## *Usage of the sprite system*

Some simple code examples. There are more commands and uses than those listed below, this is more of a quick start.

NOTES:
Bear in mind that I haven't worked on optimizing the engine – "first make it run, then make it run fast". When the feature set is complete and I'm sure I won't be rewriting any functionality, then it's ready for optimization.

After a script error, you want to clearGlobals before restarting, since the system can have zombies (half-killed objects) since the last run. (I've added a global handler called ccc that just does that, makes it more convenient)

There may be memory leaks in the system. Also, you will create leaks if you don't clean up properly – kill() the objects you create.

### Preparing a movie

Link SpriteSys.cst to the movie.
Drop CBHSpriteSys as a frame script in frame 1.
Done.

### Creating and manipulating a simple sprite

```
oSprite = script("CSprite").new()
oSprite.setLoc(point(100,100))
oSprite.setMember("AMember")
```

or do the same thing with

```
oSprite = script("CSprite").new([#Member:"AMember", #Loc:point(100,100)])
```

Most properties that you can access can be included in the new() property list.

### Accessing properties

You cannot use direct property access, like oSprite.loc. Access functions are used instead, like setLoc(value)/getLoc(). You can also use setOneProp(#Loc, value) and getOneProp(#Loc).

Use oSprite.getPublicProps() to get a list of available props to get/set, and info about them.

All props aren't implemented fully yet, I've only added what I've needed myself so far. Note that instead of locH/locV I use locX/locY.

Note: In the C# version, a direct access style will be allowed, thanks to C#s "properties". To the user, it looks like direct access, but on the inside it really is a function call. This is much how the Lingo -> Xtra access works (in the C code, a getProp() handler gets called when you do sprite.loc)

### Creating a child sprite

When creating the sprite that will be a child of another sprite, use the #Parent property:

```
oSprite1 = script("CSprite").new([#Member:"Member1"])
oSprite2 = script("CSprite").new([#Parent: oSprite1, #Member:"Member2"])
```

When you move, scale or rotate oSprite1, oSprite2 will follow:

```
oSprite1.move(point(20,20))
```

oSprite1.rotate(45)

## Dummy sprites

A sprite doesn't have to have an image or member for rendering. It can be a holder for a behavior, or it might have other ways to render. It can also be just a parent node for managing a group of sprites. In this case, the source size must be set to at least 1,1 – otherwise there is some math that will fail. I'll fix this later, but for now, you'll need to:
oSprite.setSourceRect(rect(0,0,1,1))

## Behaviors

A behavior script needs some things to work properly.
A handler named **kill** (should contain code to clean up all properties used by the behavior).

To receive mouse event, a handler called **mouseEvent**, with the arguments:

```
on mouseEvent me, a_oSender, a_sbEvent
  --a_oSender is the sprite object
  --a_sbEvent is an event like #Enter,#Leave,#Down,#Click,#StillDown,#Up,#UpOutside
end
```

To receive frame updates, a handler called **stepFrame**.

To add a behavior to a sprite, use addBehavior(object), like this:
oSprite.addBehavior(script("CMyBehavior").new())

as a second argument, you can add an identifier for later access:
oSprite.addBehavior(script("CMyBehavior").new(), #FirstOne)

Remove behaviors by calling removeBehavior(). As an argument, pass the instance of the object, or the identifier:
oSprite.removeBehavior(#FirstOne)

The behavior's kill handler will be called by removeBehavior(), unless you send an extra argument, TRUE.

## Animations

Say we want to animate a sprite's locX from 10 to 100 back to 10. First, create an animation chart:
aChart = [10,100,10]

Then create the animation object, including information about which object to animate, and which property of that object:
oAnim = script("CAnim").new([#Chart: aChart, #object:oSprite, #property:#LocX])

Finally, add it as a behavior to the sprite:
oSprite.addBehavior(oAnim)
Subject to change: maybe the animation should add itself automatically.

The chart can be defined as a property list as well, like [1:10, 6:100, 8:10]. This example means keyframes in the animation appears only on frame 1, 6 and 8. Between the keyframes, the animation is interpolated with a spline.

It's also possible to use points, [point(0,0), point(30,10)] in the lists, and set the #Property to #Loc, however this will not work properly with the editors that are on their

way. (Of course, if you're coding the list by hand, editing it in an editor isn't very relevant anyway)

You can change playback speed of the animation object:
oAnim.setSpeed(0.5) –play back at half speed TODO: negative values (backward playing) isn't implemented.

## *Director Movie Ripping and Playback*

TODO: currently, the ripping and playback ignore behaviors, a functionality that could be added – provided the behaviors are written in a manner that works with both Director's engine and this one (two code paths will be needed in the behaviors).
Also, it would be easy to implement a script for recording what's being displayed on stage while a movie is playing.

### Ripping

Open a movie with score animations (which is not dependent on scripting to work).

Make sure all members used in the score are located in an external cast.

Link SpriteSys.cst to the movie.

Create the field member to hold the score/cast data, named e.g. "MovieData" (preferrably in one of the casts containing other members used in the movie).

In the message window, write
member("MovieData").text = string(script("CMovieConverterUtils").readScore())

The script steps through the movie and creates a list with score data and info about the members used.

### Playback

In the context where you want to play the score, write:

```
oParentSpriteForPlayback = script("CSprite").new()
oClipPlayer = script("CClipPlayer").new([#Sprite: oParentSpriteForPlayback,
#Movie:"MovieData"]) -- NOTE:add #Speed:<floating-point value> for changing speed
oParentSpriteForPlayback.addBehavior(oClipPlayer)
```

Subject to change: CClipPlayer is very much like CAnim, so they should merge – or CClipPlayer should inherit from CAnim.