprite control blocks 2 3 & 4).

Whilst designed for sprite control, the usage can be expanded to write a WORD data to any VRAM address.

Return value

N/A

clearFixLayer

Clears fix layer.

Syntax

void clearFixLayer();

Explanation

Clears the display fix layer.
Clearing is done witth tile 0x0ff, palette 0x0, make sure it is transparent in your fix data.

clearFixLayer totally wipes the fix data, unlike bios FIX_CLEAR function which leaves black bars.

Note: this function operates immediately, not on next vblank.

Return value

N/A

clearSprites

Clears a set of sprites.

Syntax

void clearSprites(
WORD *spr,* First sprite to clear
WORD *count* **)** Number of sprites to clear, from starting sprite

Explanation

Clears a block of sprites from *spr* to *spr+count-1*.
Sprite clearing is done by unlinking it, setting a 0 size and position it offscreen. Tiledata isn't affected.

Return value

N/A

disableIRQ

Disables IRQ on the system.

Syntax

void disableIRQ()

Explanation

IRQ will no longer be triggered after calling this function.

Disables both IRQ1 and IRQ2.

Return value

N/A

enableIRQ

Enables IRQ on the system.

Syntax

void enableIRQ()

Explanation

IRQ will be active after calling this function.

Enables both IRQ1 and IRQ2.

Return value

N/A

initGfx

Initialize the library for graphics operations.

Syntax

void initGfx()

Explanation

Resets and sets up library for operation.
Calling this function is required before using the library.

The function notably resets frame counters and unloads timer interrupt function.

Return value

N/A

jobMeterColor

Changes current jobmeter color.

Syntax

void jobMeterColor(
WORD *color* **)** Requested color

Explanation

Macro used to change job meter color to differentiate code segments execution timing.

Return value

N/A

jobMeterSetup

Sets up the job meter.

Syntax

void jobMeterSetup(
BOOL *setDip* **)** Automatic soft dip setting

Explanation

Draws the job meter of the fix layer, using fix tile 0x000 and palette 0xf. Make sure that tile is a plain color #1 tile in your fix data for proper display.
Job meter takes place on the far right column of the fix layer.

For the job meter to be updated during vblank, devmode and soft dip 2-1 must be on.

Call function with *setDip* parameter set to *true* for the function to force devmode and soft dip 2-1 to on. This basically saves you from enabling them again manually on each boot.

Note: Forcing bios setting is kind of a hack job, it isn't guaranteed to work on all bios (tested ok on debug bios and uinibios 3.2), try out and use accordingly. Do not use in release code.

Return value

N/A

loadTIirq

Loads timer interrupt handler.

Syntax

void loadTIirq(
ushort *mode***)** IRQ mode

Explanation

Loads the required code to process the timer interrupt.

Two modes are available:

- `TI_MODE_SINGLE_DATA:` One VRAM change per interrupt
- `TI_MODE_DUAL_DATA:` Two VRAM changes per interrupt

Return value

N/A

SCClose

Readies draw data for display.

Syntax

void SCClose()

Explanation

Closes draw lists and prepare system for next vblank.

SCClose will allow draw lists to be processed upon next vblank and therefore need to be called before wait_vblank, or the library won't update display and will issue a frameskip.

Return value

N/A

setup4P

Initialize 4P input mode.

Syntax

int setup4P()

Explanation

This function will check if a 4P adapter (NEO-FTC1B / NEO-4P) is hooped to the system.
It should enable 4 players mode on any bios if hardware is found.

Return value

0 - adapter could not be found
1 - adapter was found

unloadTIirq

Unloads timer interrupt handler.

Syntax

void unloadTIirq()

Explanation

Unloads the required code to process the timer interrupts.

This actually loads a failsafe handler (acknowledge IRQ then return), shall a timer interrupt occur when unexpected.

Note: make sure you set `TinextTable` to 0, then wait for a VBlank to occur before using `unloadTIirq()` to avoid unstable behavior.

Return value

N/A

wait_vblank

Waits for next vblank.

Syntax

void wait_vblank()

Explanation

Holds program execution until next vblank is triggered.

Program will resume after the vblank function has been processed.

Return value

N/A

Pictures components

Picture

Runtime handler for a picture.

Syntax

```
typedef struct Picture {
    WORD baseSprite;        Base sprite # used for this picture
    WORD basePalette;       Base palette # used for this picture
    WORD posX;              Current position, X axis
    WORD posY;              Current position, Y axis
    WORD currentFlip;       Current flip mode.
    pictureInfo* info;      Pointer to the pictureInfo struct of this picture
} Picture;
```

Explanation

This is the base structure the library uses to handle picture type elements.
Has to be allocated in the ram section of your code.

As operation on this datatype is managed by the library, it is strongly advised to use as read only in your code.

pictureInfo

Structure holding picture information.

Syntax

```
typedef struct pictureInfo {
    WORD colSize;          Words size of each sprite tilemap (basically tileHeight*2)
    WORD unused__height;   Legacy field, unused/reserved future use
    WORD tileWidth;        Picture width, tiles unit
    WORD tileHeight;       Picture height, tiles unit
    WORD* maps[4];         Pointers to tilemaps (standard, flipX, flipY, flipXY)
} pictureInfo;
```

Explanation

pictureInfo structures are generated by the buildchar tool. Holds basic info about the picture.

Tilemap pointers are always valid. IE if you did not request flipX for that picture, maps[1] will point to the standard map.
Picture tilemaps size (WORD) is (tileWidth*tileHeight)*2.

pictureHide

Hide a picture.

Syntax

void pictureHide(
Picture* *p* **)** Pointer to picture structure to use

Explanation

Removes designated picture element from display.

Note: As hiding is done by altering Y position and sprite size, please be aware that changing Y pos of designated picture will revert it back to visible.

Return value

N/A

pictureInit

Initialize a Picture structure for use.

Syntax

void pictureInit(

Picture* *p,*	Pointer to Picture structure to use
pictureInfo* *pi,*	Pointer to pictureInfo structure
WORD *baseSprite,*	Base sprite # to use
BYTE *basePalette,*	Base palette # to use
short *posX,*	Picture initial X position
short *posY,*	Picture initial Y position
WORD *flip* **)**	Picture initial flip mode

Explanation

Initialize and prepare a Picture element for use.
Picture will be set up with provided initial position/flip.
Will reset related sprites shrink factor to 0x0fff (full size).

Return value

N/A

pictureMove

Updates position of a picture.

Syntax

void pictureMove(
Picture* *p,* Pointer to Picture structure to use
short *shiftX,* New X position
short *shiftY* **)** New Y position

Explanation

Change picture screen position.
New position is determined relatively to current position (new pos= current pos + shift).

Return value

N/A

pictureSetFlip

Sets flip mode of a picture.

Syntax

void pictureSetFlip(
Picture* *p,* Pointer to picture structure to use
WORD *flip* **)** Desired flip mode

Explanation

Change picture flip mode.
Flip modes most be specified in your chardata.xml file for the buildchar tool to make them available.
Will default to base orientation if requested flip mode isn't available.

Return value

N/A

pictureSetPos

Sets position for a picture.

Syntax

void pictureSetPos(
Picture* *p,* Pointer to Picture structure to use
short *toX,* New X position
short *toY* **)** New Y position

Explanation

Change picture screen position.
Position is set to supplied values.

Return value

N/A

pictureShow

Show a picture.

Syntax

void pictureShow(
Picture* *p* **)** Pointer to picture structure to use

Explanation

Put back a previously hidden picture on display.
Picture will be displayed at latest set position with latest set flip.

Return value

N/A

Scrollers components

Scroller

Runtime handler for a scroller.

Syntax

```
typedef struct Scroller {
    WORD baseSprite;        Base sprite # used for this scroller
    WORD basePalette;       Base palette # used for this scroller
    WORD colNumber[21];     Internal use
    WORD topBk, botBk;      Internal use
    WORD scrlPosX;          Current scroll index, X axis
    WORD scrlPosY;          Current scroll index, Y axis
    scrollerInfo* info;     Pointer to the scrollerInfo struct of this scroller
} Scroller;
```

Explanation

This is the base structure the library uses to handle scroller type elements.
Has to be allocated in the ram section of your code.

As operation on this datatype is managed by the library, it is strongly advised to use as read only in your code.

scrollerInfo

Structure holding scroller information.

Syntax

```
typedef struct scrollerInfo {
        WORD colSize;          Words size of each sprite tilemap (basically mapHeight*2)
        WORD sprHeight;        Required sprite height to use (max 32)
        WORD mapWidth;         Scroller width, tiles unit
        WORD mapHeight;        Scroller height, tiles unit
        WORD map[0];           Tilemap data (size varies)
} scrollerInfo;
```

Explanation

scrollerInfo structures are generated by the buildchar tool. Holds basic info about the scroller.

Actual map size (WORD) is (mapWidth*mapHeight)*2.

scrollerInit

Initialize a Scroller structure for use.

Syntax

void scrollerInit(

Scroller* *s,*	Pointer to Scroller structure to use
scrollerInfo* *si,*	Pointer to scrollerInfo structure
WORD *baseSprite,*	Base sprite # to use
BYTE *basePalette,*	Base palette # to use
short *posX,*	Scroller initial X position
short *posY* **)**	Scroller initial Y position

Explanation

Initialize and prepare a Scroller element for use.
Scroller will be set up with provided initial scroll positions.
Will reset related sprites shrink factor to 0x0fff (full size).

Note: do not use negating scroll positions, this will display junk on screen and hog the CPU. If you want to introduce a scrolling plane starting offscreen, add an empty lead-in screen to your picture file.

Return value

N/A

scrollerSetPos

Initialize a Scroller structure for use.

Syntax

void scrollerInit(
Scroller* *s,* Pointer to Scroller structure to use
short *toX,* Scroller X position
short *toY* **)** Scroller Y position

Explanation

Initialize and prepare a Scroller element for use.
Updates scroll positions to provided values.

Note: do not use negating scroll positions, this will display junk on screen and hog the CPU. If you want to introduce a scrolling plane starting offscreen, add an empty lead-in screen to your picture file.

Return value

N/A

Animated sprites components

aSprite

Runtime handler for an animated sprite.

Syntax

```
typedef struct aSprite {
    WORD baseSprite;            Base sprite # used for this animated sprite
    WORD basePalette;           Base palette # used for this animated sprite
    short posX;                 Animated sprite current X position
    short posY;                 Animated sprite current Y position
    short currentStepNum;       Current step number
    short maxStep;              Max step # of current animation
    sprFrame* frames;           Unused / deprecated
    sprFrame* currentFrame;     Pointer to current frame data
    animation* anims;           Pointer to animations block
    animation* currentAnimation; Pointer to current animation
    animStep* steps;            Pointer to steps block os current animation
    animStep* currentStep;      Pointer to current step
    DWORD counter;              Frame update counter - internal
    WORD repeats;               Number of repeats done
    WORD currentFlip;           Current flip mode
    WORD tileWidth;             Width of current frame, tiles unit
    WORD animID;                ID of current animation
    WORD flags;                 Flags - internal
} aSprite;
```

Explanation

This is the base structure the library uses to handle animated sprites elements.
Has to be allocated in the ram section of your code.

As operation on this datatype is managed by the library, it is strongly advised to use as read only in your code.

When animation has reached its end (when applicable), counter value will change to 0xffffffff.

spriteInfo, animation, animStep, sprFrame

Structures holding animated sprites informations.

Syntax

```
typedef struct spriteInfo {
        WORD palCount;          Number of required color palettes
        WORD frameCount;        Total number of frames
        WORD maxWidth;          Maximum width, tiles unit (width of the largest frame)
        animation* anims;       Pointer to animations block
} spriteInfo;

typedef struct animation {
        WORD stepsCount;        Number of animation steps for this animation
        WORD repeats;           Number of repeats for this animation
        animStep* data;         Pointer to steps block
        struct animation* link; Pointer to linked animation
} animation;

typedef struct animStep {
        sprFrame* frame;        Pointer to frame info
        short shiftX;           Frame X displacement from origin
        short shiftY;           Frame Y displacement from origin
        short duration;         Number of frame to display
} animStep;

typedef struct sprFrame {
        WORD tileWidth;         Frame width, tiles unit
        WORD tileHeight;        Frame height, tiles unit.
        WORD colSize;           Words size of each sprite tilemap (basically tileHeight*2)
        WORD* maps[4];          Pointers to frame tilemaps (standard, flipX, flipY, flipXY)
} sprFrame;
```

Explanation

spriteInfo, animation, animStep and sprFrame structures are generated by the buildchar and animator tools. Holds infos about animated sprite frames and animations.

Frame tilemap pointers are always valid. IE if you did not request flipX for that sprite in the buildchar tool, maps[1] will point to the standard map.

Frame tilemaps size (WORD) are (tileWidth*tileHeight)*2.

aSpriteAnimate

Animates an animated sprite.

Syntax

void aSpriteAnimate(
aSprite* *as* **)** Pointer to aSprite structure

Explanation

Updates the animated sprite animation.
Will apply position/flip/animation changes and queue required data on draw lists for update next vblank.
This function must be called every frame for each animated sprite for proper animation.

Return value

N/A

aSpriteFlip

Sets flip mode of an animated sprite.

Syntax

void aSpriteFlip(
aSprite* *as,* Pointer to aSprite structure
WORD *flip* **)** Desired flip mode

Explanation

Change animated sprite flip mode.
Flip modes most be specified in your chardata.xml file for the buildchar tool to make them available.
Will default to base orientation if requested flip mode isn't available.
Will not update the display directly, use aSpriteAnimate afterward to apply changes.

Return value

N/A

aSpriteHide

Hides an animated sprite (macro).

Syntax

void aSpriteHide(
aSprite* *as* **)** Pointer to aSprite structure

Explanation

Flag the designated aSprite as no display.
When flagged as no display, animated sprites will no longer be displayed. This allows to keep animating an offscreen/hidden object withut having to display it.

Note: If the aSprite is currently used in allocated mode, you must manually clear the sprites used by the current frame => `clearSprites(as->baseSprite, as->tileWidth);`

Return value

N/A

aSpriteInit

Initialize an aSprite structure for use.

Syntax

void aSpriteInit(

aSprite* *as,*	Pointer to aSprite structure to use
spriteInfo* *si,*	Pointer to spriteInfo structure
WORD *baseSprite,*	Base sprite # to use
BYTE *basePalette,*	Base palette # to use
short *posX,*	aSprite initial X position
short *posY,*	aSprite initial Y position
WORD *anim,*	aSprite initial animation sequence
WORD *flip* **)**	aSprite initial flip mode

Explanation

Initialize and prepare an aSprite element for use.
aSprite will be set up with provided initial position/animation/flip.
This function will not push frame to display, a call to aSpriteAnimate is required after aSpriteInit to push initial frame on display upon next vblank.
Will reset related sprites shrink factor to 0x0fff (full size).

Return value

N/A

aSpriteMove

Updates position of an animated sprite.

Syntax

void aSpriteMove(
aSprite* *as,* Pointer to aSprite structure
short *shiftX,* X shift value
short *shiftY* **)** Y shift value

Explanation

Change animated sprite screen position.
New position is determined relatively to current position (new pos= current pos + shift).
Will not update the display position directly, use aSpriteAnimate afterward to apply changes.

Return value

N/A

aSpriteSetAnim

Sets animation for an animated sprite.

Syntax

void aSpriteSetAnim(
aSprite* *as,* Pointer to aSprite structure
WORD *anim* **)** Animation ID

Explanation

Change current animation.
Animation ID are defines issued by the animator tool, see documentation for syntax.
Will not push frame to display, use aSpriteAnimate afterward to apply changes.
If requesting change to the animation sequence ID that is already running, nothing will be done.

About animation links:
When using linked animations (ie A > B > C (loop)) system will remember “A” as last requested animation ID.
This means if said animated sprite ran long enough to reach animation “C”, a request for animation ID “A” will be discarded at this is the sequence already running.

This behavior can be a problem depending on usage and how well you plan your animations sequences. In the case you want to force a sequence rewind, alter the animID field before calling aSpriteSetAnim to trick it out:

```
myaSpite.animID=0xffff;   /* unlikely you did that many animations, should
                             be an unused ID */
aSpriteSetAnim(&myaSprite,DUMMY_IDLE);  /* sets sequence */
```

Return value

N/A

aSpriteSetPos

Sets position for an animated sprite.

Syntax

void aSpriteSetPos(
aSprite* *as,* Pointer to aSprite structure
short *newX,* New X position
short *newY* **)** New Y position

Explanation

Change animated sprite screen position.
Will not update the display position directly, use aSpriteAnimate afterward to apply changes.

Return value

N/A

aSpriteShow

Shows back an animated sprite (macro).

Syntax

void aSpriteShow(
aSprite* *as* **)** Pointer to aSprite structure

Explanation

Removes the no display flag from the designated aSprite.
Returns the aSprite to its normal state, allowing it to be displayed again.

Return value

N/A

Sprite Pools components

spritePool

Runtime handler for a sprite pool.

Syntax

```
typedef struct spritePool {
        WORD poolStart;         Fist sprite # used for this sprite pool
        WORD poolEnd;           Last sprite # used for this sprite pool
        WORD poolSize;          Sprite pool size
        WORD way;               Current draw direction
        WORD currentUp;         Internal use
        WORD currentDown;       Internal use
} spritePool;
```

Explanation

This is the base structure the library uses to handle sprite pools elements.
Has to be allocated in the ram section of your code.

As operation on this datatype is managed by the library, it is strongly advised to use as read only in your code.

Way is either `WAY_UP` or `WAY_DOWN`.

spritePoolClose

Finalize sprite pool operations for display.

Syntax

void spritePoolClose(
spritePool **sp* **)** Pointer to spritePool structure

Explanation

Prepares a sprite pool for next VBlank.
Needs to be called before each Vblank, will switch pool direction and queue the necessary sprite clears for correct display.

Note: Sprite pool passed to this function is not to be used before next Vblank has occurred.

Return value

Will return 1 when draw operations exceeded total pool size, 0 otherwise.

spritePoolDrawList

Draws the supplied animated sprites list into sprite pool.

Syntax

void spritePoolDrawList(
spritePool **sp* Pointer to spritePool structure
void **list* **)** Pointer to draw list

Explanation

Utilize the supplied sprite pool to render the aSprite items in the supplied list.
This function takes care of updating the animation state, then display the updated item.

Notes: User must supply a list pointer according to the current direction of the sprite pool :

- `WAY_UP`: list must point to the first item, list will be read upward until null is found
- `WAY_DOWN`: list must point to the last+1 element, list will be read downward until null is found

This function isn't interrupt safe.

Return value

N/A

spritePoolInit

Initialize the supplied sprite pool handler.

Syntax

void spritePoolInit(
spritePool **sp,* Pointer to spritePool structure
WORD *baseSprite,* Startig sprite of sprite pool
WORD *poolSize* **)** Sprite pool size

Explanation

Sets up the supplied sprite pool handler for use.

This function will issue a sprite clear of sprites *baseSprite* to *baseSprite+poolSize-1*.

Return value

N/A