Sprite

COLLABORATORS

	TITLE : Sprite	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 23, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sprite

1.1 Sprite V1.00

```
PureBasic - Sprite library V1.00

  'Sprites' are well known from game players. This is small pictures,
  sometimes called 'brushes', which can be displayed at any position
  on a screen. The sprites can move over graphics without destroy
  them, unlike other graphics operations which are destructives.
  PureBasic sprites are fully based on the 'Blitter', a very fast
  hardware chip which can move quickly tons of data. This library is
  very optimized and as a particularity: it works in parallel with
  the main CPU, the 680x0. So you can display some sprite and
  use the CPU to achieve other tasks like artificial intelligence.
  All these functions are 100% OS friendly and you can use it for
  any kind of programs.

Commands summary in alphabetical order:

   AddBlockSprite
   AddBufferedSprite
   AddSprite
   CreateSpriteBuffer
   FreeSprite
   FreeSpriteBuffer
   InitSprite
   LoadSprite
   ResetSpriteServer
   RestoreBackground
   SpriteDepth
   SpriteHeight
   SpriteWidth
   StartSpriteServer
   StopSpriteServer
   UseSpriteBuffer
   WaitSpriteServer

Example:

   Sprite example
```

1.2 addblocksprite

```
  SYNTAX
AddBlockSprite(#Sprite, x, y)

  COMMAND
Display the sprite at the specified position on the current sprite
buffer. This function is the fastest way to display a sprite at
the screen but has some limitations:

  + The sprite width must be a multiple of 16 (ie: 16, 32, 48, 64...)
  + The position 'x' must be a multiple of 16.
  + There is no transparent colour for this sprite.


Note: If another sprite is being displayed, the function add this
sprite to the server queue list and return immediately. You can
never assume than this sprite is effectively display when this
function return ! Use the command 'WaitSpriteServer()' if you
want to be sure than this sprite is really displayed. See the
'StartSpriteServer()' description for more informations about
this asynchrone process.
```

1.3 addbufferedsprite

```
  SYNTAX
AddBufferedSprite(#Sprite, x, y)

  COMMAND
Display the sprite at the specified position on the current sprite
buffer. Before to display this sprite, the background which will be
destroyed by the sprite is saved in the sprite buffer. You must
allocate some memory to store the background data (this is done
via 'CreateSpriteBuffer()'). The saved background can be restored
later with the command 'RestoreBackground()'. The colour 0 of the
sprite is considered as transparent. This command is not clipped,
so be sure to display the sprite inside the BitMap.


Note: If another sprite is being displayed, the function add this
sprite to the server queue list and return immediately. You can
never assume than this sprite is effectively display when this
function return ! Use the command 'WaitSpriteServer()' if you
want to be sure than this sprite is really displayed. See the
'StartSpriteServer()' description for more informations about
this asynchrone process.
```

1.4 addsprite

```
  SYNTAX
AddSprite(#Sprite, x, y)
```

```
  COMMAND
Display the sprite at the specified position on the current sprite
buffer. The background area is destroyed by this function. If you
need to preserve the background, use 'AddBufferedSprite()' instead.
The colour 0 of the sprite is considered as transparent. This
command is not clipped, so be sure to display the sprite inside
the BitMap.


Note: If another sprite is being displayed, the function add this
sprite to the server queue list and return immediately. You can
never assume than this sprite is effectively display when this
function return ! Use the command 'WaitSpriteServer()' if you
want to be sure than this sprite is really displayed. See the
'StartSpriteServer()' description for more informations about
this asynchrone process.
```

1.5 createspritebuffer

```
  SYNTAX
CreateSpriteBuffer(#SpriteBuffer, Size, BitMapID)

  FUNCTION
Creates and initializes a new sprite buffer. 'BitMapID' is the identifiant
of the bitmap on which you want to display the sprites. 'Size' is only useful
when you display some sprites with the 'AddBufferedSprite()' command,
when a background is saved. This 'Size' is function of the number of
sprites displayed at the same time, their sizes, and of the bitmap
depth. To calculate it, you can use the following rule:

  Size = (BitMapDepth * (SpriteWidth*SpriteHeight)*NumberOfSpriteDisplayed) / 8  ↩
     + 1000

This newly created sprite buffer becomes the current sprite buffer.
```

1.6 freesprite

```
  SYNTAX
FreeSprite(#Sprite)

  FUNCTION
Remove the specified sprite from memory. You can no more use it.
```

1.7 freespritebuffer

```
  SYNTAX
FreeSpriteBuffer(#SpriteBuffer)

  FUNCTION
Free the specified sprite buffer and release all its allocated memory.
```

1.8 initsprite

```
  SYNTAX
Result = InitSpriteFile(#MaxDisplayedSprites, #MaxSpriteBuffers, #MaxSprites)

  FUNCTION
Init all the sprite environments for later use.  You  must  put  this
function at the top of your source code if you want to use the sprite
commands. You can test the result to see if the sprite environment
has been correctly initialized. If not, quit the program or disable
all the calls to the sprite related commands.
```

1.9 loadsprite

```
  SYNTAX
Result = LoadSprite(#Sprite, FileName$)

  STATEMENT
Load the specified sprite into the memory for immediate use. The sprite
must be in IFF/ILBM format (compressed or not, both cases are supported).
If something wrong happens, a negative value is returned. Else, all is
fine... Here are the possible values for 'Result':

  -1 : File not found or can't be opened.
  -2 : This file is not an IFF/ILBM file
  -3 : Not enough memory
  -4 : Corrupted IFF/ILBM file
```

1.10 resetspriteserver

```
  SYNTAX
ResetSpriteServer()

  STATEMENT
Once you have finished to display all the needed sprites, you have to reset
the sprite server to tell the system than all is fine. Internally, the sprite
queue list is reseted to 0, so all non yet displayed sprites will be never
displayed. Use the 'WaitSpriteServer()' function to be sure than the sprite
server has finished. A good solution is to reset the sprite server at every
frame (for a game of course).
```

1.11 restorebackground

```
  SYNTAX
RestoreBackground()

  STATEMENT
Restore the previously destroyed background of the current sprite buffer.
Each sprite buffer has its own background area. The background has been
saved with the command 'AddBufferedSprite()'.
```

1.12 spritedepth

```
  SYNTAX
Result.w = SpriteDepth(#Sprite)

  STATEMENT
Return the depth of the specified sprite.
```

1.13 spriteheight

```
  SYNTAX
Result.w = SpriteHeight(#Sprite)

  STATEMENT
Return the height in pixel of the specified sprite.
```

1.14 spritewidth

```
  SYNTAX
Result.w = SpriteWidth(#Sprite)

  STATEMENT
Return the width in pixel of the specified sprite.
```

1.15 startspriteserver

```
  SYNTAX
StartSpriteServer()

  STATEMENT
Allocate the 'Blitter' chip ressources in OS compliant way and intialize
the server for immediate use. Once you have called this function, you can
use quietly any of the AddSprite() functions. But remember than the
Blitter is owned only by your program and the whole OS can no more use
it (to draw window, text...). The graphics are freezed, so don't forget to
stop the server as soon as you don't need anymore of the sprite
functionnality. A good idea is to start/stop the server at every frame
so you will ensure a minimum for the OS survive... This function is of course
very fast.


Some additional notes about the sprite server (for advanced users):

The server can be see as a queue list and when you add a sprite to
this list there is two possibility: the list is empty, so the sprite
is displayed immediately, or the list has some entries and your sprite
is added at the end of the list. All the entry are processed one
after one, in the right order. The good point on this system is than
the main CPU doesn't need to wait until the 'Blitter' finishes his work.
```

So we gain substantial CPU power against other classic solutions.
To be sure than all your sprites has been displayed, simply use the
'WaitSpriteServer()' command.

1.16 stopspriteserver

SYNTAX
StopSpriteServer()

STATEMENT
Stop the sprite server activity and release immediately the 'Blitter'
chip to the AmigaOS. If some sprites haven't been displayed, there are
lost. This command is very fast.

1.17 usespritebuffer

SYNTAX
UseSpriteBuffer(#SpriteBuffer)

STATEMENT
Change the current sprite buffer with the supplied one.

1.18 waitspriteserver

SYNTAX
WaitSpriteServer()

STATEMENT
Wait until the sprite server has finished to display all the sprites.
This command is needed unless you are sure than all your sprites
has been displayed. This command should be tipically called at the
end of the main loop, when all other 'CPU' only functions have been
processed. It's one of the advantage of the parallel working: you can
use the CPU while the sprite are displayed.