A Gentle Introduction To SuperSpriteSurface for REALbasic

Tutorial 1

by Thomas Cunningham
mauitom@maui.net

February 2006

Summary Of This Document

This powerful alternative to the build-in classes for using sprite and sprite animations in your REALbasic applications. The demo's that come with SuperSpriteSurface (SSS) are excellent, but may intimidate someone new to programming, so consider this tutorial and associated demo projects as a kinder introduction to SSS.

Platform Supported

As of this writing, the SuperSpriteSurface (SSS) is for the Mac platform only. However, John has a Windows version in the works and should be released soon. I'm using Rb 5.5.5 for this project, however, if you are using Rb2005 or Rb2006, everything should work the same. Just keep in mind, my screenshots will not look like your IDE.

The SuperSpriteSurface class is written by John Balestrieri <mrjohn@tinrocket.com> and is available from John's web site http://www.tinrocket.com/. It sells for $29.95. A demo version is available for download. And if you end up using these classes, pay the man!

Assumptions

I will assume that you are somewhat familiar with REALbasic (Rb), but I do include screenshots to help you.

A Sprite Surface

A sprite surface is used in computer programming to, mostly, display moving graphics. SSS, like the built-in Sprite Surface, is used as a two dimensional (2D) display area. It is very similar to the Canvas class in some regards, but has been extended to handle animations of images wrapped in sprites. Here is a definition of a sprite from pcwebopaedia.com.

> *A graphic image that can move within a larger graphic. Animation software that supports sprites enables the designer to develop independent animated images that can then be combined in a larger animation. Typically, each sprite has a set of rules that define how it moves and how it behaves if it bumps into another sprite or a static object.*

A sprite does not live on its own, it must be attached to a sprite surface. Sprites are mainly used in computer games. Scrolling top down and side action games are the most common types of games that use sprites.

The SuperSpriteSurface

So why should you consider purchasing this group of classes when Rb comes with this built-in capability? Simply put, SSS is better than the classes what come with Rb.

Its underlying display technology uses OpenGL (OGL). Most, if not many of today's modern games use OGL to display there screen graphics. It's fast, it's pretty eye candy.

SSS has a powerful framework to detect collisions. Collisions — that is detecting when your sprite say, hits a wall — is a very big deal if you want your game to behave properly. It sounds easy, but detecting and responding to collisions is very complicated.

The SSS Application Programming Interface (API) was purposely designed to be similar to the built-in Rb classes for ease of use. However, don't be fooled, overall, using sprites is not what I would term easy. There are multiple steps to making your sprites and sprite surface look and work properly. Using sprites in computer programming is a huge subject, as a quick search online will reveal. I will only scratch the surface in this tutorial in the hopes that it will get you going with SSS.

SuperSprites

A sprite is really just an image. John names his sprites, SuperSprites! To make a nice game you will need lots of images to use. I will not get in to this at all. In fact, this tutorial and its associated demo Rb projects will not use any external artwork. I will use the drawing functions that come with Rb to use as our image. This keeps things simple so we can concentrate on the basics. But, let's look at one sequence of images that come with SSS as a little background.

Color Image

Mask Image

These are the two shield images John uses in his Knights Demo. Most sprite images will be used this way, that is, one color image and one mask. The mask is used to make the image look pretty and is used to give the image a look of transparency. Again, I'm not going to discuss this subject, but I did want you to see what the concept of a sprite image is about.

Please take note of the black backgrounds of the images and that shades of gray are used in the mask image. You do need to know that much about sprite images. When these images are combined, which you will learn how to do in a few pages, they will appear on the sprites associated surface like this.

The Above Two Images Combined On A Blue Background

That screenshot shows the shield on a SuperSpriteSurface that has a blue background. Notice how John created his mask to get this look. Study it a bit, it's a bit tricky, at least it is for non-artists such as myself.

The SuperSprite has three Events which you will use to animate, move or otherwise interact with you sprite. This is based on the Rb model of an events based API. The SuperSprite class has an assortment of Methods and Properties associated with it. Please refer to the HTML documentation that come with SSS. For our little beginner project, we will only use the Image and Position properties.

The Surface

OK, let's move on to the surface, the SuperSpriteSurface. Again, a sprite must be used, or more formally, attached to a surface to function. The SSS is a subclass of an Rb Rectangle, so it's a RectControl object in Rb. It has three new Events in addition to the Super's Events, FrameworkMessage, NextFrame and Register.

The FrameworkMessage and Register Events are to check and see if you've purchased a license from John.

The NextFrame event represents each time the surface refreshes itself. The elapsedTime parameter of this event lets you know how many seconds has elapsed since it displayed the last "frame". Internally, you get to use one of two modes, threaded or non threaded, to control which mode you want to use to drive the sprite animation sequences. I just suggest you use the threaded mode and leave it at that. When you know what you're doing later, you can experiment with the mode.

The SSS has ten additional properties associated with it. We will just concentrate on two of them for now, the BackColor and Sprites properties. The BackColor should be clear enough, you get to change the color of the surface to whatever you want. The default color will be black. The combined shield image above shows this property set to a blue.

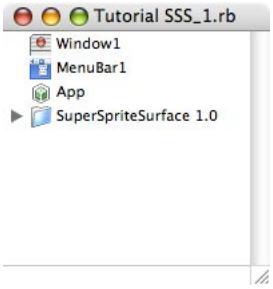
Of the five Methods on SSS, please take note of the Run, Reset and Stop methods. This should give you some indications on the API used to tell SSS what it should be doing.

The Sprites property can be looked at as an array of sorts. It's really a SuperSpriteGroup object, but it helps to simplify the concept if you think of it as the array of sprites attached to the surface. Once you create a sprite and assign it an image, you attach it to the surface via this property. You can always interrogate the surface and retrieve properties of the sprites attached to the surface via the Sprites (SuperSpriteGroup) methods.

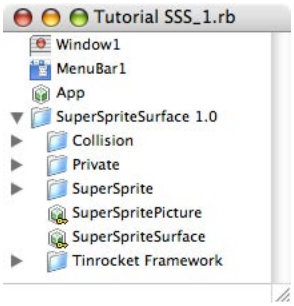
Project One

Step One

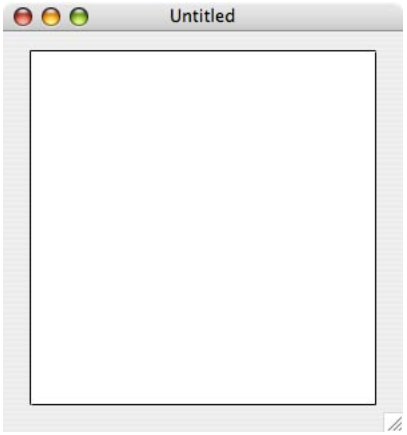
And away we go, let's get started. Begin a new project and drag the SuperSpriteSurface 1.0 folder to your project window.



Click on the disclosure triangle of this folder.



Here you see the guts of the SSS. All you will need from this folder is the SuperSpriteSurface. So open Window1 and drag this class in to Window1. Resize it and position it similar to this.



Run the project and you will see a nice black square in your window. As I mentioned above, the default background color of the surface is black.

Please note something at this stage, the surface is not "running". Remember above when I asked you to observe the methods of the surface? Remember the Run() Method? We have not told the surface to run yet, so it is just sitting there. If you did call the Run method at this point, it would still look the same since we have not attached any SuperSprites to it.

This points out that you can use SSS like a canvas, that is, to display non moving images. Why might you do this? Perhaps just to get the nice look that OpenGL can give you, or perhaps you want to make a slideshow application.

Step Two

OK, let's show something on our SuperSpriteSurface, a SuperSprite. The easiest way to do this is to use a subclass. Go to File:New Class and in the properties window, assign Class1's super to SuperSprite. Rename class1 to TestSuperSprite. Double click TestSuperSprite and click to reveal its Events().

Again, by examining the Events() that John has given you, you can gain some insight in to what a SuperSprite really is. It is defined as an object that can be Constructed, an object that can Collide and an object that has a NextFrame. Simple really.

Let me bring up one thing that tripped me up when I first started with SSS. Please note that both a SuperSprite *and* a SuperSpriteSurface have a NextFrame() Event. Don't use the NextFrame() event of the surface. To me when I first started, it seemed like the place to place code. Wrong. Think in a object oriented fashion. The sprite is the object that should know what it should do as your animation is running, not the surface. The surface is just a container that manages the sprites that are attached to it, not a controller of how its objects should behave, each SuperSprite should own this responsibility.

Step Two A

I want to keep this tutorial as simple as I can think of, so I don't want to involve any artwork. But our sprite needs an image. I purpose that we use the DrawString method on the graphics class to produce our image. It's not very pretty or very interesting to look at, but that's not the point here, we just want to see how SSS operates.

We will create our image in the Constructor() Event( ) of our SuperSprite. We want to build an Rb picture object using a color image and a mask image. We will try and duplicate what we observed above in the shield images. Here is my code along with my comments.

TestSuperSprite Constructor Code:

```
// create a picture to use as a sprite.
Dim colorPic, maskPic As Picture
// create a new Rb picture
colorPic = NewPicture(100,40,32)
// create an all black background
colorPic.Graphics.ForeColor = RGB(0,0,0)
colorPic.Graphics.FillRect 0,0,100,40
// draw some Red text at size 24.
colorPic.Graphics.ForeColor = RGB(200,0,0)
colorPic.Graphics.TextSize = 24
```

```
// must move it down an to the right to see it!
colorPic.Graphics.DrawString "Test", 10, 20

// create the mask at the same size as the colorPic
maskPic = NewPicture(100,40,32)
// as above, create an all black background
maskPic.Graphics.ForeColor = RGB(0,0,0)
maskPic.Graphics.FillRect 0,0,100,40
// this is our white to gray mask color,
maskPic.Graphics.ForeColor = RGB(200, 200, 200)
// draw the same as colorPic
maskPic.Graphics.TextSize = 24
maskPic.Graphics.DrawString "Test", 10, 20
```

Again, nothing fancy for now we have a Rb picture object to work with. We now need to assign this to our SuperSprite Image property. But uh oh, as I look at the SSS documents, this property is not an Rb picture but a SuperSpritePicture. We have one more layer object to learn about and use with SSS.

The SuperSpritePicture class has ten different Methods! - Gulp! Ah, but they are all constructors. John wanted you to be able to create a SuperSpritePicture from as many sources as possible and to be able to use a mask or not. This is very convenient, after all, as I said, a sprite is really just an image, so SSS gives you many options to load up pictures to create SuperSprites - cool.

So let's use one these methods to transform our Rb image to a SuperSpritePicture class and transform it to our SuperSprite Image property. We place this code below what we wrote above.

```
// transform this Rb picture to a ssPicture
Dim ssPicture As SuperSpritePicture
ssPicture = New SuperSpritePicture(colorPic, maskPic TRUE)
// assign it to our image property
Me.Image = ssPicture
```

And that's it, we now have a sprite that our surface can deal with and we have an image that our sprite can deal with. As a note, the True parameter has to do with an internal implementation dealing with OpenGL, setting it to true will make your image look "smooth".

For now, that's it with our SuperSprite class, I will come back and use its NextFrame event a little later in this tutorial.

Our next design consideration is which class should handle *where* our SuperSprite will be placed on our SuperSpriteSurface and which class should create our TestSuperSprite. I will delegate these responsibilities to our surface as part of an initialization process.

Step Three

We move on to Window1 and our SuperSpriteSurface. Let's use our SuperSpriteSurface1 objects Open() event to put all of our pieces together. As I stated early on, learning to use SSS is not easy, not hard, but does take some planning and a few steps. My code and comments from the Open() event will explain the next few steps.

```
// Always Reset your surface
Me.Reset
// Make my background color blue
Me.BackColor = &c33669A
// Make our animation mode threaded
Me.Threaded = True
Me.ClickToStop = False

// Create a TestSuperSprite
Dim tempSprite As New TestSuperSprite
// Our constructor takes care of an image.
// But remember, it does not PLACE the sprite anywhere
tempSprite.Position = New TinrocketVector2D(10, 40)

// we now attach our new sprite to the surface
Me.Sprites.Append(tempSprite)

// Run!
Me.Run
```

A few housekeeping steps at first gets our surface in a state we want it to run. We next create a SuperSprite and position it using a two dimensional vector class that John includes with SSS, although you could always use Rb's 3DVector class, ignoring the Z property or just assign to the X and Y properties of Position.

We attach our sprite to the surface and tell it to run. Feel free to run your project now and see your beautiful work! And be sure and see what happens if you do NOT make that last call for the surface to run.

Step Four

To finish off our first example, lets make our SuperSprite move, after all, that's what sprites are supposed to do! We now will use the NextFrame event of our subclass, TestSuperSprite. This event has a very useful parameter, elapsedTime. If you test to see what value this parameter passes each time it fires, you will find it is a very small number, like around 0.0006 most of the time.

The concept to use this parameter and this event is simple. You scale the number as John does is his demo's. It's just math and fiddling around with the values to get your sprite to move in a manner that you want. An online search on the subject of time based movement is plentiful to learn more on this subject and as mentioned, John has a few nifty examples in his demo's so be sure and study them.

For this project, all I want to do is move our sprite to the right, by adding to the sprites Position.X property at a certain rate. Once the sprite reaches the right side of our surface, I will reposition it on the left side again. Nothing fancy. I place an adjuster factor to make it move at a lazy pace. My NextFrame event code looks like this.

```
Const adjustor = 20
```

```
Me.Position.X = Me.Position.X + (adjustor * elapsedTime)
If Me.Position.X > Me.SuperSpriteSurfaceControl.Width Then
  Me.Position.X  =  - 40
End
```

Run the project and you will see your sprite move along in a nice smooth manner with no flicker.

Winding It Up

That's it for this first installment on how to use SSS. Hopefully, I've got you up and running with this very simple project that explains some of the steps required to create a sprite and move it in a sprite surface. We've only scratched the surface. The follow up to this tutorial will be to implement the Collision() Event of our SuperSprite. The project name for this tutorial is Tutorial SSS_1.rb.