Getting Started Developing a simple SCML Implementation

The purpose of this guide is to get you started with the basics needed to create an implementation of the current version of Spriter. Spriter is currently at release a2, and still in alpha. This guide will be expanded, and eventually replaced with something more comprehensive once development progresses further, and all of the basic features are implemented, and there are sample files available to test implementations with.

This guide will only cover what's needed to implement SCML based animations using only the features that are available in Spriter version a2. It should be possible to infer much of the rest of the implementation requirements from the format specification outline, once you have a firm grasp on what's in this guide.

You can view an outline of the currently implemented subset of the full-format here.

An outline of the complete format specification can be found here.

Developers who attempt to implement features from beyond the scope of this guide are encouraged to ask questions on our forums, or to my email, both of which are listed at the end of this guide.

It is recommended you look over the subset linked above, and have it handy in another tab, or side by side with this guide, to follow along.

Version Info

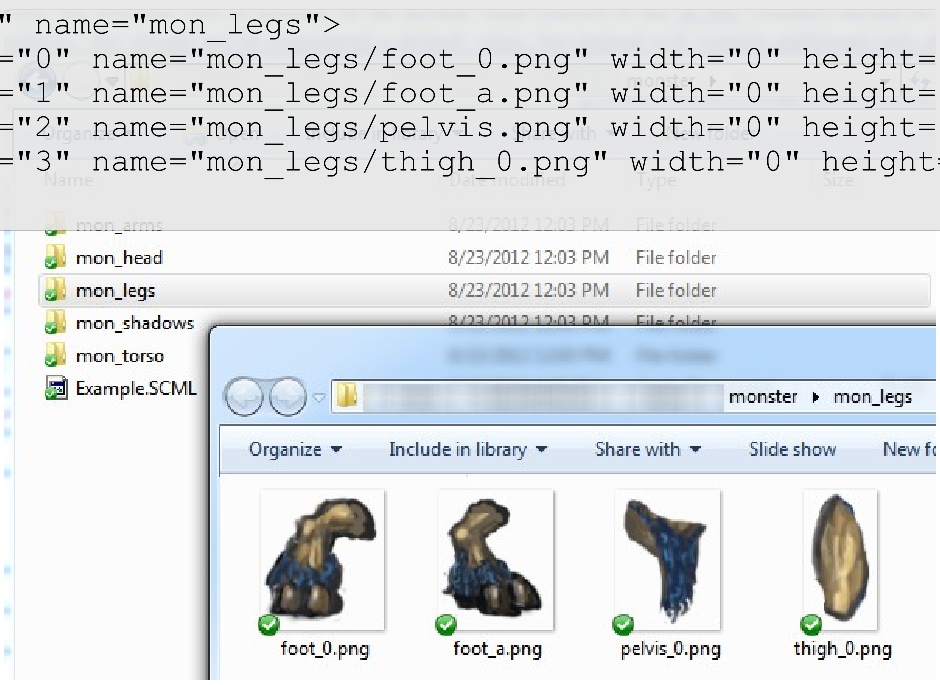
An SCML file begins with basic XML and SCML specific version data.

```
<?xml version="1.0" encoding="UTF-8"?>
<spriter_data scml_version="1.0" generator="BrashMonkey
Spriter" generator_version="a2">
```

File Structure

Immediately after the version information comes a representation of the file structure. Here is a simple example of a <folder> containing 4 <file>s:

```
<folder id="0" name="mon_legs">
     <file id="0" name="mon_legs/foot_0.png" width="0" height="0"/>
     <file id="1" name="mon_legs/foot_a.png" width="0" height="0"/>
     <file id="2" name="mon_legs/pelvis.png" width="0" height="0"/>
     <file id="3" name="mon_legs/thigh_0.png" width="0" height="0"/>
</folder>
```



Name	Date modified	Type	Size
mon_arms	8/23/2012 12:03 PM	File folder	
mon_head	8/23/2012 12:03 PM	File folder	
mon_legs	8/23/2012 12:03 PM	File folder	
mon_shadows	8/23/2012 12:03 PM	File folder	
mon_torso			
Example.SCML			

In the currently released alpha of Spriter, the 'width' and 'height' attribute are always 0, and should be ignored. In a near future version they will begin to actually reflect the original dimensions of the image file.

The <folder> and <file> 'id' values will be used by sprites to reference the image files that should be displayed. In future versions these same <folder> <file> structures will be used for other purposes such as texture atlas files and sound effects. The current version only uses them for images.

The values of 'id' can be assumed to progress in numerical order, so the first, second, and third <file> within a <folder> will always have the id's of “0”, “1”, and “2”, respectively.

The full relative filename (subfolder/filename) is included in the 'name' attribute for <file>s to avoid any unnecessary string operations.

Entities

Each SCML file will contain one or more <entity>s. The <entity> tag is a collection of zero or more <animation>s. The current version of Spriter will always export exactly one <entity> per file, and does not yet allow the user to 'name' it. Again, the 'id's are guaranteed to be in numerical order. Examples of <entity>s could be 'name'd “main character”, “turret”, or “ui button”.

```
<entity id="0" name="">
```

Animations

Each <animation> within an entity consists of the attributes: 'id', again listed in numerical order, the 'name' of the animation, and it's total 'length' in whole milliseconds. Currently the editor only displays looping animations, but always saves as 'looping=”false”'. You can refer to the file spec document for additional details about the unimplemented looping features, which should be coming soon.

```
<animation id="0" name="Idle" length="2000" looping="false">
```

The Mainline

Every <animation> contains exactly one <mainline> (main timeline) which contains zero or more <key>s (key frames):

```
<mainline>
     <key id="0">
          <object_ref id="0" timeline="0" key="0" z_index="0"/>
          <object_ref id="1" timeline="1" key="0" z_index="1"/>
          <object_ref id="2" timeline="3" key="0" z_index="2"/>
          <object_ref id="3" timeline="2" key="0" z_index="3"/>
          <object folder="3" file="0" x="-33" y="251" pivot_x="0"
pivot_y="1" z_index="14"/>
     </key>
     <key id="1" time="202">
          <object_ref id="0" timeline="0" key="1" z_index="0"/>
          <object_ref id="1" timeline="1" key="1" z_index="1"/>
          <object_ref id="2" timeline="2" key="1" z_index="2"/>
          <object_ref id="3" timeline="3" key="1" z_index="3"/>
          ...
```

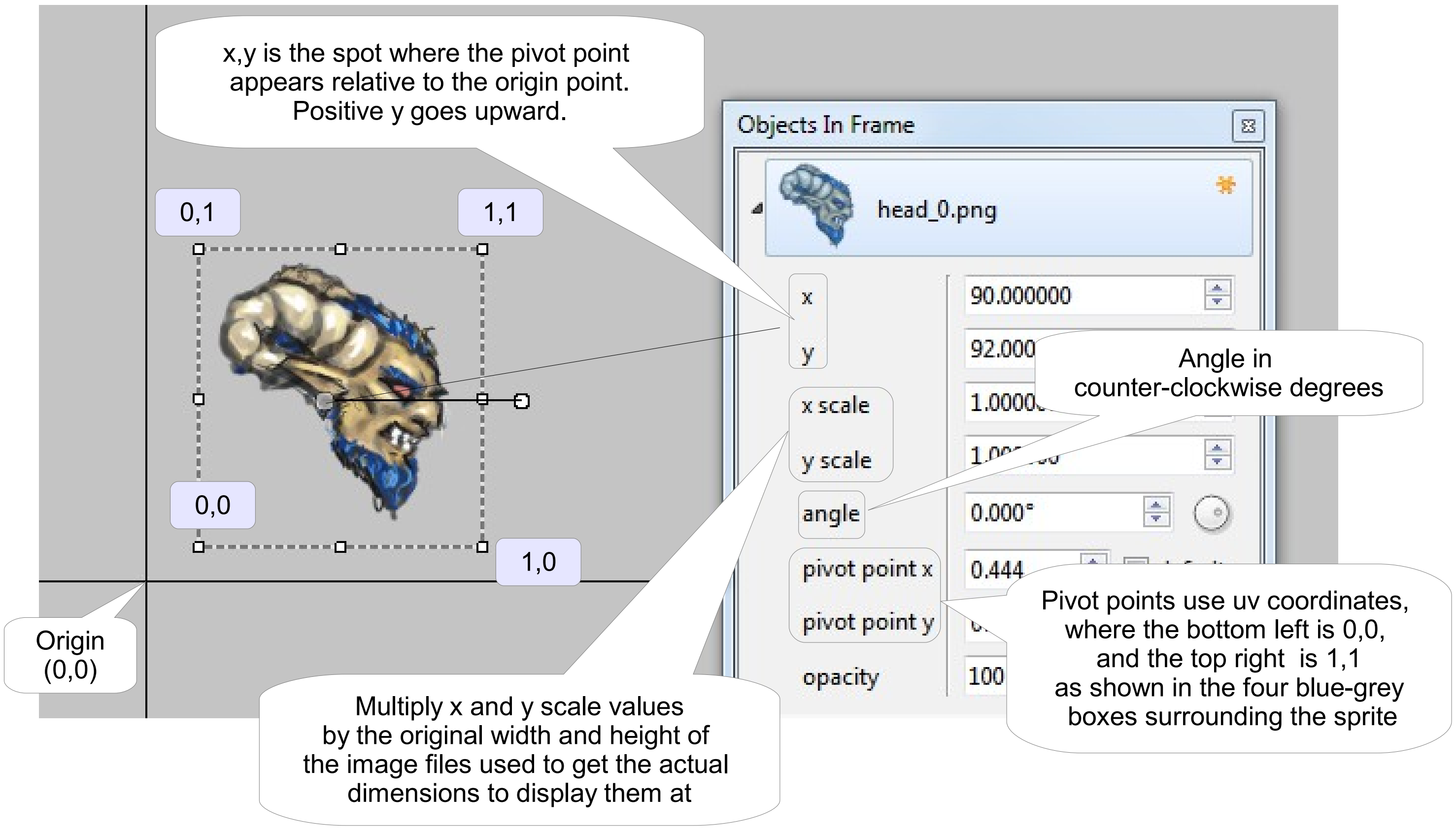
Keys

A key is a keyframe, and as seen above, has 2 attributes: the numerically ordered 'id' value, and a 'time' value. 'Time' is expressed in whole milliseconds, and reflects the start 'time' of the particular <key>, between 0 and the <animation>'s 'length'. A key object contains zero or more and any combination of the following: <object>s, <object_ref>s, and <bone>s. The current version of Spriter does not have bones implemented, but they shall be implemented by version a4. When a 'key' does not contain a 'time' attribute, the default value of 'time' should be assumed to be “0”.

Objects

SCML allows for two types of objects. *Transient objects* and *persistent objects*. *Transient objects* use the <object> tag within the <mainline>, and only exist on one keyframe. To put in another way, their scope is local to the containing <key>. They cannot be tweened, and are not tied to other objects on other keys:

```
<object folder="3" file="0" x="-33" y="251" angle="29.267996" pivot_x="0"
pivot_y="1" z_index="14"/>
```


x,y is the spot where the pivot point appears relative to the origin point. Positive y goes upward.
0,1
1,1
0,0
1,0
Origin (0,0)
Objects In Frame
head_0.png
x
90.000000
y
92.000
x scale
1.0000
y scale
angle
0.000°
pivot point x
0.444
pivot point y
opacity
100
Angle in counter-clockwise degrees
Pivot points use uv coordinates, where the bottom left is 0,0, and the top right is 1,1 as shown in the four blue-grey boxes surrounding the sprite
Multiply x and y scale values by the original width and height of the image files used to get the actual dimensions to display them at

An object tag contains all of the information needed to properly display it within a single tag. The 'folder' & 'file' attributes correspond with the <folder> & <file> id's in the <folder> <file> structure at the start of the file. In a future version if the 'pivot_x' or 'pivot_y' attributes are missing, the <object> should use the default pivot points specified in the <folder> <file> tag structure. One cannot set default pivot points in the current Spriter version, and the default pivot point is always (x:0,y:1), which is the top left of the image. The ability to set a custom default pivot point per file will most likely arrive in one of the next two releases.

The 'x' and 'y' positions are relative to the origin of the actual entity on screen. In the editor, the origin is the central point where the lines in the canvas meet. 'y' increases upward. 'angle' will always be between 0 and 359.999999 degrees increasing counter-clockwise. The 'z_index' indicates the order in which the object should be drawn, but this information can also be inferred from the order in which the <object> or <object_ref> appears in the <key>. A transient object should be drawn it's it's indicated position, and remain on screen in that position until playback reaches the next <mainline> <key>.

Object References

Persistent objects are not local to the key itself, but to the entire animation. These objects can be tweened (in the current version, they *must* be tweened, but this will change in a future version), and these objects are linked to other instances of themselves across keys, regardless of what image is currently being used. This will be important not just for tweening, but will grow to be useful in a larger variety of ways as Spriter matures to include features such as bones and procedural animation. In the <mainline> <key>s, *transient objects* are *referenced* with the <object_ref> tag.

```
<object_ref id="1" timeline="1" key="0" z_index="1"/>
```

Currently, the 'id' tag exactly mirrors the 'z_index' tag, and does not yet serve a useful purpose that can't be served by the 'z_index' value. 'z_index' works the same as for a normal transient <object> tag. The 'timeline' and 'key' attributes refer to the <timeline> and <key> tags that follow the current <mainline>.

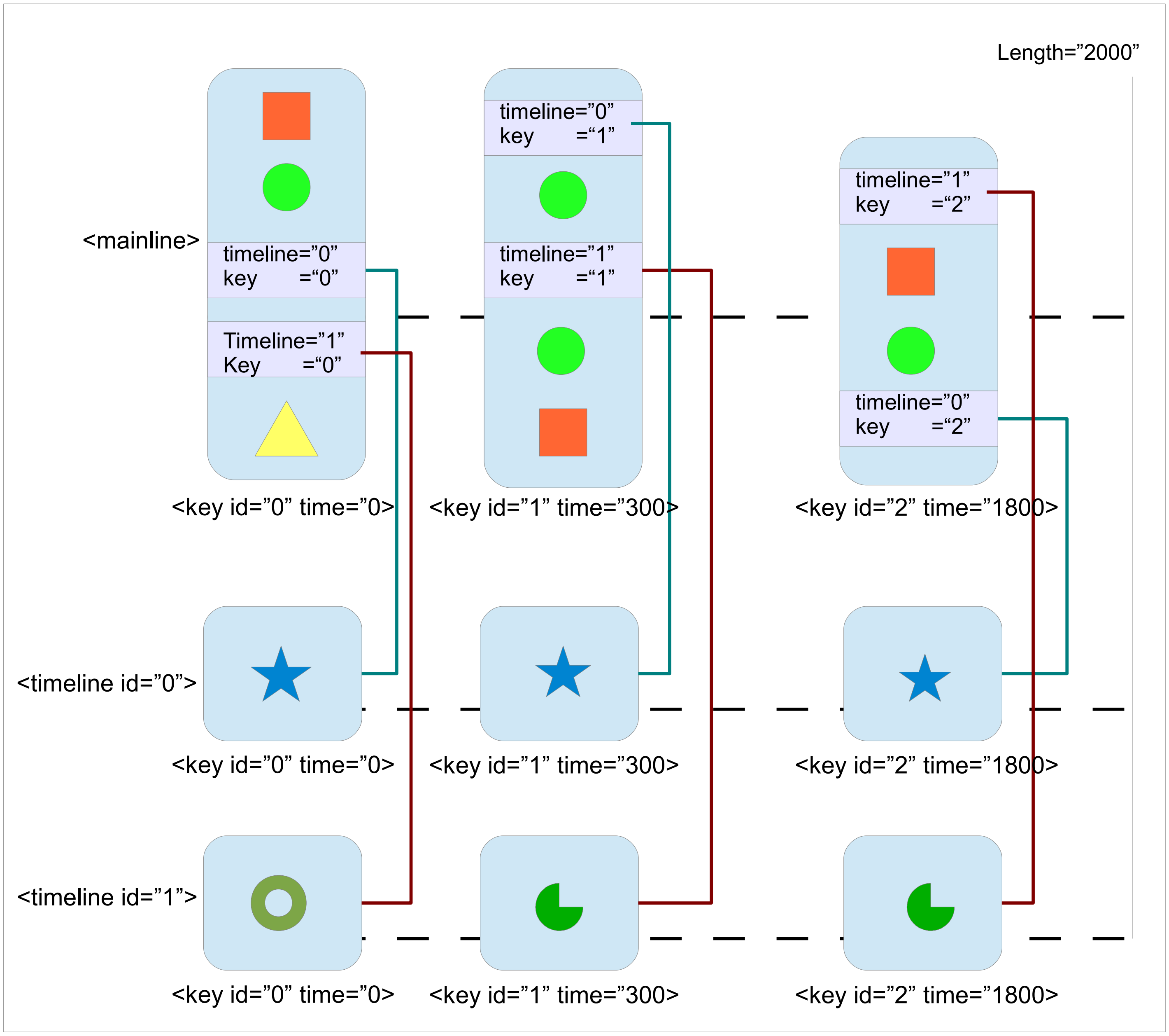
Timelines

Each <timeline> tag contains <key>s referenced by <object_ref>s. Essentially, each <timeline> repesents one persistent object that has scope to the entire animation. These objects should only be drawn when they are referenced by <object_ref> in a <mainline> <key>. For the most part, the <key>s and <object>s in <timeline>s function in mostly the same way as they did in <mainline>. The main difference for now is that they can be tweened. The color coded 'timeline' and 'key' tags in the <object_ref> tag above, reference the corresponding tags below. It should simply draw the object below as it it were an <object> replacing the <object_ref> at that spot in the referring <key>'s z_order.

```
<timeline id="1">
   <key id="0">
      <object folder="1" file="0" x="69" y="279" pivot_x="0.422535" pivot_y="0.713043"
angle="354.644"/>
   </key>

   <key id="1" time="276">
      <object folder="1" file="0" x="75.1949" y="274.711" pivot_x="0.422535"
pivot_y="0.713043" angle="358.632"/>
   </key>

   <key id="2" time="544">
      <object folder="1" file="0" x="82" y="270" pivot_x="0.422535" pivot_y="0.713043"
angle="3.01279"/>
   </key>

   <key id="3" time="776">
      <object folder="1" file="0" x="87.7189" y="263.944" pivot_x="0.422535"
pivot_y="0.713043" angle="5.069"/>
   </key>
</timeline>

<timeline id="2">
   <key id="0">
      <object folder="1" file="1" x="72" y="219" pivot_x="0.181818" pivot_y="0.628205"
angle="348.799"/>
   </key>
   ...
```

Mainlines and Timelines Together

We've covered all of the tag information needed to implement all features available in the current alpha of Spriter. But there are some additional things that should be taken into consideration when implementing timelines and tweening. If the relationship between mainlines and timelines isn't yet clear, it's worth understanding a bit better so you can model your implementation around the concept. Take a look at the example animation below:



Each shape represents an <object> in a <key>. The objects in <mainline> <key>s are transient objects and have nothing to do with each other. Where you see the 'timeline' and 'key' attributes instead of a shape in the <mainline> <key>s, those are <object_ref>s and the corresponding <object> (shape) is retrieved from it's corresponding <timeline> <key> below, and inserted in that spot in the z-order.

In the current build of Spriter, there are no options for curve_types when tweening. It assumes the default “linear” tweening. In the future, the options for curve_type will be “instant”, “linear”, “quadratic”, and “cubic”. When the attribute is missing, the 'curve_type' “linear” should be assumed:

```
<key id="0" time="100" curve_type="linear" spin="1”>
```

If the 'spin' attribute is missing, the default “1” should be assumed. 1 means the object should spin counter-clockwise when tweened, and -1 means it should spin clockwise.

If you are unfamiliar with how to perform linear interpolation, and your platform doesn't have any built in functions, we can reduce all the math needed into one formula. Assume you have two keys: Key A, and Key B. Key A has a time attribute of timeA, and Key B's is timeB. Your currentTime in the playback of the animation is somewhere between timeA and timeB. You want find a value like 'x' or 'yScale' for this particular moment that is between Key A's version of the value, represented by 'a' in the formula. Key B's version of that value is represented by 'b'.

```
tweenedValue=a+((b – a)*((currentTime-timeA)/(timeB-timeA)))
```

Here is a specific example. Say we have the following two keys for an object, and we want to tween the 'x' attribute of the object. I've removed the rest of the attributes to keep it simple:

```
<key id="3" time="570">
     <object x="80"/>
</key>
<key id="4" time="742">
     <object x="89"/>
</key>
```

For this example, let's assume the current time we are at in playback is **604.**

```
x=80+((89 – 80)*((604-570)/(742-570)))
```

The one special case for this is 'angle', since angles wrap around at the 360 – 0 point, and the value for 'spin' determines the spin of the angle. To ensure you're always spinning in the correct direction, there is a simple rule to apply:

Assuming angleA is the angle from Key A, and angleB is from Key B:

If `spin=”1”` (counter-clockwise) and `angleB – angleA` is negative, then add 360 to angleB before using it in the above formula.

If `spin=”-1”`, and `angleB – angleA` is positive, then subtract 360 from angleB before using it in the above formula.

To make sure that looping functions correctly, if looping is enabled, and you are on the last key with a tweened object, use the <animation> 'length' to determine timeB. If an animation is not looped, the values at the last key should be held and untweened until the animation is complete.

And lastly, this aspect is only partially implemented in the current version of Spriter, but persistent objects do not need to exist on every mainline key. Only draw the object if it appears as an <object> or <object_ref> in the <mainline>. Also, make sure that when you are getting timeA and timeB of your two keys when tweening you are getting it from the 'time' attribute of the referenced <key> in <timeline>, and not the originating <key> in <mainline>. Spriter currently requires an object to be keyed on each <mainline> keyframe it appears on. In the future this will be expanded to give artists more options, and the <key> referenced won't always have the same 'time' value as the key that references it. As long as you always use the 'time' from the <timeline> <key> when tweening, it should work correctly.

That's the Basics of it

Much of the remaining aspects of SCML that aren't yet implemented in the editor are self-explanatory, and you can view the full format spec here.

Please post any questions or suggestions to brashmonkey.com/forum, or contact me directly at lucid@brashmonkey.com.