Inside FBSpriteWorld

by Robert Hommel and Tony Myles

SpriteWorld was originally written by Tony Myles in THINK C and generously placed by him in the public domain. FBSpriteWorld is my attempt to implement a SpriteWorld-like animation architecture in FutureBasic. It is not a line-by-line, routine-by-routine translation of the THINK C version. It does not contain the many optimizations and assember bit blitting routines found in the original. While Tony's C version can be used 'right out of the box' to produce commercial-quality results, you will probably have to tweak FBSpriteWorld to achieve the same. Anyone interested in serious animation on the Macintosh should download Tony's SpriteWorld - it is a remarkable piece of work, and my version of it is a pale imitation, at best.

FBSpriteWorld was written with Tony Myles' permission and encouragement. However, he had nothing to do with the FutureBasic code itself. All deficiencies, omissions, and bugs are mine and mine alone.

I am publishing FBSpriteWorld for a number of reasons. First, because I have benefited greatly from the generousity of other programmers who have placed their code in the public domain. FBSpriteWorld is a way to partially repay the debt. Second, because of the great deal of interest in producing animation on the Macintosh. While it may not seem so at first (as we struggle our way through off-screen pixel maps and color tables), the Mac is a wonderful tool for producing high-quality audio/video effects. This code will, I hope, provide some insights into how to take the first steps in producing high-quality animation on our favorite computer. Third, to demonstrate (as if we need to) that FutureBasic is a serious programming environment that rivals C or any other language in its power and ease-of-use. Finaly, I am publishing FBSpriteWorld as a starting place. I encourage you to use FBSpriteWorld as a foundation for bigger and better things. Make it faster, smaller, easier, more powerful! And, please, if you write that FrameFromCIcon function or assembly language bit blitter, post it for the rest of us!

If we work together, maybe we can equal or even exceed the original SpriteWorld. Who knows, maybe we can even convince Tony that FB>C!!

What is FBSpriteWorld?

FBSpriteWorld is a collection of routines that you can use to implement smooth animation in your applications. FBSpriteWorld was designed with an eye towards the style of animation seen in color arcade games in particular. You can use SpriteWorld to…

- perform smooth multi-layered animation
- perform collision detection
- create animations from pict resources
- synchronize animation on millisecond intervals
- perform simple about box animations or write full blown arcade games
- create animation that will perform equally well at all screen depths

To use SpriteWorld it may help to have at least a passing familiarity with…

- QuickDraw, especially offscreen GrafPorts, CopyBits, and regions
- the Time Manager

Legalities

The only thing I ask in exchange for FBSpriteWorld, is a free copy for both Tony Myles and myself (that is make us fully paid, registered users) of any cool game that you write with FBSpriteWorld.

Tony Myles
America Online: Suiryu
Compu$erve: 72070,3000
Internet: suiryu@aol.com

Robert Hommel
America Online: RWHommel
Compuserve: 71061,3327
Internet:
hommelr@ccmail.avery.com

I would further ask that if you use my routines verbatim in your program, you credit me in your program and/or documentation.

Introduction to Sprites

What is a sprite?

“**Sprite**” is a technical term meaning an animated object that appears on the computer’s screen and may move around or exhibit other interesting behaviour. A good example is an arcade game in which you pilot a space ship against hordes of alien invaders. In this case the ship and little aliens can be considered sprites.

What a sprite really is in terms of FBSpriteWorld’s implementation, is basically a data structure that contains a series of the graphic images or frames of the sprite, a rectangle that specifies where on the screen the sprite is to be drawn, and various other parameters that specify how far it should move and in what direction, as well as when it should move and be drawn. FBSpriteWorld provides a suite of routines you can use to create sprites, and specify all of their animation characteristics.

Sprites have this notion of a current frame. The current frame of a sprite is the image that will be drawn when the the animation is rendered on the screen. By advancing the current frame in sequence the sprite will appear to be animated. At the same time the screen position at which the sprite’s current frame will be drawn can be adjusted giving the illusion of movement.

Sprites also support the notion of collisions. As a sprite moves around on the screen, it may come in contact with another sprite. This is called a collision. FBSpriteWorld provides routines to detect collisions and act upon them. By default nothing will happen, the sprites will harmlessly pass through each other completely unaware of anything. On the screen the sprite images will smoothly overlap one another.

FBSpriteWorld uses sprites to drive the animation. Each frame of the animation is built by processing the sprites and then drawing them. When the sprites are processed, their new positions are calculated based on their movement parameters, and their current frame is changed based on their frame advance parameters. In general the sprites are used as a mechanism to shield you from all the gory details of the animation.

Introduction to FBSpriteWorld

Why FBSpriteWorld?

Unlike the Amiga and other game machines, the Macintosh has no sprite animation hardware built-in. Sprite animation therefore, must to be implemented in software. FBSpriteWorld is an attempt to implement a sprite-based animation architecture on the Macintosh. This has been done before, but it was not done right, or it was not released to the general developer community.

FBSpriteWorld achieves its smooth animation using a frame differential technique. This means that for each frame of the animation, only the areas of the screen that have changed are actually drawn. This technique uses a double buffering scheme, meaning that two offscreen areas, one to keep a fresh copy of the background and the other to serve as a work area, are used to render the animation before it is drawn on screen. This calls for a tree step process to building a frame of the animation.

- A section of the background is copied to the offscreen work image. The area of this section is calculated by taking the union of the sprite's last position and its current position.
- The current frame of the sprite is then drawn in the sprite's current position in the offscreen work area. The result is the sprite resting on the piece of background copied in step one.
- This piece is then copied from the work area to the corresponding position on the screen, effectively erasing the sprite from its old position on screen, and drawing the sprite in its new position simultaneously.

This simple animation technique is commonly used by games and animation applications on the Macintosh.

When creating animations using FBSpriteWorld you will deal primarily with four simple data structures: SpriteWorlds, SpriteLayers, Sprites, and Frames. These four structures have a containment relationship in that SpriteWorlds contain any number of SpriteLayers, which contain any number of Sprites, which contain one or more Frames.

SpriteWorlds

SpriteWorlds provide a context for the animation to take place. The context provided by a SpriteWorld is essentially the graphics environment for the animation both on and off screen. Everything in the animation happens under the domain of a SpriteWorld.

A SpriteWorld contains a frame for the offscreen background area, the offscreen work area, and the screen itself. You can choose to create these frames yourself, or have them created automatically depending on the circumstances of your animation.

A SpriteWorld also maintains a list of the sprite layers that are taking part in the animation. There are routines for adding and removing layers from a world. There can be any number of layers in a given world.

In terms of the actual drawing the FBSpriteWorld is responsible for erasing the sprite offscreen and drawing the sprite onscreen. You can install custom routines to handle this by writing a custom pixel blitter. By default FBSpriteWorld uses QuickDraw's CopyBits routine for all drawing operations.

SpriteLayers

SpriteLayers are used to maintain groups of related sprites. Using SpriteLayers you can animate sets of sprites in separate overlapping planes, creating the illusion that the sprites are passing in front, and behind other sprites. When drawing occurs each sprite in each layer, and each layer in the world, is drawn consecutively, one overlapping the other. The first layer is drawn first, the last drawn last, so that the sprites in each layer overlap each other properly.

Aside from the animation this layering facility is also used by the collision detection mechanism. If you have your sprites arranged in logical layers, ie. the good guys in one layer, the bad guys in another, then detecting collisions is simply a matter of checking one layer of sprites against another. You can also detect collisions between sprites residing within a single layer.

Sprites

Sprites are the star of the show. Any animation you create will consist of one or more sprites. These sprites move about the screen and do interesting things according to parameters you can specify using the routines provided. You can specify the timing, direction, and distance a sprites moves at any one time. By installing a custom move routine, your sprites can exhibit extremely complex behaviour such as simulated gravitational forces.

A sprite contains one or more frames. As a sprite moves it may change which frame is to be currently drawn, producing the illusion of animation. You can specify the timing of these frame changes, and by installing a custom frame routine, you can perform more sophisticated frame animation such as rotating a space ship when certain key is pressed.

When one sprite overlaps another, a collision has occurred. By installing a collision routine the sprite can take action, when a collision is detected, such as playing an explosion sound.

Frames

Frames are used to maintain the individual graphic images of a sprite. Each frame simply contains an offscreen GrafPort in which the actual image is stored, and a mask.

Using FBSpriteWorld

As this is the first version of FBSpriteWorld, the emphasis is on creating an architecture for animation. In future versions, we will attempt to optimize the speed of the animation and enhance the features set of this archetecture. In this section we will attempt to describe how you can easily make use of FBSpriteWorld to create relatively fast, smooth animation.

Getting Started

Performing animation using FBSpriteWorld involves 4 **core** steps, and 2 initialization or housekeeping steps…

- Initialize the FBSpriteWorld package.
- **Create the various pieces, ie. the FBSpriteWorld, the SpriteLayers, the Sprites, and the Frames.**
- **Assemble the various pieces, ie. add the Frames to the Sprites, add the Sprites to the SpriteLayers, add the SpriteLayers to the FBSpriteWorld.**
- **Set the various movement and frame advance parameters that define a Sprite’s behaviour.**
- **Drive the animation using SWProcessSpriteWorld and SWAnimateSpriteWorld in a tight loop. Collision detection may optionally be performed here.**
- When the animation is finished, you must dispose of all the pieces (FBSpriteWorlds, SpriteLayers, Sprites, and Frames) that were created earlier.

While the animation is running you may add or remove individual Sprites or entire SpriteLayers.

Creating an animation

Before FBSpriteWorld can be used it must first be initialized with a call to SWEnterSpriteWorld. SWEnterSpriteWorld performs some checks to see if FBSpriteWorld can run, and then sets up some internal data structures. You must call SWEnterSpriteWorld before calling any other FBSpriteWorld routine.

```
err = FN SWEnterSpriteWorld
```

Once the FBSpriteWorld package is initialized you can start creating the pieces that will make up your animation. The central piece to any animation is the SpriteWorld. If your application has a window in which you would like to display the animation you can easily create a SpriteWorld by dimensioning a SpriteWorld variable and calling SWCreateSWFromWindow.

```
DIM SpriteWorld.SpriteWorldRec
WINDOW 1                                                 windowPort&=FN
GetCurrPort                                          err = FN
SWCreateSpriteWorld(@SpriteWorld, windowPort&)
```

For even the simplest animation you must create at least one SpriteLayer. This is accomplished by dimensioning a SpriteLayer variable. There is no limit to the number of SpriteLayers that you might use in an animation (the default is 10 Layers; simply change the _maxLayers constant in FBSpriteWorld.glbl if you need more).

```
Dim SpriteLayer.SWSpriteLayerRec
```

Next, we need at least one Sprite. The easiest way to create a Sprite is to base it on a PICT resourse. This does not actually attach the PICT to the Sprite (this is done in succeeding steps), but rather uses the PICT to determine the size of the Sprite's current rectangle. To do this, dimension a Sprite variable and call SWSpriteFrom Pict:

```
DIM mySprite.SWSpriteRec
FN SWSpriteFromPict(@mySprite, currentFrame, curX, curY,  @wRect, ->
   visible, deltaX, deltaY, frameAdvance,  _pictRSRC)
```

Finally, each Sprite must contain at least one frame. Most will contain more. For example, to create 10 frames and add them to a sprite, dimension an array with 10 elements and call SWFrameFromPict, then call SWAddFrameToSprite:

```
DIM myFrame(9).SWFrameRec                    'assumes option  base 0
FOR x=0 to 9
   err=FN SWFrameFromPict(@myFrame(x),  _spriteRSRC+(x*3))
   err=FN SWAddFrameToSprite(@mySprite, @myLayer(x))
NEXT
```

Assembling The Pieces

Before an animation can be run the pieces that you have created must be assembled. This is accomplished by adding the Sprites to the SpriteLayers, and adding the SpriteLayers to the SpriteWorld.

```
   'repeat this call for each sprite
err = FN SWAddSpriteToLayer(@spriteLayer, @mySprite)

   'repeat this call for each sprite layer
err = FN SWAddLayerToWorld(@SpriteWorld, @spriteLayer)
```

Defining Sprite Behaviour

Before and possibly during the animation you will want to define the movement behaviour of your Sprites. The following code snippet cover most of the basic behavioural parameters you will be dealing with.

```
   ' set the sprite's location
FN SWSetSpriteLocation(@mySprite, curX, curY)

   ' set how often a sprite moves in milliseconds
FN SWSetSpriteMoveInterval(@mySprite, 30);

   ' set the sprite's movement direction/distance
FN SWSetSpriteMoveDelta(@mySprite, 10, 5)

These parameters are set initially with SWInitSprite or
SWSpriteFromPict, but may be changed at any time.
```

Preparing the Background

Prepare the Background Frame and Load Frame for the first frame of animation.

```
   ' prepare background for animation
FN SWRefreshBackground(@SpriteWorld)
```

Driving The Animation

Once everything is in place the animation is driven by repeated calls to SWProcessSpriteWorld and SWAnimateSpriteWorld. SWProcessSpriteWorld runs through each Sprite installed in each SpriteLayer in the SpriteWorld and advances the position of the Sprite’s destination rectangle according to each Sprite’s movement characteristics. SWAnimateSpriteWorld draws the each Sprite that needs to be drawn on the screen. Rigorous checking is done on each Sprite to determine if it really needs to be drawn since a considerable time savings is gained by skipping even one small sprite.

```
   ' core animation loop
WHILE (animationIsRunning)
         ' move the sprites to their new positions
   FN SWProcessSpriteWorld(@SpriteWorld)

         ' render a frame of the animation
   FN SWAnimateSpriteWorld(@SpriteWorld)
WEND
```

Detecting Collisions

Since collision detection is such an application specific problem, FBSpriteWorld employs a very simple, but effective and easy to use, collision detection mechanism.

The SWCollideSpriteLayer function is used to check the Sprites in the source SpriteLayer, against the Sprites in the destination SpriteLayer for collisions. In order to check for collisions between the Sprites of a single SpriteLayer, you must pass the same SpriteLayer as the source and the destination.

```
   ' see if our ship has collided with any enemies
FN SWCollideSpriteLayer(@shipSpriteLayer,  @enemySpriteLayer)

   ' we may want to see if any of the enemies
   ' have collided with each other
FN SWCollideSpriteLayer(@enemySpriteLayer,  @enemySpriteLayer)
```

When a collision is detected the collision routine, if any, of the source sprite is called. For anything useful to happen you must install a collision routine in the Sprites you expect to be involved in collisions.

A collision is defined as the condition that occurs when the rectangle that defines the current screen location of a Sprite intersects the corresponding rectangle of another Sprite. This may or may not mean that the actual images of the Sprites as they appear on screen overlap. Therefore it is up to the collision routine you provide to determine a more precise definition of a collision for your Sprites if necessary.

FBSpriteWorld Reference

This section serves as a reference to the routines FBSpriteWorld provides. There are routines for manipulating each of the four core data structures, SpriteWorlds, SpriteLayers, Sprites, and Frames. Some utility routines are also provided.

SWEnterSpriteWorld

This function initializes the FBSpriteWorld package.

```
err = FN SWEnterSpriteWorld
```

DESCRIPTION

The SWEnterSpriteWorld function is used to initialize the FBSpriteWorld package. SWEnterSpriteWorld performs some checks to see if FBSpriteWorld can run, and then sets up some internal data structures. You must call SWEnterSpriteWorld before calling any other FBSpriteWorld routine.

SWEnterSpriteWorld returns an error code if initialization fails, other wise it returns _noErr.

SWCreateSpriteWorld

This function will create a new SpriteWorld containing the frames to be used for the screen, background, and work area.

```
err = FN  SWCreateSpriteWorld(SpriteWorldP&,wPort&,swRect,bk GrndPict&)
```

SpriteWorldP&	Pointer to a previously dimmed SpriteWorldRec[ord] variable.
wPort&	The grafPort in which you are creating the SpriteWorld .
swRect	The bounding rectangle of the SpriteWorld.
bkGrndPict&	A handle to a PICT to use as the background.

DESCRIPTION

The SWCreateSpriteWorld function is used to create a new SpriteWorld . It attempts to create the backFrame and loadFrame offScreen grafPorts, based on the current screen depth. It returns _noErr if successful, otherwise returns _swOutOfMemory.

SWCreateSWFromWindow

This function creates a new SpriteWorld containing the frames to be used for the screen, background, and work area, based on a window's grafPort.

```
err = FN SWCreateSWFromWindow(SpriteWorldP&, wPort&)
```

SpriteWorldP&	Pointer to a variable of type SpriteWorldRec.
wPort&	Pointer to the grafPort of the window in which you are creating the SpriteWorld.

DESCRIPTION

The SWCreateSWFromWindow function is used to create a new SpriteWorld from a window. It uses the window's grafPort record to set the boundsRect of the SpriteWorld, and to create the backFrame and loadFrame , based on the current screen depth. It returns _noErr if successful, otherwise returns _swOutOfMemory.

SWDisposeSpriteWorld

This will dispose of an existing SpriteWorld, releasing the memory it occupies.

```
OSErr = FN SWDisposeSpriteWorld(SpriteWorldP&)
```

SpriteWorldP&   Pointer to a SpriteWorld to be disposed.

DESCRIPTION

The SWDisposeSpriteWorld function is used to dispose of a SpriteWorld previously created using SWCreateSpriteWorld . The memory occupied by the background and work frames will be released, as will the memory used by any Sprites contained in the SpriteWorld. Returns OSErr.

SWAddLayerTo World

This function will add a SpriteLayer to a SpriteWorld.

```
err = FN SWAddLayerToWorld(SpriteWorldP&,spriteLayerP&)
```

SpriteWorldP&   Pointer to a SpriteWorld to which the layer will be added.
spriteLayerP&   Pointer to the SpriteLayer to add.

DESCRIPTION

The SWAddLayerToWorld function is used to add a previously created SpriteLayer to a SpriteWorld. A world can contain any number of layers, up to _m axLayers. Once a layer is added to a world, it becomes an active part of the animation. Any sprites in the layer will be processed and drawn when the next frame of the animation is rendered. Returns _swTooManyLayers if an attempt is made to add more than _maxLayers to the SpriteWorld.

SWRemoveLayer

This function will remove a SpriteLayer from a SpriteWorld.

```
FN SWRemoveLayer(SpriteWorldP&, spriteLayerP&)
```

SpriteWorldP&   Pointer to a SpriteWorld from which the SpriteLayer is to be removed.
spriteLayerP&   Pointer to SpriteLayer to remove.

DESCRIPTION

The SWRemoveSpriteLayer function is used to remove a SpriteLayer from a SpriteWorld. This is done when you want to remove an entire layer of sprites from the animation. The sprites in the layer that is removed will not be processed or drawn when the next frame of the animation is rendered.

SPECIAL CONSIDERATIONS

Removing the layer will not erase the sprites where they are on the screen. If you wish the sprites in the layer to disappear and the animation to continue, you must first set the sprite’s visibility to false, render a frame of the animation, and then remove the layer from the world.

SEE ALSO

SWSetSpriteVisibile

SWUpdateSpriteWorld

The function will draw the current frame of the animation in reponse to an update event.

```
FN SWUpdateSpriteWorld(SpriteWorldP&)
```

SpriteWorldP&   Pointer to a SpriteWorld to be updated.

DESCRIPTION

The SWUpdateSpriteWorld function is used to copy the current load frame to the window. You will typically call this function when the window in which your animation is running receives an update event.

SWProcessSpriteWorld

This function processes all the Sprites in a FBSpriteWorld, updating their positions, resetting their timers, calling their custom move and frame procs, etc.

```
FN SWProcessSpriteWorld(SpriteWorldP&)
```

SpriteWorldP&   Pointer to a SpriteWorld to be processed.

DESCRIPTION

The SWProcessSpriteWorld function is used to perform all the automatic processing of every Sprite in the SpriteWorld. This includes updating the Sprites' positions, resetting their movement and frame change timers, and calling their custom move and frame routines, if any. This function, in conjunction with SWAnimateSpriteWorld, drives the animation.

This function does no drawing, it simply processes a sprite in terms of its movement and frame changing characteristics using the parameters you specify when setting up the sprite.

SEE ALSO

SWAnimateSpriteWorld

SWAnimateSpriteWorld

This function will render a frame of the animation using the frame differential technique.

```
FN SWAnimateSpriteWorld(SpriteWorldP&)
```

SpriteWorldP&   Pointer to a SpriteWorld to be animated.

DESCRIPTION

The SWAnimateSpriteWorld function is used to render a frame of the animation by drawing all the Sprites in the specified SpriteWorld in their new positions. You will typically call this function right after SWProcessSpriteWorld in your main animation loop. This function marks all the sprites as no longer in need of drawing, so that next time around if the sprite has not been moved or otherwise changed in any way, it will not be drawn again unnecessarily.

SWAddSpriteToLayer

This function will add an existing Sprite to a SpriteLayer.

```
FN SWAddSpriteToLayer(SpriteLayerP&, newSpriteP&)
```

spriteLayerP&	Pointer to an existing SpriteLayer.
newSpriteP&	Pointer to a Sprite to be added to the specified SpriteLayer.

DESCRIPTION

The SWAddSpriteToLayer function is used to add an existing Sprite to a SpriteLayer.

SWRemoveSprite

This function will remove a sprite from a layer.

```
FN SWRemoveSprite(SpriteLayerP&, oldSpriteP&)
```

spriteLayerP&	Pointer to an existing SpriteLayer.
oldSpriteP&	Pointer to a Sprite to be removed from the specified SpriteLayer.

DESCRIPTION

The SWRemoveSprite function is used to remove a Sprite from a SpriteLayer . This is done when you want to remove the sprite from the animation. The sprite that is removed will not be processed or drawn when the next frame of the animation is rendered.

SPECIAL CONSIDERATIONS

Removing a sprite from a layer will not erase the sprite where it is on the screen. If you want the sprite to disappear and the animation to continue, you must first set the sprite’s visibility to false, render a frame of the animation, and then remove the sprite from the layer .

SWCollideSpriteLayer

This function will check for collisions between two SpriteLayers.

```
FN SWCollideSpriteLayer(srcSpriteLayerP&,
        dstSpriteLayerP&)
```

srcSpriteLayerP&	Pointer to a SpriteLayer containing one or more Sprites.
dstSpriteLayerP&	Pointer to another SpriteLayer containing one or more Sprites.

DESCRIPTION

The SWCollideSpriteLayer function is used to check the Sprites in the source SpriteLayer, against the Sprites in the destination SpriteLayer for collisions. In order to check for collisions between the Sprites of a single SpriteLayer, you must pass the same SpriteLayer as the source and the destination.

When a collision is detected the collision routine, if any, of the source sprite is called. For anything useful to happen you must install a collision routine in the Sprites you expect to be involved in collisions.

A collision is defined as the condition that occurs when the rectangle that defines the current screen location of a Sprite intersects the corresponding rectangle of another Sprite. This may or may not mean that the actual images of the Sprites as they appear on screen overlap. Therefore it is up to the collision routine you provide to determine a more precise definition of a collision for your Sprites if necessary.

SEE ALSO

SWSetCollideProc.

SWInitSprite

This function will create a new sprite with no frames.

```
                FN SWInitSprite(newSpriteP&, currentFrame, curRectP&, ->
boundsRectP&, deltaX, deltaY, frameAdvance)
```

newSpriteP&	Pointer to a newly created Sprite.
currentFrame	Index to the current frame.
curRectP&	Pointer to the current location rectangle.
boundsRectP&	Pointer to the bounds rectangle for this Sprite.
deltaX	Number of pixels to move the Sprite's horizontal position each time SWProcessSpriteWorld is called.
deltaY	Number of pixels to move the Sprite's vertical position each time SWProcessSpriteWorld is called.
frameAdvance	Number to increment (or decrement, if frameAdvance is negative) the Sprite's current frame index each time SWProcessSpriteWorld is called.

DESCRIPTION

The SWInitSprite function will create and initialize a new Sprite with an empty frame set. For the Sprite to be of any use, you must create and add some Frames. This function is called by SWSpriteFromPict, but may be called directly as well.

SWSpriteFromPict

This function will create a new sprite with no frames, based on the specified PICT resource.

```
FN SWSpriteFromPict(newSpriteP&, currentFrame, curX, curY,  ->
  boundsRectP&, deltaX, deltaY, frameAdvance, pictID)
```

newSpriteP&	Pointer to a newly created Sprite.
currentFrame	Index to the current frame.

curX Current horizontal location in local coordinates

curY Current vertical location in local coordinates

boundsRectP& Pointer to the bounds rectangle for this Sprite.

deltaX Number of pixels to move the Sprite's horizontal position each time SWProcessSpriteWorld is called.

deltaY Number of pixels to move the Sprite's vertical position each time SWProcessSpriteWorld is called.

frameAdvance Number to increment (or decrement, if frameAdvance is negative) the Sprite's current frame index each time SWProcessSpriteWorld is called.

pictID Number of a PICT resource (typically the PICT used for the first Frame of animation).

DESCRIPTION

The SWSpriteFromPict function will create and initialize a new Sprite with an empty frame set. This function does not attach the PICT to the Sprite as a Frame; rather, it uses the PICT frameRect to size the Sprite's currentRect field, offsetting it to curX,curY. For the Sprite to be of any use, you must create and add some Frames.

SPECIAL CONSIDERATIONS

For optimum animation speed, the Sprite's currentRect field should be the same size as the PICT resource used for the current Frame. Typically, all Frames attached to the Sprite will be the same size. If you pass the ID of the PICT used as the first Frame to SWSpriteFromPict, you will always have a properly sized currentRect. If you add Frames of different sizes to a Sprite, be sure to change size of the currentRect field using SWSetCurRect before rendering the Frames.

SWCloneSprite

This function will create a duplicate of an existing Sprite.

```
FN SWCloneSprite(srcSpriteP&, newSpriteP&)
```

srcSpriteP& Pointer to an existing Sprite to be cloned.

newSpriteP& Pointer to the newly created Sprite.

DESCRIPTION

The SWCloneSprite function will create a duplicate of an existing Sprite. The frame set of the Sprite is not duplicated; instead, both Sprites share the same frame set. NewSprite must have been previously dimensioned.

SPECIAL CONSIDERATIONS

Since these Sprites share their frames you must be careful when disposing of these Sprites not to dispose of the frame set twice. To help determine if the Sprite is a clone, SWCloneSprite sets the _isClone field of the Sprite Record to _zTrue for each new Sprite that it creates.

SWDisposSprite

This function will dispose of an existing Sprite, releasing the memory it occupies.

```
osErr = FN SWDisposSprite(deadSpriteP&, disposeFrames)
```

deadSpriteP&	A Sprite to be disposed.
disposeFrames	A boolean value indicating whether or not the frames of the Sprite should be automatically disposed along with the Sprite.

DESCRIPTION

The SWDisposSprite function will dispose of a Sprite, and optionally dispose of the frames used by the Sprite.

The disposeFrame parameter is typically used when you have a number of Sprites that share the same frames. In order to avoid disposing of the frames twice you will want to dispose of the first Sprite that owns the frames by passing true into disposeFrames, subsequent Sprites should be disposed of by passing false into disposeFrames.

This function is called by SWDisposSpriteWorld, but may be called anytime during a running animation to free memory for Sprites no longer in use.

SWAddFrame

This function will add a frame to an existing Sprite.

```
err = FN SWAddFrameToSprite(srcSpriteP&, newFrameP&)
```

srcSpriteP&	Pointer to an existing Sprite.
newFrameP&	Pointer to a new frame to be added to the Sprite.

DESCRIPTION

The SWAddFrameToSprite function will add a new frame to an existing Sprite. This frame may also be added to other Sprites so that they may share frames, thus saving memory. Returns _swTooManyFrames if you attempt to add more than _maxFrames to a Sprite.

SWRemoveFrame

This function will remove a Frame from an existing Sprite.

```
FN SWRemoveFrame(srcSpriteP&, oldFrameP&)
```

srcSpriteP&	Pointer to an existing Sprite.
oldFrameP&	Pointer to a Frame to be removed from the Sprite.

DESCRIPTION

The SWRemoveFrame function will remove a Frame from an existing Sprite. You will probably never want to do this, since you can simply dispose of a Sprite and its Frames automatically when your animation is finished.

SWSetCurrentFrame

This function will set a Sprite’s current frame using the specified index into the Sprite’s Frame list.

```
FN SWSetCurrentFrameIndex(srcSpriteP&, frameIndex)
```

srcSprite	Pointer to an existing Sprite.
frameIndex	An index into the Sprite’s Frame list.

DESCRIPTION

The SWSetCurrentFrame function will set a Sprite’s current frame using the specified index into the Sprite’s Frame list. The current Frame will be rendered in the animation at the Sprite’s current location.

SWSetSpriteFrameAdv

This function will set the value by which the current frame index of the Sprite will be automatically incremented.

```
FN SWSetSpriteFrameAdv(srcSpriteP&, frameAdvance)
```

srcSpriteP&	Pointer to an existing Sprite.
frameAdvance	The value by which the current frame index will be automatically incremented.

DESCRIPTION

The SWSetSpriteFrameAdv function allows you to specify the value by which the current frame index of the Sprite will be automatically incremented. The Sprite’s current frame index will be automatically incremented when the Sprite is processed by the SWProcessSpriteWorld function.

SWSetFrameRange

This function specifies the range of frame indexes within which the current Frame of the Sprite will be advanced.

```
FN SWSetFrameRange(srcSpriteP&, firstFrameIndex,
        lastFrameIndex)
```

srcSpriteP&	Pointer to an existing Sprite.
firstFrameIndex	A value indicating the index of the first Frame in the range.
lastFrameIndex	A value indicating the index of the last Frame in the range.

DESCRIPTION

The SWSetFrameRange function is used to specify the range of frame indexes within which the current Frame of the Sprite will be advanced. This allows you use a subset of a Sprite’s Frames to be animated. The current Frame of the Sprite will be automatically advanced when the Sprite is processed by SWProcessSpriteWorld.

SPECIAL CONSIDERATIONS

SWInitSprite, SWAddFrameToSprite, and SWDisposFrame set the frame range from 0 to totalFrames. Therefore, if you are using all of a Sprite's Frames in the current animation, you do not need to explicitly set the frame range using this function.

SEE ALSO

SWSetSpriteFrameAdv

SWSetFrameTime

This function sets the time interval between automatic advances of the Sprite’s current frame.

```
FN SWSetFrameTime(srcSpriteP&, timeInterval)
```

srcSpriteP& Pointer to an existing Sprite.

timeInterval A value indicating the millisecond time interval between Frame advances. A value of 0 indicates the frame advance should happen as often as possible. A value of -1 indicates the frame advance should never happen.

DESCRIPTION

The SWSetSpriteFrame function is used to set the time interval between automatic advances of the Sprite’s current frame. To advance the current Frame as quickly as possible pass 0 into the timeInterval parameter. The current Frame of the Sprite will be automatically advanced when the Sprite is processed by SWProcessFBSpriteWorld after the specified time interval has passed.

SEE ALSO

SWSetSpriteFrameAdv

SWSetFrameChangeProc

This function set the routine to be called when the Sprite’s current is advanced.

```
FN SWSetFrameChangeProc(srcSpriteP&, frameProcP&)
```

srcSpriteP& Pointer to an existing Sprite.

frameProcP& Pointer to a procedure to be called when the current Frame is advanced.

DESCRIPTION

The SWSetFrameChangeProc function is used to specify a routine to be called when the current Frame of the Sprite is to be advanced. This routine could do some additional processing in order to determine which Frame of the Sprite should be made current.

The routine you pass must be of type ENTERPROC/EXITPROC which is defined like so…

```
"MyFrameChangeProcedure"
ENTERPROC(srcSpriteP&, curFrame)
```

srcSpriteP& A Sprite being processed by SWProcessFBSpriteWorld.

curFrame A value indicating the index of the current Frame. Your function may change this value indicating a different Frame to be made current.

```
EXITPROC
RETURN
```

SWMoveSpriteRect

This function will move a Sprite to a new rectangle.

```
FN SWMoveSpriteRect(srcSpriteP&, newRectP&)
```

srcSpriteP& Pointer to a Sprite to be moved.

newRectP& A pointer a rectangle representing the Sprite's new location.

DESCRIPTION

The SWMoveSpriteRect function is used to move a Sprite’s current position to an absolute coordinate bounded by newRect. Unlike SWSetCurRect, the Sprite’s last position is remembered so that when the animation is rendered the Sprite will be properly erased from its last known position.

If the Sprite has a movement routine installed, it will not be called by this function.

SEE ALSO

SWSetCurRect, SWOffsetSprite, SWSetSpriteLocation, SWMoveSprite

SWMoveSprite

This function will move a Sprite’s current position to an absolute horizontal, and vertical coordinate.

```
FN SWMoveSprite(srcSpriteP&, horizLoc, vertLoc)
```

srcSpriteP& Pointer to a Sprite to be moved.

horizLoc A value indicating the absolute horizontal coordinate to which the Sprite will be moved.

vertLoc A value indicating the absolute vertical coordinate to which the Sprite will be moved.

DESCRIPTION

The SWMoveSprite function is used to move a Sprite’s current position to an absolute horizontal, and vertical coordinate. The Sprite’s last position is remembered so that when the animation is rendered the Sprite will be properly erased from its last known position.

If the Sprite has a movement routine installed, it will not be called by this function.

SEE ALSO

SWSetCurRect, SWOffsetSprite, SWSetSpriteLocation, SWMoveSprite

SWOffsetSprite

This function will offset the Sprite’s current position to a relative horizontal, and vertical coordinate.

```
FN SWOffsetSprite(srcSpriteP&, horizDelta, vertDelta)
```

srcSpriteP& Pointer to a Sprite to be moved.

horizDelta A value indicating the relative horizontal coordinate by which the Sprite will be offset.

vertDelta A value indicating the relative vertical coordinate by which the Sprite will be offset.

DESCRIPTION

The SWOffsetSprite function is used to offset a Sprite’s current position to a relative horizontal, and vertical coordinate. The Sprite’s last position is remembered so that when the animation is rendered the Sprite will be properly erased from its last known position.

If the Sprite has a movement routine installed, it will not be called by this function.

SEE ALSO

SWSetCurRect, SWMoveSprite, SWSetSpriteLocation, SWMoveSpriteRect

SWSetSpriteLocation

This function will set a Sprite’s current and last known position to an absolute horizontal, and vertical coordinate.

```
void SWSetSpriteLocation(srcSpriteP&, horizLoc, vertLoc)
```

srcSpriteP& Pointer to a Sprite to be moved.

horizLoc A value indicating the absolute horizontal coordinate to which the Sprite’s current and last known position will be set.

vertLoc A value indicating the absolute vertical coordinate to which the Sprite’s current and last known position will be set.

DESCRIPTION

The SWSetSpriteLocation function is used to set a Sprite’s current and last known position to an absolute horizontal, and vertical coordinate. The Sprite will not be erased from wherever it was before this function was called. You will typically use this function before the animation starts or when introducing a new Sprite into the animation, to set the Sprite initial position.

If the Sprite has a movement routine installed, it will not be called by this function.

SEE ALSO

SWSetCurRect, SWOffsetSprite, SWMoveSprite, SWMoveSpriteRect

SWSetCurRect

This function will set a Sprite's current and last known position to a new rectangle.

```
void SWSetSpriteLocation(srcSpriteP&, newRectP&)
```

srcSpriteP&	Pointer to a Sprite to be moved.
newRectP&	Pointer to the new rectangle which current and old position will be set to

DESCRIPTION

The SWSetSpriteLocation function is used to set a Sprite's current and last known position to an absolute horizontal, and vertical coordinate. The Sprite will not be erased from wherever it was before this function was called. You will typically use this function before the animation starts or when introducing a new Sprite into the animation, to set the Sprite initial position.

If the Sprite has a movement routine installed, it will not be called by this function.

SEE ALSO

SWMoveSprite, SWOffsetSprite, SWSetSpriteLocation, SWMoveSpriteRect

SWSetMoveBounds

This function will set a Sprite's movement boundary rectangle.

```
FN SWSetMoveBounds(srcSpriteP&, boundsRectP&)
```

srcSpriteP&	Pointer to an existing Sprite.
boundsRectP&	Pointer to a rectangle describing the Sprite's movement boundary.

DESCRIPTION

The SWSetSpriteMoveBounds function is used to specify a movement boundary rectangle for a Sprite. Enforcement of this movement boundary is left to the Sprite's movement routine provided by you. You may want to use this rectangle as an area around which the Sprite might bounce, or wrap, or some other complex movement behavior

SWSetMoveDelta

This function sets the values by which the Sprite's current position will be automatically offset.

```
FN SWSetMoveDelta(srcSpriteP&, horizDelta, vertDelta)
```

srcSpriteP&	Pointer to an existing Sprite.
horizDelta	A value by which the current horizontal position will be automatically offset.
vertDelta	A value by which the current vertical position will be automatically offset.

DESCRIPTION

The SWSetMoveDelta function is used to specify the values by which the Sprite's current position will be automatically offset. The horizontal and vertical deltas indicate the direction and distance the Sprite will be moved when the Sprite is processed by SWProcessSpriteWorld.

SWSetMoveTime

This function sets the time interval between movements of the Sprite.

```
FN SWSetMoveTime(srcSpriteP&, timeInterval)
```

srcSpriteP&	Pointer to an existing Sprite.
timeInterval	A value indicating the millisecond time interval between movements of the Sprite.

DESCRIPTION

The SWSetMoveTime function is used to specify a millisecond time interval between movements of the Sprite. The Sprite will be automatically moved when it is processed by the SWProcessSpriteWorld function after the specified time interval has passed.

SWSetSpriteMoveProc

This function sets a Sprite’s movement routine to be called when the Sprite is automatically moved.

```
FN SWSetSpriteMoveProc(srcSpriteP&, moveProcP&)
```

srcSpriteP&	Pointer to an existing Sprite.
moveProcP&	Pointer to a procedure to be called when the Sprite is moved.

DESCRIPTION

The SWSetSpriteMoveProc function is used to specify a routine to be called when the Sprite is automatically moved. The Sprite will be automatically moved when it is processed by the SWProcessSpriteWorld function.

The routine you pass must be of type ENTERPROC/EXITPROC which is defined like so…

```
"MyMovementProcedure"
ENTERPROC(spriteWorldP&, spriteP&, curRectP&)
```

spriteWorldP&	Pointer to SpriteWorld containing Sprite
spriteP&	Pointer to Sprite being moved.
curRectP&	Pointer to a Sprite's current rectangle.

```
EXITPROC
RETURN
```

SWBounceMoveProc

This procedure can be installed as a Sprite’s movement routine to make the Sprite bounce around the screen.

```
"SWBounceMoveProc"
ENTERPROC(spriteWorldP&, srcSpriteP, curRectP&)
```

spriteWorldP&	Pointer to SpriteWorld containing Sprite.
srcSpriteP&	Pointer to Sprite being moved.
curRectP&	Pointer to the Sprite's current rectangle.

DESCRIPTION

The SWBounceMoveProc procedure is provided for use as a movement routine for a Sprite. When this procedure is installed as a Sprite’s movement routine, the Sprite will bounce around inside the area defined by its movement boundary rectangle.

SEE ALSO

SWSetMoveBounds

SWWrapMoveProc

This procedure can be installed as a Sprite’s movement routine to make the Sprite wrap from one side of the screen to the other.

```
"SWWrapMoveProc"
 ENTERPROC(spriteWorldP&, srcSpriteP, curRectP&)
```

spriteWorldP&	Pointer to SpriteWorld containing Sprite.
srcSpriteP&	Pointer to Sprite being moved.
curRectP&	Pointer to the Sprite's current rectangle.

DESCRIPTION

The SWWrapSpriteMoveProc procedure is provided for use as a movement routine for a Sprite. If this function is installed as a Sprite’s movement routine, and the Sprite moves outside the area defined by its movement boundary rectangle, it will be wrapped to the other side.

SEE ALSO

SWSetMoveBounds

SWSetCollideProc

This function sets a Sprite’s collision routine, to be called when the Sprite is involved in a collision with another.

```
FN SWSetCollideProc(srcSpriteP&, collideProcP&)
```

srcSpriteP&	Pointer to an existing Sprite.
collideProcP&	Pointer to a new collision routine.

DESCRIPTION

The SWSetCollideProc function is used to specify a collision routine, to be called when the Sprite is involved in a collision with another. This routine may do some futher processing to determine if the Sprites have actually collided, and if so, perform some action such as playing an explosion sound.

The routine you pass must be of type ENTERPROC/EXITPROC which is defined like so…

```
"MyMovementProcedure"
ENTERPROC(srcSpriteP&, destSpriteP&, sectRectP&)
```

srcSpriteP&	Pointer to Sprite whose collision routine is being called.
destSpriteP&	Pointer to Sprite being collided with.
sectRectP&	Pointer to rectangle representing the intersection of the source Sprite's currentRect and the destination Sprite's currentRect.

```
EXITPROC
RETURN
```

SEE ALSO

SWCollideSpriteLayer.

SWSetSpriteVisible

This function sets the visibility of a Sprite.

```
FN SWSetSpriteVisible(srcSpriteP&, isVisible)
```

srcSpriteP&	Pointer to an existing Sprite.
isVisible	A boolean value specifying the visible state of the Sprite.

DESCRIPTION

The SWSetSpriteVisible function is used to specify whether a Sprite that taking part in the animation should actually be drawn. The result of calling this function is reflected when the animation is rendered using SWAnimateSpriteWorld.

SWFrameFromPict

This function creates a Frame from a PICT resource

```
err = SWFrameFromPict(srcFrameP&, pictID)
```

srcFrameP&	Pointer to a previously dimensioned Frame variable.
pictID	Number of a PICT resource.

DESCRIPTION

The SWFrameFromPict function creates a new Frame from the specified PICT resource. It creates two off screen pixel maps, one for the image and one for the mask. It then copies the image and mask to the off screen maps and releases the PICT from memory (the two pixel maps remain in memory until released with SWDisposFrame). Returns _swOutOfMemory if unable to create off screen pixel maps, or RESERROR if there is a problem with pictID.

SPECIAL CONSIDERATIONS

PictID should actually point to the first of three PICTs used by SWFrameFromPict. These three PICTs should be stored in your resource file as follows:

pictID	Color image
pictID+1	Black & white image
pictID+2	Mask image

SWFrameFromPict will determine the current color depth of the display and load the appropriate image.

SWSetBackgroundPict__

Sets the background PICT of the specified SpriteWorld.

```
FN SWSetBackgroundPict(spriteWorldP&, pictHdl&)
```

spriteWorldP&	Pointer to the SpriteWorld receiving the new background
pictHdl&	Handle to the new background PICT.

DESCRIPTION

The SWSetBackgroundPict function attaches a new background PICT to the SpriteWorld. SWCreateSpriteWorld initially sets the SpriteWorld's background PICT, but you can use this function to change the background on the fly.

SPECIAL CONSIDERATIONS

This function does not update the backFrame or loadFrame of the SpriteWorld, and thus does not by itself change what appears on the screen. Follow this function with SWRefreshBackground to render the new background .

SWRefreshBackground

Intitalizes or updates the SpriteWorld's background by coping its background PICT to the backFrame and loadFrame.

```
FN SWRefreshBackground(spriteWorldP&)
```

spriteWorldP&	Pointer to a previously defined SpriteWorld

DESCRIPTION

The SWRefreshBackground function copies a SpriteWorld's current background PICT to the backFrame and loadFrame. This is required before rendering the first frame of animation , and may be called during a running animation to change the background on the screen . In the latter case, you must precede SWRefreshBackground with SWSetBackgroundPict.

SPECIAL CONSIDERATIONS

If you change backgrounds on the fly, any stationary Sprites whose frames have not changed will not be redrawn. You should call SWSetSpriteVisible, with the IsVisible parameter set to _zTrue, at the same time you call SWRefreshBackground to make sure these Sprites appear over your new background.

FunctionName

Short description here.

```
Function (param1, param2)
```

param1	Describe the parameters here.
param2	Describe the parameters here.

DESCRIPTION

Describe the function here.

SPECIAL CONSIDERATIONS

Describe special considerations here.

Summary of FBSpriteWorld

```
DEFINT A-Z                    'Default variable type = integer

'--------------------------------------------------------------------- --------------
'Constants
'--------------------------------------------------------------------- --------------

_maxLayers=10                 'maximum Layers per World
_maxSprites=10                  'maximum Sprites per Layer
_maxFrames=10                 'maximum Frames per Sprite
_layerListSize=40                  '4 bytes * _maxLayers
_spriteListSize=40                 '4 bytes * _maxSprites
_frameListSize=40                  '4 bytes * _maxFrames
```

```
_swTooManyLayers=1                'error codes
_swTooManySprites=2
_swTooManyFrames=3
_swNotSystemSeven=4
_swTimeMgrNotPresent=5
_swOutOfMemory=6

'--------------------------------------------------------------------- --------------
'Records (Data Structures)
'-----------------------------------------------------------------------------------

DIM RECORD SpriteWorldRec         'SPRITEWORLD DATA STRUCTURE
 DIM boundsRect.8                'bounding Rect of SpriteWorld
 DIM totalLayers                 'number of Layers in this World
 DIM layerList.layerListSize        'list of Layer pointers
 DIM windowFramePtr&              'pointer to on-screen grafPort
 DIM backFramePtr&                'pointer to background Frame grafPort
 DIM loadFramePtr&                'pointer to loadFrame grafPort
 DIM backPictHdl&                'handle to background PICT
DIM END RECORD.SpriteWorldRec

DIM RECORD SWLayerRec             'LAYER DATA STRUCTURE
 DIM totalSprites                'number of Sprites in this Layer
 DIM spriteList.spriteListSize      'list of Sprite pointers
DIM END RECORD.SWLayerRec

DIM RECORD SWSpriteRec            'SPRITE DATA STRUCTURE
 DIM currentFrameNum              'index to current Frame
 DIM currentRect.8                'current location Rect
 DIM oldRect.8                 'previous location Rect
 DIM deltaRect.8                 'union of currentRect & oldRect
 DIM sBoundsRect.8                'bounding Rect of Sprite
 DIM isVisible                 'boolean visible flag
 DIM needsToBeDrawn               'flag to draw Sprite
 DIM needsToBeErased              'flag to erase Sprite
 DIM drawPartial                 'flag to draw only part of Sprite
 DIM totalFrames                 'number of Frames in Sprite
 DIM frameList.frameListSize        'list of Frame pointers
 DIM firstFrameIndex              'index of first Frame to render
 DIM lastFrameIndex               'index of last Frame to render
 DIM xDelta                'pixels to move horizontally
 DIM yDelta                'pixels to move vertically
 DIM maskRegionHdl&               'handle to MaskRegion
 DIM frameAdvance                'value to advance Frames
 DIM frameTimeTask.tmXQSize         'Frame Time Task
 DIM frameTTHasFired              'flag if Time Task has fired
 DIM frameTimeInterval            'milliseconds between Frame changes
 DIM frameChangeProcPtr&          'Frame change procedure ptr
 DIM moveTimeTask.tmXQSize        'Move Time Task
 DIM moveTTHasFired               'flag if Move Task has fired
 DIM moveTimeInterval             'milliseconds between moves
 DIM moveProcPtr&                'movement procedure ptr
 DIM collideProcPtr&              'collide procedure ptr
 DIM isClone                  'flag indicates if cloned
DIM END RECORD.SWSpriteRec

DIM RECORD SWFrameRec              'FRAME DATA STRUCTURE
 DIM imageMapPtr&                'pointer to image grafPort
 DIM maskMapPtr&                 'pointer to mask grafPort
 DIM fBoundsRect.8                 'bounding Rect of Frame
DIM END RECORD.SWFrameRec

'--------------------------------------------------------------------- --------------
'Globals
'--------------------------------------------------------------------- --------------

DIM gColorDepth                  'current screen color depth

END GLOBALS
```