animationEngine 6.0 Users manual. ©2005 - 2011 derbrill IT-service
(Revision 6- 003)

Before you start, make sure you do understand the following liveCode concepts:
What is a stack, what are cards, what are controls? What are messages and handlers? What is a function? What are properties? What are custom properties? What is a script library?
If you can answer the above questions, you are good to start working with animationEngine. It is a lot of fun.
To get started quickly:

Open stack animationEngine.livecode, the stack you have downloaded from http://www.runrev.com
If you just want to play around a little, you can hit the checkbox labeled "Use me!" and you are ready to go.
Next, open the liveCode messageBox and type:

aeMoveTo the long name of stack "animationEngine", the screenLoc,1000,"bounce"

and hit return. Congratulations! You moved your first object with animationEngine. If you want to start using animationEngine by script, you could do something like this:

```
-- in the script of the first card of your mainStack
on openStack
  if "animationEngine" is not among the lines of the stacksInUse then
    start using stack "animationEngine"
   end if
end openStack
```

We hope you enjoy working with animationEngine! If you have any questions regarding how to put it into best use, please visit:

http://forums.runrev.com/phpBB2/viewforum.php?f=27

What is new in version 6?

animationEngine 6 has been optimized to work with liveCode version 6.7 and higher, including, but without warranty the current developer preview of the liveCode 8 engine (which is by the time of this writing version 8 DP 6). One of the most notably changes is one in licensing. animationEngine is now dual licensed!

Licensing terms:

These scripts are licensed to you if you agree to be bound to one of the following License types at your choice.

1) GPL 3
You may use animationEngine as FREE Software as outlined in the terms of the GPL3 or any higher version of the GPL as found here: http://www.gnu.org/licenses/gpl-3.0.html

2) Commercial license:
If you do not want to disclose the sources of your application you have the option to purchase a commercial license by paying a fee. You can buy a commercial license from the runrev marketplace. At the time of the writing of this document this can be done following this link: http://livecode.com/products/extensions

You are paying a license fee for the major version of animationEngine. If you are licensing animationEngine 6.0, you will be able to use all upgrades that carry the same major version number (in this case 6.x). Once the switch is made to a version 7.x of the library, you will need to upgrade your license to use the latest version. However, of course you may continue to use any version of the library you currently have licensed, without needing to purchase an upgrade.

If you have previously licensed an earlier version of animationEngine, a discounted upgrade option may be available.

Such a commercial license releases you from the requirements of the copyleft GPL license, which include: distribution of all source code, including your own product; licensing of your own product under the GPL license; prominent mention of the derbrill copyright and the GPL license; and disclosure of modifications to the library.

Code Contributions
If you want to contribute to animationEngines codebase and want your changes to be accepted into the main trunc, you will have to accept our open source contribution agreement as found here: http://www.derbrill.de/osca.pdf

animationEngine 6 adds a couple of major features to its toolset. Please make sure that you download and try the demo stack for this version.

Availability of source code: you will find the latest sources and documentation on Github.

https://github.com/derbrill/animationEngine

Finally, thank you for helping out! If you purchased a license for animationEngine, I really appreciate that! It helps moving the library forward! If you are using animationEngine in a GPL context and still want to help, I would really appreciate a donation via PAYPAL to info@derbrill.de

What is new in version 6? - continued

animationEngine 6 adds 4 major features, along with a couple of helper functions to its Toolset:

1) **aeChangeAngleCircular** changes the position of an object on a cirCular path. It allows you to use easing effects for the transition.

Example:

aeChangeAngleCircular the long ID of grc "ball",the loc of grc "path",80,tNewAngle,2000,"counterclockwise","bounce"
aeChangeAngleCircular the long ID of grc "ball",the loc of grc "path",80,270,2000,"clockwise","inout"

Parameters:

long ID of the object to move
centerX of path
centerY of path (centerX and centerY can be combined expression)
radius from center
desired new angle
duration in millisecs
direction (clockwise OR counterClockwise)
easing method (in, out, inOut, overshoot, bounce or empty)

See stack ae6demo to see it in action.

2) **aeMorphGraphic** transitions a graphic between 2 pointlists. This is also known as "tweening".

Example:

aeMorphGraphic the long ID of grc 1,sPoints1,sPoints2,1500, tEffect

Parameters:

long ID of a graphic object
pointList (start)
pointList (end)
duration in millisecs
easing method (in, out, inOut, overshoot, bounce or empty)

See stack ae6demo to see it in action.

Attention!

Currently the pointLists must hold an equal number of entries, otherwise the morphing of the graphics will break. If there is enough interested in this feature, I will try to explore means to change that behaviour.

What is new in version 6? - continued

3) **aeMorphGradientRamp** changes the ramp of a fillgradient of an object. It allows you to use easing effects for the transition.

Example:

aeMorphGradientRamp the long ID of grc 1,tGradient1,tGradient2,3000,"overshoot"

Parameters:

long ID of the graphic that changes the gradient
startGradient
endGradient
duration in millisecs
easing method (in, out, inOut, overshoot, bounce or empty)

See stack ae6demo to see it in action.

Attention!

Currently the gradientLists must hold an equal number of entries, otherwise the morphing of the graphics will break. If there is enough interested in this feature, I will try to explore means to change that behaviour.

2) **aeRotateGroup** rotates all members of the target group around an arbitrary point.

Example:

aeRotateGroup the long ID of grp "rotateMe", the loc of grc "Center",90,1000,"overshoot"

Parameters:

long ID of a group
centerX of rotationPoint
centerY of rotationPoint (centerX and centerY can be combined expression)
desired angle in degrees
duration in millisecs
easing method (in, out, inOut, overshoot, bounce or empty)

See stack ae6demo to see it in action.

Setting up collision listeners

Accurate collision detection used to be a CPU consuming task in liveCode. With the introduction of the improved intersect command in LC 5.0 this changed a lot. You can now easiely check for collisions of objects that use an alpha channel (images, graphics and even buttons with their icon set) AnimationEngine now uses the improved intersect tests and adds a way to listen for collisions with just a few lines of script. To be frank, this is the feature we are most fond of in AE5!

Here is an example of how to do that:
Create a stack. Create a text field and set its name to "output"
Create 6 graphic objects (we are using graphics only for the simplicity, you could also use a couple of png images, or buttons with their icon set. If you are using buttons, you need to set the following properties:
opaque false ; showBorder false ; showName false ; shadow false ; hiliteBorder false ; armBorder false ; showFocusBorder false ; hiliteIcon 0 ; hoverIcon 0)
Name your graphic objects. Set the names of them to gr1,gr2,gr3,gr4,gr5,gr6
Make sure you set the opaque of the graphics to true.

Create a button and set its script to:

```
on mouseUp
  if "animationEngine" is not among the lines of the stacksInUse then
    answer "This stack needs animationEngine 5 or higher to run."
    exit mouseUp
  end if
  aeStopListeningForCollisions
  -- this will clear the list of all objects listening for collisions
  local tList
  set the flag of me to not the flag of me
  if the flag of me then
    -- set up graphics to be draggable
    repeat with i=1 to the number of graphics
      set the constrainRectangular of graphic i to the rect of this cd
      -- graphics need to be opaque to be draggable
      set the opaque of graphic i to true
    end repeat
    repeat with i = 2 to the number of graphics
      put the long id of graphic i & cr after tList
    end repeat
    -- remove trailing carriage return
    delete char -1 of tList
    set the aeListenForCollisionsWith of graphic 1 to tList
    -- right now we assume all but graphic 1 are predators
    -- you could set up lists for each graphic though
    aeStartListeningForCollisions
    set the label of me to "Stop listening for collisions"
  else
    aeStopListeningForCollisions
    set the label of me to "Start listening for collisions"
  end if
end mouseUp
```

The script sets up a list of graphic references (the long ID is a reference to the graphic, that identifies it uniquely). right after that a property in animationengine is set that tells the graphic with the lowest layer (the first graphic you created on the card) to listen for collisions with the other objects. As you can see there is quite a bit of overhead in the script that just checks if anmationEngine is present and to give a means to start and stop listening for collisions. The actual listening part is pretty straightforward.

Setting up collision listeners (continued)

Next open a script editor to edit the card script:

```
on aeCollision pObjects
   put the short name of the target && "collides with" & cr & pObjects into fld "output"
end aeCollision

on constrainRectangularCallBack
  local tObjects
  put aeCollidingObjects() into tObjects
  if the keys of tObjects is empty then
    put empty into fld "outPut"
  end if
end constrainRectangularCallBack
```

The first handler of the script is a callback message that is sent to the control listening for collisions. You could also put it directly into that control. AnimationEngine sends a message to the listening control (referred here as the target). The message is sent with one parameter. A list of all objects that are currently colliding with your listening control. Of course only that controls you told the target to listen to previously. The second handler you see is a callback message animationEngine sents to controls being dragged, if you are using animationEngines constrain handlers. In this demonstration constrainRectangularCallback is used to clear the output field, if no collisions occur. This is done using a function call to animationEngine. aeCollidingObject returns an array, that holds the colliding partners of all listening objects. So if you do not want to rely on messages being pushed to your controls you could instead use that function to set up a list of all colliding objects in a central place. If you decide to rely on the callback message being pushed to the listening control and use that to create your collision response or instead poll the function might be a matter of taste, however, the overhead of calling the function is a little bigger as reacting to the callback message. how often the tests for collisions are executed depnds on the frameRate you set for animationEngine. By default this will be 25 fps. If you want to change that framerate, please look up **aeSetFrameRate** in the handlers section of this document.

Moving objects:

If you are new to animationEngine, it might be a little tough to find the info on which means you have to animate Objects. In this small step by step tutorial, we will show you how to start moving objects around. The easiest way to move an object is animationEngines command **aeMoveTo**.

Make sure animationEngine is loaded and in use. You have animationEngine commands available if you either issue this line of script: *start using stack "animationEngine"* or check the checkbox labeled "use me!" in animationengine.

Create a new stack with a button and a graphic. Set the name of the graphic to "movingGrc"

Open the script of the button and apply the following script:

```
on mouseUp
  aeMoveTo the long ID of grc "movingGrc",the loc of this card,1000,"inOut"
end mouseUp
```

Switch to the run tool and click the button. See your graphic moving from its current location to the middle of the card. The transition will take 1 second. As of version 5 of the liveCode engine, it is recommended that you set the layerMode property of the moving graphic to "dynamic". Depending on the device you are targeting with your application, you should also read up the following properties in the liveCode dictionary:

- the compositorType
- the compositorCacheLimit
- the compositorTileSize

Switching back to animationEngine now. If you want to move more than one object at the same time and want to assure they all start at the same point in time with their movement, you can make use of the command **aeLockMoves**. Assuming you have 5 graphic objects on your card, that you want to move simultanously try the following:

```
on mouseUp
  aeLockMoves
  aeMoveTo the long ID of grc 1,random(the width of this cd),random(the height of this cd),1000,"inOut"
  aeMoveTo the long ID of grc 2,random(the width of this cd),random(the height of this cd),1000,"bounce"
  aeMoveTo the long ID of grc 3,random(the width of this cd),random(the height of this cd),1000,"overshoot"
  aeMoveTo the long ID of grc 4,random(the width of this cd),random(the height of this cd),1000,"out"
  aeMoveTo the long ID of grc 5,random(the width of this cd),random(the height of this cd),1000
  aeUnlockMoves
end mouseUp
```

aeLockMoves will put all aeMoveTo commands that follow it on a heap and execute them simultanously as soon as you issue **aeUnlockMoves**.

If you are moving controls with aeMoveTo and decide the motion needs to stop, you can make use of the command **aeStopMoving**.

```
on mouseUp
  aeStopMoving the long ID of grc 1
end mouseUp

on mouseUp
  aeStopMoving "all"
end mouseUp
```

Functions in animationEngine:

ae3dConvertToScreen(x,y,z,originX,originY[,focalLength])
get ae3dConvertToScreen(50,50,20,100,100)
get ae3dConvertToScreen(50,50,20,the loc of this card,450)

This function converts a point in 3D space to screencoordinates. You specify the x,y and z coordinate as well as the origin of the 3d object on the card. Optional you might specify the focal length of the camera as a fifth parameter. If you do not specify a focal length a default value of 300 is used. 300 is a good value for most situations. If the value is smaller the object gets more depth, but might be distorted.

aeCollidingObjects()
put aeCollidingObjects() into tObjects

This function returns an array of all colliding objects defined by all collision listeners. The keys are references to the listening control, the contents is a list of references to all controls colliding with that control

aeEaseIn(startValue,endValue,duration,elapsedTime,exponent)
get aeEaseIn(10,20,1000,50,2)
get aeEaseIn(10,20,1000,500,4)

Use this function to generate easeIn values. Call it in a timed script to get all values inbetween. You specify a start and an end value, the ease duration, the elapsed time and an exponent (for quadric, cubic or whatever easing you wish)

aeEaseInOut(startValue,endValue,duration,elapsedTime,exponent)
get aeEaseInOut(10,20,1000,50,2)
get aeEaseInOut(10,20,1000,500,4)

Use this function to generate easeIn and out values. Call it in a timed script to get all values inbetween. You specify a start and an end value, the ease duration, the elapsed time and an exponent (for quadric, cubic or whatever easing you wish)

aeEaseOut(startValue,endValue,duration,elapsedTime,exponent)
get aeEaseOut(10,20,1000,50,2)
get aeEaseOut(10,20,1000,500,4)

Use this function to generate easeOut values. Call it in a timed script to get all values inbetween. You specify a start and an end value, the ease duration, the elapsed time and an exponent (for quadric, cubic or whatever easing you wish)

aeMathEaseOut - Reserved for internal use.

aeOvershootEaseIn(startValue,endValue,duration,elapsedTime)
get aeOvershootEaseIn(10,20,1000,50,2)
get aeOvershootEaseIn(10,20,1000,500,4)

Use this function to generate easeIn values. Call it in a timed script to get all values inbetween. You specify a start and an end value, the ease duration and the elapsed time. Values will overshoot the Endvalue and finally reach the Endvalue if the elapsed time reaches the duration.

aeWithinEllipse(ellipseX,ellipseY,ellipseX radius,ellipseY radius,pointX,pointY)

Lets you test if a testpoint pointX,pointY lies within the ellipse specified by the first parameters.

Functions in animationEngine:

aspectRatio(width,height)
get aspectRatio(the width of grc 1,the height of grc 1)

This function returns the width-to-height ratio of the specified width and height.

aspectResize - Reserved for internal use.

circleCollide(x1,y1,x2,y2,threshold)
get circleCollide(100,150,200,140,80)
get circleCollide(the loc of grc "test",the loc of img "myImage",80)

This function returns true if there is a circular collision of the 2 objects. Else it returns false. Threshold is the sum of both circles radii.

circleLineSegmentCollide(x1,y1,x2,y2,x3,y3,radius)
get circleLineSegmentCollide(100,200,200,250,80,80,40)
get circleLineSegmentCollide(line 1 of the points of grc "line",line 2 of the points of grc "line",the loc of img "circle",40)

This function returns true if there is a collision between a line segment (specified by startpoint x1,y1 and the endpoint x2,y2 and a circle (specified by the circles centerpoint x3,y3 and it´s radius). Else it returns false.

closestPointOnLine(x1,y1,x2,y2,x3,y3)
get closestPointOnLine(10,10,200,40,20,20)
get closestPointOnLine(line 1 of the points of grc "line",line 2 of the points of grc "line",the loc of button "myButton")

This function returns the closest point to x3,y3 that lays on the line specified by the points x1,y1 and x2,y2. As a line is endless, this point doesn't necessarily lie on the linesegment specified by x1,y1/x2,y2, but might lie on the extension of that line.

distance(x1,y1,x2,y2)
get distance(100,100,200,200)
get distance(the loc of grc "myGraphic",the loc of img "myImage")

This function returns the distance between to given points in pixel.

Functions in animationEngine:

findAngle(x1,y1,x2,y2)
get findAngle(100,200,50,300)
get findAngle(the loc of grc "myGraphic,the loc of btn "myButton")

This function returns the center angle of a circle in degrees. x1,y1 are the coordinates of the center point of the circle. x2,y2 are the coordinates of a point on the circular arc. Possible values are 0-360, where 0 degrees means x2,y2 being on top of x1,y1.

findAngle2(x1,y1,x2,y2,startangle)
get findAngle2(100,200,50,300)
get findAngle2(the loc of grc "myGraphic",the loc of btn "myButton")

returns the center angle of a circle in degrees. x1,y1 are the coordinates of the center point of the circle. x2,y2 are the coordinates of a point on the circular arc. Possible values are 0-360, where 0 degrees means x2,y2 being at the startangle.

findAngleX(x1,y1,x2,y2)
get findAngleX(100,200,50,300)
get findAngleX(the loc of grc 1,the loc of grc 2)

returns the center angle of a circle in degrees. x1,y1 are the coordinates of the center point of the circle. x2,y2 are the coordinates of a point on the circular arc. Possible values are 0-360, where 0 degrees means x2,y2 being right to x1,y1.

findAngleY(x1,y1,x2,y2)
get findAngleY(100,200,50,300)
get findAngleY(the loc of grc 1,the loc of grc 2)

returns the center angle of a circle in degrees. x1,y1 are the coordinates of the center point of the circle. x2,y2 are the coordinates of a point on the circular arc. Possible values are 0-360, where 0 degrees means x2,y2 being on top of x1,y1.

This function can be used as a synony for findAngle()

findPreciseAngle(x1,y1,x2,y2)
get findPreciseAngle(100,200,50,300)
get findPreciseAngle(the loc of grc "myGraphic,the loc of btn "myButton")

This function returns the center angle of a circle in degrees. x1,y1 are the coordinates of the center point of the circle. x2,y2 are the coordinates of a point on the circular arc. Possible values are 0-360, where 0 degrees means x2,y2 being on top of x1,y1. In difference to the findAngle function values are not rounded.

findPreciseAngleX(x1,y1,x2,y2)
get findAngleX(100,200,50,300)
get findAngleX(the loc of grc 1,the loc of grc 2)

returns the center angle of a circle in degrees. x1,y1 are the coordinates of the center point of the circle. x2,y2 are the coordinates of a point on the circular arc. Possible values are 0-360, where 0 degrees means x2,y2 being right to x1,y1. In difference to findAngleX() the returned value is not rounded.

Functions in animationEngine:

getSlope(x1,y1,x2,y2)
get getSlope(10,30,40,50)
get getSlope(line 1 of the points of grc 1,line 2 of the points of grc 1)

This function returns the slope of a line defined by its start end endpoints.

imageCollide(the name of image 1,the name of image 2[,threshold]) *DEPRECATED*
get imageCollide(the name of img "myImage1",the name of img "myImage2")

This function returns if the opaque pixels of 2 images collide. You can specify a threshold value between 0 and 255 to avoid collisions with shadows or dirt in any images mask. If the threshold value is rather high it might be that imageCollide() doesn´t detect collisions as precise as you might need. imageCollide() has been updated in 5.0 and makes use of the new intersect function in the liveCode engine. **The function is now deprecated**, as you may as well use the built in intersec function directly. To ensure existing projects do not require too many code changes, the function call remains in animationEngine.

intersectRect(left1,top1,right1,bottom1,left2,top2,right2,bottom2)
get intersectrect(10,80,90,150,0,40,70,120)
get intersectrect(the rect of btn "myButton",the rect of grc "myGraphic")

This function returns the smallest rectangle of intersection of any 2 rectangles.

isoToScreen(x,y,z,originX,originY)
get isoToScreen(100,200,20,200,200)

This function returns the screen coordinates of a 3D point in isometric view. Returned values are rounded in order to set the points of a graphic control to isoToScreen points.

isoToScreenX(x,y,z,originX)
get isoToScreenX(100,200,20,200)

This function returns the x component of a 3D point converted to screen coordinates in isometric view. It is needed by the isoToScreen function. Returned values are not rounded.

isoToScreenY(x,y,z,originY)
get isoToScreenY(100,200,20,200)

This function returns the y component of a 3D point converted to screen coordinates in isometric view. It is needed by the isoToScreen function. Returned values are not rounded.

Functions in animationEngine:

lineIntersectionPoint(x1,y1,x2,y2,x3,y3,x4,y4)
get lineIntersectionPoint(100,100,100,120,0,0,0,50)

This function returns the intersectionpoint of 2 lines or "parallel" if the lines are parallel.

lineSegmentIntersectionPoint(x1,y1,x2,y2,x3,y3,x4,y4)
get lineSegmentIntersectionPoint(100,200,120,210,0,0,140,220)

This function returns the intersectionpoint of 2 line segments or "no intersectionPoint" if the line segments do not intersect.

midBottom(object reference)
get midBottom(the name of img "myImage1")
get midBottom(the long ID of grc "myGraphic")

returns the middlepoint of the bottom edge of any controls bounding rectangle.

midLeft(object reference)
get midLeft(the name of img "myImage1")
get midLeft(the long ID of grc "myGraphic")

This function returns the middlepoint of the left edge of any controls bounding rectangle.

midRight(object reference)
get midRight(the name of img "myImage1")
get midRight(the long ID of grc "myGraphic")

This function returns the middlepoint of the right edge of any controls bounding rectangle.

midTop(object reference)
get midTop(the name of img "myImage1")
get midTop(the long ID of grc "myGraphic")

This function returns the middlepoint of the top edge of any controls bounding rectangle.

pointInPoly(x,y,pointlist)
get pointInPoly(100,200,the points of grc "myGraphic")
get pointInPoly(the loc of btn "myButton",the points of grc "myGraphic")

This function returns if a point x,y is left,right or on the line specified by point x1,y1 and the point x2,y2.

Functions in animationEngine:

pointOnCircle(x,y,angle in degrees,radius)
get pointOnCircle(the loc of grc 1,50,120)
set the loc of grc 1 to pointOnCircle(100,100,20,90)

This function returns the coordinates of a point on the circular arc.

pointOnEllipse(x,y,angle in degrees,x radius,y radius)
get pointOnEllipse(the loc of grc 1,50,120,90)
set the loc of grc 1 to pointOnEllipse(100,100,20,90,120)

This function returns the coordinates of a point on the elliptical arc.

pointOnLine(x1,y1,x2,y2,distance in pixel)
get pointOnLine(100,100,200,200,10)
set the loc of grc 1 to pointOnLine(50,100,30,600,8)

This function returns the point on a given line [x1,y1;x2,y2] in a specified distance from the first point.

polyCollide(pointlist1,pointlist2[,method])
get polyCollide(the points of grc 1,the points of grc 2,"SAT")
get polyCollide(the points of grc 1,the points of grc 2,"PIP")
get polyCollide(the points of grc 1,the points of grc 2)

This function returns if 2 given polygons collide. Method is one of SAT (Separating Axes Theorem),PIP (Point in Polygon test) or LIT (Line intersection test)

If you don´t specify a method LIT is used.
SAT is the fastest method, but only works reliable for convex polygons.
PIP is the best choice if you are sure that the polygons are not overlapping before the first test.
LIT is the most precise, but slowest method.

polyCollide is an early out function. This assures a fast computation of the polygon collision test.

rotate3DPoint(x,y,z,XRotation,Yrotation,Zrotation[,pFocalLength])
get rotate3DPoint(100,200,20,0,45,0,400)
get rotate3DPoint(100,200,20,0,0,90)

Use this function to rotate a 3D point around the x- y- and/or z-axis. You can rotate over all 3 axis with a single call of the function. You get a non rounded point triple in return.

Functions in animationEngine:

rotateIsoPoint(x,y,z,XRotation,Yrotation,Zrotation)
get rotateIsoPoint(100,200,20,0,45,0,400)
get rotateIsoPoint(100,200,20,0,0,90)

Use this function to rotate an isometric 3D point around the x- y- and/or z-axis. You can rotate over all 3 axis with a single call of the function. You get a non rounded point triple in return.

whereIsThePoint(x1,y1,x2,y2,x3,y3)
get whereIsThePoint(100,200,50,300,80,100)
get whereIsThePoint(line 1 of the points of grc 1,line 2 of the points of grc 1,the loc of btn "myButton")

This function returns:
"left" when the point (x3,y3) is left of the line (x1,y1,x2,y2)
"right" when the point (x3,y3) is right of the line (x1,y1,x2,y2)
"on" when the point (x3,y3) is on the line (x1,y1,x2,y2)

Attention! The sorting of the points is important. The first 2 parameters represent the point you look at, the second 2 parameters the point where you stand. If you reorder the paramters you pass for the same constellation of points you get the inverse result.

Handlers in animationEngine:

aeChangeBackColor pControl,pColor,pDuration[,pMethod]
aeChangeBackColor the long ID of fld "myField","white",500,"inout"
aeChangeBackColor the long ID of grc "myGraphic",127,255,127,1500,"overshoot"
aeChangeBackColor the long ID of grc "myGraphic","#bbcc12",120

This command lets you transition from the controls current backgroundColor to a new backgroundColor in a given amount of time applying an easing effect if you wish to. To stop the color transition before it has finished use **aeStopChangingBackColor.**
aeStopChangingBackColor the long ID of grc 1
aeStopChangingBackColor "all"

You will need to give aeChangeBackColor the following parameters.

pControl: a long reference to the control, card or stack that should change its backgroundColor
pColor: a color reference in any valid revTalk format
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the control that changes the color. Once the transition is done aeChangeBackColorDone is sent to the control.

aeChangeBackColorDone

Callbackmessage sent to a control after it changed its background color scroll using aeChangeBackColor.

aeChangeBackColorOfChunk pField,pChunk,pColor,pDuration[,pMethod]
aeChangeForeColorOfChunk the long id of fld 1,"char 2 to 4 of word 2","blue",1000
aeChangeForeColorOfChunk the long id of fld "myField","line 3","red",1000,"overshoot"

This command lets you transition from the fields current chunk background color to a new color in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeBackColorOfChunk the following parameters:

pField: a long reference to the field that contains the chunk to be changed
pChunk: the chunk being changed
pColor: any color reference
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the field. Once the transition is done aeChangeBackColorOfChunkDone is sent to the field.

aeChangeBackColorOfChunkDone

Callbackmessage sent to a control after it changed its background color scroll using aeChangeBackColor.

Handlers in animationEngine:

aeChangeForeColor pControl,pColor,pDuration[,pMethod]
aeChangeForeColor the long ID of fld "myField","white",500,"inout"
aeChangeForeColor the long ID of grc "myGraphic",127,255,127,1500,"overshoot"
aeChangeForeColor the long ID of grc "myGraphic","#bbcc12",120

This command lets you transition from the controls current foregroundColor to a new foregroundColor in a given amount of time applying an easing effect if you wish to.

You will need to give aeChangeForeColor the following parameters:

pControl: a long reference to the control, card or stack that should change its foregroundColor
pColor: a color reference in any valid revTalk format
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the control that changes the color. Once the transition is done aeChangeForeColorDone is sent to the control.

aeChangeForeColorDone

Callbackmessage sent to a control after it changed its foreground color using aeChangeForeColor.

aeChangeForeColorOfChunk pField,pChunk,pColor,pDuration[,pMethod]
aeChangeForeColorOfChunk the long id of fld 1,"char 2 to 4 of word 2","blue",1000
aeChangeForeColorOfChunk the long id of fld "myField","line 3","red",1000,"overshoot"

This command lets you transition from the fields current chunk Foreground color to a new color in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeForeColorOfChunk the following parameters:

pField: a long reference to the field that contains the chunk to be changed
pChunk: the chunk being changed
pColor: any color reference
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

CallForemessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the field. Once the transition is done aeChangeForeColorOfChunkDone is sent to the field.

aeChangeForeColorOfChunkDone

Callbackmessage sent to a control after it changed its foreground color using aeChangeForeColor.

Handlers in animationEngine:

aeChangeHeight pControl,pNewHeight,pDuration[,pMethod]
aeChangeHeight the long ID of fld "myField",300,1000,"inout"
aeChangeHeight the long ID of grc "grcTest",300,1000,"bounce"

This command lets you transition from the controls current height to a new height in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeHeight the following parameters:

pControl: a long reference to the control, card or stack that should change its height
pNewHeight: the desired new height after the transition
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the control that changes the color. Once the transition is done aeChangeHeightDone is sent to the control.

aeChangeHeightDone

Callbackmessage sent to a control after it changed its height using aeChangeHeight.

aeChangeHscroll pControl,pNewHScroll,pDuration[,pMethod]
aeChangeHscroll the long ID of fld "myField",the formattedHeight of fld "myField",1000,"Out"
aeChangeHscroll the long ID of grp "myGroup",500,1000,"inOut"

This command lets you transition from the controls current horizontal scroll to a new hScroll in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeHScroll the following parameters:

pControl: a long reference to thefield or group that should change its hScroll
pNewHScroll: the desired new value for the hScroll
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to he field or group that changes its hScroll. Once the transition is done aeChangeHScrollDone is sent to the field or group.

aeChangeHScrollDone

Callbackmessage sent to a control after it changed its horizontal scroll using aeChangeHScroll.

Handlers in animationEngine:

aeChangeRect pControl,pNewRect,pDuration[,pMethod]
aeChangeRect the long ID of fld "myField",the topLeft of fld "myField",500,300,1000,"overshoot"
aeChangeRect the long ID of grc "grcTest", 200,200,500,300,1000,"bounce"

This command lets you transition from the controls current rectangle to a new rectangle in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeRect the following parameters:

pControl: a long reference to the control, card or stack that should change its width
pNewRect: the desired new rectangle in any expression revTalk evaluates as a rectangle
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the control that changes the rectangle. Once the transition is done aeChangeRectDone is sent to the control.

aeChangeRectDone

Callbackmessage sent to a control after it changed its rectangle using aeChangeRect.

aeChangeScroll pControl,pNewHScroll,pNewVScroll,pDuration[,pMethod]
aeChangeScroll the long ID of fld "myField",the formattedHeight of fld "myField",the formattedWidth of fld "myField",1000,"Out"
aeChangeScroll the long ID of grp "myGroup",500,300,1000,"inOut"

This command lets you transition from the controls current vertical scroll to a new vScroll in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeVScroll the following parameters:

pControl: a long reference to the field or group that should change its vScroll
pNewHScroll: the desired new value for the hScroll
pNewVScroll: the desired new value for the vScroll
pDuration: the time it should take to do the transition in milliseconds
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the field or group that changes its vScroll. Once the transition is done aeChangeVScrollDone is sent to the field or group.

Handlers in animationEngine:

aeChangeTextSize pValue
send "aeChangeTextSize 3" to field "myField"
send "aeChangeTextSize -1" to field "myField"

Use this handler to change the textsize in a field, keeping the formatting intact. If you have a field with different styles and textSizes the handler grows or shrinks the textsizes relatively, keeping all styling intact. If the smallest size in the field has been reached, the result contains: Smallest size reached

aeChangeThumbposition pScrollbar,pNewThumbposition,pDuration[,pMethod]
aeChangeThumbPosition the long ID of scrollbar 1,30000,1000,"inOut"
aeChangeThumbPosition the long ID of scrollbar 1,0,1000,"bounce"

This command lets you transition from the scrollbars current thumbposition to a new thumbposition in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeHScroll the following parameters:

pControl: a long reference to thefield or group that should change its hScroll
pNewThumbposition: the desired new value for the thumbPosition
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the scrollbar that changes its thumbposition. Once the transition is done aeChangeThumbPositionDone is sent to the scrollbar.

aeChangeVscroll pControl,pNewHScroll,pDuration[,pMethod]
aeChangeVscroll the long ID of fld "myField",the formattedWidth of fld "myField",1000,"Out"
aeChangeVscroll the long ID of grp "myGroup",500,1000,"inOut"

This command lets you transition from the controls current vertical scroll to a new vScroll in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangeVScroll the following parameters:

pControl: a long reference to thefield or group that should change its vScroll
pNewVScroll: the desired new value for the vScroll
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to he field or group that changes its vScroll. Once the transition is done aeChangeVScrollDone is sent to the field or group.

aeChangeVScrollDone

Callbackmessage sent to a control after it changed its vertical scroll using aeChangeVScroll.

Handlers in animationEngine:

aeChangeWidth pControl,pNewwidth,pDuration[,pMethod]
aeChangewidth the long ID of fld "myField",300,1000,"inout"
aeChangewidth the long ID of grc "grcTest",300,1000,"bounce"

This command lets you transition from the controls current width to a new width in a given amount of time, applying an easing effect if you wish to.

You will need to give aeChangewidth the following parameters:

pControl: a long reference to the control, card or stack that should change its width
pNewwidth: the desired new width after the transition
pDuration: the time it should take to do the transition
pMethod: the easing method used to calculate the transition or empty. Valid methods are: in,out,inOut,bounce or overshoot

Callbackmessages to the control:
During the transition aeEnterFrame and aeExitframe are sent to the control that changes the color. Once the transition is done aeChangewidthDone is sent to the control.

aeChangeWidthDone

Callbackmessage sent to a control after it changed its width using aeChangeWidth.

aeCrossFade pControl1,pControl2,pDuration
aeCrossFade the long ID of btn "myButton",the long id of grp "myGroup",2000
aeCrossfade the long ID of stack "myStack",the long ID of stack "myOtherStack",1000

This command lets you fade in a control or stack while another one fades out in a given amount of time.

Callbackmessages to the calling control:
During the transition aeEnterFrame and aeExitframe are sent to the control that calls the fade. Once the transition is done aeCrossfadeDone is sent to the calling control.

aeCrossFadeDone

Callbackmessage sent to a control that called the fade, after the fading control has finished the transition.

aeEnterFrame

Callback handler sent to an object moved by various ae handlers, before the action on the object is being performed. aeEnterFrame is sent with a parameter specifying the handler that called aeEnterFrame.

aeExitFrame

Callback handler sent to an object moved by various ae handlers, after the action on the object is being performed. aeExitFrame is sent with a parameter specifying the handler that called aeExitFrame.

Handlers in animationEngine:

aeFadeIn pControl,pDuration
aeFadeIn the long ID of btn "myButton",2000
aeFadeIn the long ID of stack "myStack",1000

This command lets you fade in a control or stack in a given amount of time. At the beginning the control or stack is hidden and then its blendlevel is set.

Callbackmessages to the calling control:
During the transition aeEnterFrame and aeExitframe are sent to the control that calls the fade. Once the transition is done aeFadeInDone is sent to the calling control.

aeFadeInDone

Callbackmessage sent to a control that called the fade, after the fading control has finished the transition.

aeFadeOut pControl,pDuration
aeFadeOut the long ID of btn "myButton",2000
aeFadeOut the long ID of stack "myStack",1000

This command lets you fade out a control or stack in a given amount of time. At the beginning the control or stack is show and then its blendlevel is set. At the end of the transition the control or stack is hidden and the blendlevel reset.

Callbackmessages to the calling control:
During the transition aeEnterFrame and aeExitframe are sent to the control that calls the fade. Once the transition is done aeFadeOutDone is sent to the calling control.

aeFadeOutDone

Callbackmessage sent to a control that called the fade, after the fading control has finished the transition.

aeGeneral - reserved for internal use

aeLockBackColors
aeLockBackColors
aeChangeBackColor the long ID of grc "myGraphic",127,255,127,1500,"overshoot"
aeChangeBackColor the long ID of grc "myGraphic","#bbcc12",120
aeUnlockBackColors

use this, if you want to perform aeChangeBackColor of chunk on more than one chunk simultanously.

aeLockBackColorOfChunks
aeLockBackColorOfChunks
aeChangeBackColorOfChunk the long id of fld 1,"char 2 to 4 of word 2","blue",1000
aeChangeBackColorOfChunk the long id of fld "myField","line 3","red",1000,"overshoot"
aeUnlockBackColorOfChunks

use this, if you want to perform aeChangeBackColor of chunk on more than one chunk simultanously.

Handlers in animationEngine:

aeLockForeColors
aeLockForeColors
aeChangeForeColor the long ID of grc "myGraphic",127,255,127,1500,"overshoot"
aeChangeForeColor the long ID of grc "myGraphic","#bbcc12",120
aeUnlockForeColors

use this, if you want to perform aeChangeForeColorOfChunkt on more than one control simultanously.

aeLockForeColorOfChunks
aeLockForeColorOfChunks
aeChangeForeColorOfChunk the long id of fld 1,"char 2 to 4 of word 2","blue",1000
aeChangeForeColorOfChunk the long id of fld "myField","line 3","red",1000,"overshoot"
aeUnlockForeColorOfChunks

use this, if you want to perform aeChangeForeColorOfChunkt on more than one control simultanously.

aeLockHeights
aeLockHeights
aeChangeHeight,the long ID of grc "Test",the height of this card,1000,"Bounce"
aeChangeHeight,the long ID of grc "Test2",the height of this card,1000,"Bounce"
aeUnlockHeights

use this, if you want to perform aeChangeHeight on more than one control simultanously.

aeLockRects
aeLockRects
aeChangeRect,the long ID of grc "Test",the rect of this card,1000,"Bounce"
aeChangeRect,the long ID of grc "Test2",the rect of this card,1000,"Bounce"
aeUnlockRects

use this, if you want to perform aeChangeRect on more than one control simultanously.

aeLockThumbPositions
aeLockThumbPositions
aeChangeThumbPosition,the long ID of sb "Test",3000,1000,"Bounce"
aeChangeThumbPosition,the long ID of sb "Test2",0,1000,"inOut"
aeUnlockThumbPositions

use this, if you want to perform aeChangeThumbPosition on more than one scrollbar simultanously.

Handlers in animationEngine:

aeMoveDone

Callback handler sent to an object that has completed its move by aeMoveTo

aeMoveTo pControl,pX,pY,pDuration[,pEasingEffect]
aeMoveTo the long ID of button "myButton",the loc of this card,2000,"inOut"
aeMoveTo the name of button "myButton",150,200,1000,"bounce"

aeMoveTo is a handler that moves Controls or Stacks.
Unlike the other moving methods, there is no need to write a timed script to use aeMoveTo. The command alone is sufficient.

pControl is any valid reference to a control or stack
pX is the x-location of the destination point
pY is the y-location of the destination point
pDuration is the duration of the move in milliseconds
pEasingEffect is one of in,out,inOut,bounce,overshoot (or empty if no easing is wanted.

Callbacks:

aeMoveTo sends callback messages to the moving controls, which might be useful to use in your projects:

aeMoveDone - sent when the object finished moving
aeEnterFrame - sent before the location of the moving object is updated
aeExitFrame - sent after the location of the moving object has been updated

Related command:

aeSetFrameRate - Lets you specify the framerate you want to use with your animation. A framerate of 25 is a good compromise between smoothness and CPU usage.

Attention!

Stacks move relative to screen coordinates, unlike other objects.
aeMoveTo will break when messages are locked. You will not need to write a timer, however, there is a timed message that gets sent by animationEngine to animationEngine. As soon as messages are locked the command will not work. The name of the message being sent is aeGeneral. Make sure not to trap aeGeneral in a front- or backscript

aeSetFrameRate numberOfFramesPerSecond

aeSetFrameRate 25
aeSetFrameRate 12.5
aeSetFrameRate 50

aeSetFrameRate lets you set the desired frames per seconds for animations done with aeMoveTo. Default value is 25 fps.

Handlers in animationEngine:

aeStartListeningForCollisions

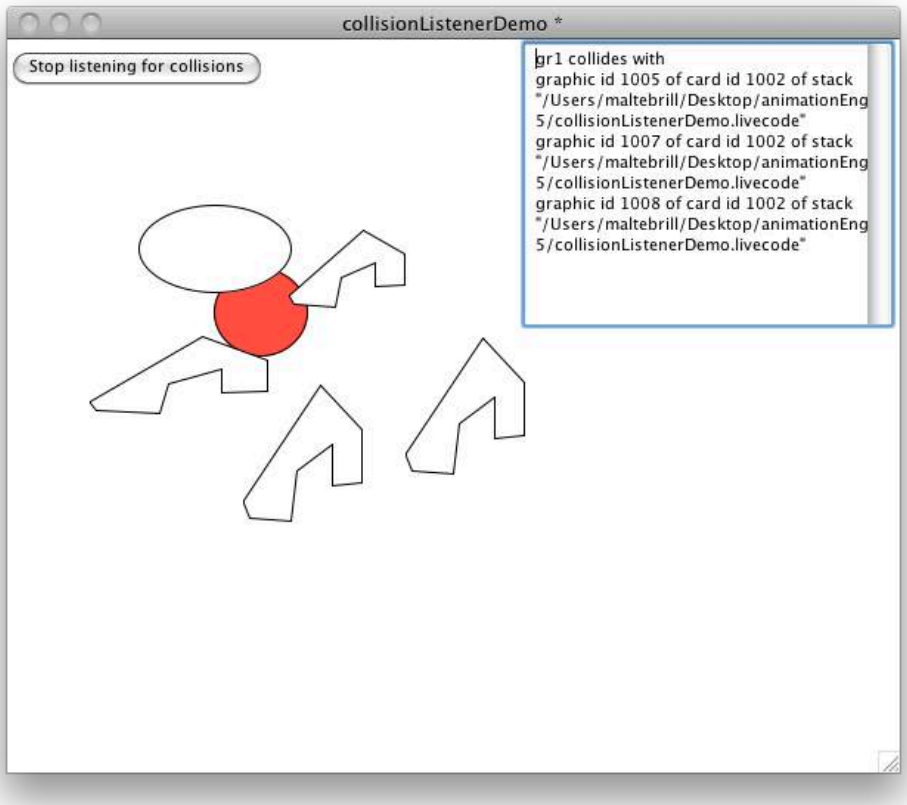
is the command that tells animationEngine to start listening for collisions you between controls you have defined setting up the **aeListenForCollisionsWith** virtual property of the listening control. How often the tests are performed is governed by **aeSetFrameRate**.

Callback messages sent:
aeCollision will be sent to the target control (the one listening for collisions). aeCollision carries a parameter that holds a list of references to all controls colliding with the target.

aeStopListeningForCollisions

is the command that tells animationEngine to stop listening for collisions. It clears the list of all objects listening for collisions internally (set with **aeListenForCollisionsWith**) , so you will have to set up the listeners before you use **aeStartListeningForCollisions**

For more detailed examples, please read page 2 and 3 of this document and take a look at the demo stack "collionListenerDemo.livecode" that comes with the animationEngine 5 download package.



Handlers in animationEngine:

constrainCircularCallBack
on constrainCircularCallBack
-- do something useful
end constrainCircularCallBack

This message is sent to a control whose constrainCircular is set, when it is being dragged. You can use it instead of a mouseMove handler in that control.

constrainCircularExit
on constrainCircularExit
-- do something useful
end constrainCircularExit

This message is sent to a control whose constrainCircular is set, when the mouse is released after a drag. You can use it instead of a mouseUp and mouseRelease handler in that control.

constrainCircularInit
on constrainCircularInit
-- do something useful
end constrainCircularInit

This message is sent to a control whose constrainCircular is set, when the mouse is pressed. You can use it instead of a mouseDown handler in that control.

constrainEllipticalCallBack
on constrainEllipticalCallBack
-- do something useful
end constrainEllipticalCallBack

This message is sent to a control whose constrainElliptical is set, when it is being dragged. You can use it instead of a mouseMove handler in that control.

constrainEllipticalExit
on constrainEllipticalExit
-- do something useful
end constrainEllipticalExit

This message is sent to a control whose constrainElliptical is set, when the mouse is released after a drag. You can use it instead of a mouseUp and mouseRelease handler in that control.

constrainEllipticalInit
on constrainEllipticalInit
-- do something useful
end constrainEllipticalInit

This message is sent to a control whose constrainElliptical is set, when the mouse is pressed. You can use it instead of a mouseDown handler in that control.

Handlers in animationEngine:

constrainLinearCallBack
on constrainLinearCallBack
-- do something useful
end constrainLinearCallBack

This message is sent to a control whose constrainLinear is set, when it is being dragged. You can use it instead of a mouseMove handler in that control.

constrainLinearExit
on constrainLinearExit
-- do something useful
end constrainLinearExit

This message is sent to a control whose constrainLinear is set, when the mouse is released after a drag. You can use it instead of a mouseUp and mouseRelease handler in that control.

constrainLinearInit
on constrainLinearInit
-- do something useful
end constrainLinearInit

This message is sent to a control whose constrainLinear is set, when the mouse is pressed. You can use it instead of a mouseDown handler in that control.

constrainRectangularCallBack
on constrainRectangularCallBack
-- do something useful
end constrainRectangularCallBack

This message is sent to a control whose constrainRectangular is set, when it is being dragged. You can use it instead of a mouseMove handler in that control.

constrainRectangularExit
on constrainRectangularExit
-- do something useful
end constrainRectangularExit

This message is sent to a control whose constrainRectangular is set, when the mouse is released after a drag. You can use it instead of a mouseUp and mouseRelease handler in that control.

constrainRectangularInit
on constrainRectangularInit
-- do something useful
end constrainRectangularInit

This message is sent to a control whose constrainRectangular is set, when the mouse is pressed. You can use it instead of a mouseDown handler in that control.

Handlers in animationEngine:

drawIsoBox x,y,z,A,B,C,originX,originY

```
lock screen
  repeat with j=5 down to 1
    put 80,50+j*15,40+j*30 into tColor
    if item 3 of tColor<50 then add (30+random(20)) to item 3 of tColor
    put item 1 of tColor+27,item 2 of tColor+27,item 3 of tColor+27 into tColor2
    repeat with i=5 down to 1
      put j*30 into tJ
      put i*30 into tI
      drawIsoBox tJ,0,tI,30,random(10)*10+10,30,250,470
      set the backColor of the last grc to tColor
      set the forecolor of the last grc to tColor2
      set the opaque of the last grc to true
    end repeat
  end repeat
  unlock screen
```

This handler creates a box in isometric view at the coordinates x,y,z with a width A, a height B and depth C at the screenCoordinates originX,originY. A new graphic control is created to represent the box on the card.

drawIsoLine x1,y1,z1,x2,y2,z2,originX,originY

```
 lock screen
  drawIsoLine "0,0,0,200,0,0,250,400"
  set the foreColor of the last grc to "red"
  drawIsoLine "0,0,0,0,200,0,250,400"
  set the foreColor of the last grc to 0,125,0
  drawIsoLine "0,0,0,0,0,200,250,400"
  set the foreColor of the last grc to "blue"
  unlock screen
```

Use the drawIsoLine handler to draw lines in isometric view.

Handlers in animationEngine:

moveCircular
is a handler that moves any object on a circular path. Call it repeatedly in a send in time structure to create smooth animation.

Demo:
Create a graphic called "myGraphic"
Create a button with the following Script in it:

```
on mouseUp
  if the flag of me is empty then set the flag of me to false
  set the flag of me to not the flag of me
  hereWeGo
end mouseUp

on hereWeGo
  send moveCircular to grc "myGraphic"
  if the flag of me then
    send "hereWeGo" to me in 20 milliseconds
  end if
end hereWeGo
```

The handler initialises a custom property set for the target with the following properties in it:
centerX - x location of the centerpoint
centerY - y location of the center point
isAngle - the center angle of the circle (in degrees)
isRadius - the radius
step - the direction and speed the object moves on the circle.

You refer to these properties using array notation, e.g. *set the moveCircular["step"] of grc "myGraphic" to 1.5*

moveCircularChangeStepIn - Reserved for internal use.
moveCircularChangeStepInOut - Reserved for internal use.
moveCircularChangeStepOut - Reserved for internal use.

moveCircularEaseIn newStep,duration[,exponent]

```
send "moveCircularEaseIn 2,1000,4" to btn "myCircularMovingButton"
send "moveCircularEaseIn 0,1500" to grc "myCircularMovingGraphic"
```

Use this handler to change the speed of a circular moving control with an ease in effect. You pass the following parameters:

newStep - the new value for the moveCircular["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveCircularEaseInOut newStep,duration[,exponent]

```
send "moveCircularEaseInOut 2,1000,4" to btn "myCircularMovingButton"
send "moveCircularEaseInOut 0,1500" to grc "myCircularMovingGraphic"
```

Use this handler to change the speed of a circular moving control with an ease in and out effect. You pass the following parameters:
newStep - the new value for the moveCircular["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

Handlers in animationEngine:

moveCircularEaseOut newStep,duration[,exponent]
send "moveCircularEaseOut 2,1000,4" to btn "myCircularMovingButton"
send "moveCircularEaseOut 0,1500" to grc "myCircularMovingGraphic"

Use this handler to change the speed of a circular moving control with an ease out effect. You pass the following parameters:

newStep - the new value for the moveCircular["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveCircularReversePath
send "moveCircularReversePath" to btn "myMovingButton"

This handler reverses the direction the control moves on the circular path.

moveElliptical

is a handler that moves any object on a elliptical path. Call it repeatedly in a send in time structure to create smooth animation.

Demo:
Create a graphic called "myGraphic"
Create a button , copy and paste the following Script to it:

```
on mouseUp
  if the flag of me is empty then set the flag of me to false
  set the flag of me to not the flag of me
  hereWeGo
end mouseUp

on hereWeGo
  send moveElliptical to grc "myGraphic"
  if the flag of me then
    send "hereWeGo" to me in 20 milliseconds
  end if
end hereWeGo
```

The handler initialises a custom property set for the target with the following properties in it:
centerX - x location of the centerpoint
centerY - y location of the center point
isAngle - the center angle of the ellipse (in degrees)
radiusX - the x-radius of the ellipse
radiusY - the y-radius of the ellipse
step - the direction and speed the object moves on the ellipse.

You refer to these custom properties using array notation, e.g. *set the moveElliptical["step"] of btn "myButton" to 1.9*

Handlers in animationEngine:

moveEllipticalChangeStepIn - Reserved for internal use.
moveEllipticalChangeStepInOut - Reserved for internal use.
moveEllipticalChangeStepOut - Reserved for internal use.

moveEllipticalEaseIn newStep,duration[,exponent]

send "moveEllipticalEaseIn 2,1000,4" to btn "myCircularMovingButton"
send "moveEllipticalEaseIn 0,1500" to grc "myCircularMovingGraphic"

Use this handler to change the speed of an elliptical moving control with an ease in effect. You pass the following parameters:

newStep - the new value for the moveElliptical["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveEllipticalEaseInOut newStep,duration[,exponent]

send "moveEllipticalEaseInOut 2,1000,4" to btn "myCircularMovingButton"
send "moveEllipticalEaseInOut 0,1500" to grc "myCircularMovingGraphic"

Use this handler to change the speed of an elliptical moving control with an ease in and out effect. You pass the following parameters:

newStep - the new value for the moveElliptical["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveEllipticalEaseOut newStep,duration[,exponent]

send "moveEllipticalEaseOut 2,1000,4" to btn "myCircularMovingButton"
send "moveEllipticalEaseOut 0,1500" to grc "myCircularMovingGraphic"

Use this handler to change the speed of an elliptical moving control with an ease out effect. You pass the following parameters:

newStep - the new value for the moveElliptical["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveEllipticalReversePath

send "moveEllipticalReversePath" to grc "myMovingGraphic"

reverses the direction the control moves on the elliptical path.

Handlers in animationEngine:

moveLinear *DEPRECATED*

is a handler that moves any object on a linear path. Call it repeatedly in a send in time structure to create smooth animation. It is recommended to use aeMoveTo instead of moveLinear, as that method performs much better for multiple objects and timing is more predictable, as aeMoveTo is frame accurate. However it might prove useful if you want to perform a pingpong animation or similar. That said, moveLinear is not being developed further. It will continue to work in upcoming versions, but no new features will be added.

Demo:
Create a graphic called "myGraphic"
Create a button, copy and paste the following script to it:

```
on mouseUp
  if the flag of me is empty then set the flag of me to false
  set the flag of me to not the flag of me
  hereWeGo
end mouseUp

on hereWeGo
  send moveLinear to grc "myGraphic"
  if the flag of me then
    send "hereWeGo" to me in 20 milliseconds
  end if
end hereWeGo
```

The handler initialises a custom property set for the target with the following properties in it:

startPoint - the location of the start point of the line (x,y)
endPoint - the location of the end point of the line
step - the direction and speed the object moves on the path
isDistance - how far the object has moved already (in Pixel)
moveDone- true if the object has completed the move
pingpong - true: object swaps start- and endpoint of the path when move is completed
false: animation starts at original startpoint again

moveLinearChangeStepIn - Reserved for internal use.
moveLinearChangeStepInOut - Reserved for internal use.
moveLinearChangeStepOut -Reserved for internal use.

Handlers in animationEngine:

moveLinearEaseIn newStep,duration[,exponent]

send "moveLinearEaseIn 2,1000,4" to btn "myLinearMovingButton"
send "moveLinearEaseIn 0,1500" to grc "myLinearMovingGraphic"

Use this handler to change the speed of a linear moving control with an ease in effect. You pass the following parameters:

newStep - the new value for the moveLinear["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveLinearEaseInOut newStep,duration[,exponent]
send "moveLinearEaseInOut 2,1000,4" to btn "myLinearMovingButton"
send "moveLinearEaseInOut 0,1500" to grc "myLinearMovingGraphic"

Use this handler to change the speed of a linear moving control with an ease in and out effect. You pass the following parameters:

newStep - the new value for the moveLinear["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveLinearEaseOut newStep,duration[,exponent]
send "moveLinearEaseOut 2,1000,4" to btn "myLinearMovingButton"
send "moveLinearEaseOut 0,1500" to grc "myLinearMovingGraphic"

Use this handler to change the speed of a linear moving control with an ease out effect. You pass the following parameters:

newStep - the new value for the moveLinear["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

moveLinearMoveDone

is a message sent to the object when the endpoint of a linear move is reached. If you want to react on the completition of a move put a handler into the object you are moving.

In the moving object:

```
on moveLinearMoveDone
  play audioclip "fanfare"
end moveLinearMoveDone
```

moveLinearReversePath

send "moveLinearReversepath" to btn "myLinearMovingButton"

is a handler that that swaps start- and enpoint of a linear path.

Handlers in animationEngine:

movePolygonal

is a handler that moves any object on a polygonal path. Call it repeatedly in a send in time structure to create smooth animation.

Demo:
Create a graphic called "myGraphic"
Create a button, copy and paste the following script to it:

```
on mouseUp
  if the flag of me is empty then set the flag of me to false
  set the flag of me to not the flag of me
  hereWeGo
end mouseUp

on hereWeGo
  send movePolygonal to grc "myGraphic"
  if the flag of me then
    send "hereWeGo" to me in 20 milliseconds
  end if
end hereWeGo
```

The handler initialises a custom property set for the target with the following properties in it:

pointList - the points of the polygon (x,y) each one on a seperate line
current - the current endpoint
startPoint - the location of the start point of the current line (x,y)
endPoint - the location of the end point of the current line
step - the direction and speed the object moves on the path
isDistance - how far the object has moved already (on the current line, in Pixel)
moveDone- true if the object has completed the move

You refer to these custom properties using array notation, e.g. *set the movepolygonal["step"] of grc "myGraphic" to 4*

Attention! The animation will loop if you don´t react on moveDone being set to true. If you don´t want a loop try the following to test behaviour:

(continued on next page...)

Handlers in animationEngine:

movePolygonal (continued)

Change the script of the button to:

```
on mouseUp
  if the flag of me is empty then set the flag of me to false
  set the flag of me to not the flag of me
  hereWeGo
end mouseUp

on hereWeGo
  if the movePolygonal["moveDone"] of grc "myGraphic" is true then
    set the loc of grc "myGraphic" to the movePolygonal["endpoint"] of grc "myGraphic"
    set the movePolygonal["moveDone"] of grc "myGraphic" to false
  else
    if the flag of me then
      send movePolygonal to grc "myGraphic"
      send hereWeGo to me in 3 milliseconds
    end if
  end if
end hereWeGo
```

movePolygonalChangeStepIn - Reserved for internal use.
movePolygonalChangeStepInOut - Reserved for internal use.
movePolygonalChangeStepOut - Reserved for internal use.

movePolygonalEaseIn newStep,duration[,exponent]

```
send "movePolygonalEaseIn 2,1000,4" to btn "myPolygonalMovingButton"
send "movePolygonalEaseIn 0,1500" to grc "myPolygonalMovingGraphic"
```

Use this handler to change the speed of a polygonal moving control with an ease in effect. You pass the following parameters:

newStep - the new value for the movePolygonal["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

Handlers in animationEngine:

movePolygonalEaseInOut newStep,duration[,exponent]

send "movePolygonalEaseInOut 2,1000,4" to btn "myPolygonalMovingButton"
send "movePolygonalEaseInOut 0,1500" to grc "myPolygonalMovingGraphic"

Use this handler to change the speed of a polygonal moving control with an ease in and out effect. You pass the following parameters:

newStep - the new value for the movePolygonal["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

movePolygonalEaseOut newStep,duration[,exponent]

send "movePolygonalEaseOut 2,1000,4" to btn "myPolygonalMovingButton"
send "movePolygonalEaseOut 0,1500" to grc "myPolygonalMovingGraphic"

Use this handler to change the speed of a polygonal moving control with an ease out effect. You pass the following parameters:

newStep - the new value for the movePolygonal["step"] of the control
duration - the duration of the ease effect in milliseconds
exponent - optional parameter to achieve cubic, quadric or other easing

movePolygonalMoveDone

is a message sent to the object when the endpoint of a polygonal move is reached. If you want to react on the completition of a move put a handler into the object you are moving.

In the moving object:

```
on movePolygonalMoveDone
  play audioclip "fanfare"
end movePolygonalMoveDone
```

movePolygonalReversePath

send "movePolygonalReversePath" to grc "myGraphic"

This handler reverses the polygonal path of an object.

Properties in animationEngine

aeListenForCollisionsWith pObjectReferences

```
repeat with i = 2 to the number of graphics
  put the long id of graphic i & cr after tList
end repeat
delete char -1 of tList
set the aeListenForCollisionsWith of graphic 1 to tList
```

setting the aeListenForCollisionsWith (virtual) property of a control tells it which controls it should listen for Collisions with.
Attention! This property is not persistent. It is not saved with your stack.

aspectScale left,top,right,bottom[,setLockLoc]

```
set the aspectScale of img "myImage" 1 to 100,100,400,400,true
set the aspectScale of fld "myField" to the rect of grc "myGraphic"
```

setting the aspectScale of a control scales it in the correct width-to-height ratio to fit in the specified rectangle with maximum extend. You can specify if the lockLoc of the control should be set to avoid unwanted resizing when closing/reopening cards.

constrainCircular centerX,centerY,radius

```
set the constrainCircular of btn "yourButton" to 100,100,50
```

Allows dragging a control on a circle. The control can not be dragged outside of that circle.
Attention! You need to make sure that the following messages are not trapped in the target control:
mouseDown, mouseUp, mouseRelease, mouseMove. If you need to use these messages make sure that you pass them.
You might use the following messages instead to avoid conflicts:
mouseDown / touchStart - **constrainCircularInit**
mouseUp / touchEnd - **constrainCircularExit**
mouseRelease / touchRelease - **constrainCircularExit**
mouseMove / touchMove - **constrainCircularCallback**

constrainElliptical centerX,centerY,radiusX,radiusY

```
set the constrainElliptical of btn "yourButton" to 100,100,50,100
```

Allows dragging a control on an ellipse. The control can not be dragged outside of that ellipse.
Attention! You need to make sure that the following messages are not trapped in the target control:
mouseDown, mouseUp, mouseRelease, mouseMove. If you need to use these messages make sure that you pass them.
You might use the following messages instead to avoid conflicts:
mouseDown /touchStart - **constrainEllipticalInit**
mouseUp / touchEnd - **constrainEllipticalExit**
mouseRelease / touchRelease - **constrainEllipticalExit**
mouseMove /touchMove - **constrainEllipticalCallback**

constrainLinear x1,y1,x2,y2

```
set the constrainLinear of btn "yourButton" to 100,100,200,200
```

Allows dragging a control on a line. The control can not be dragged outside of that line.
Attention! You need to make sure that the following messages are not trapped in the target control:
mouseDown, mouseUp, mouseRelease, mouseMove. If you need to use these messages make sure that you pass them.
You might use the following messages instead to avoid conflicts:
mouseDown / touchStart - **constrainLinearInit**
mouseUp /touchEnd - **constrainLinearExit**
mouseRelease / touchRelease - **constrainLinearExit**
mouseMove / touchMove - **constrainLinearCallback**

Properties in animationEngine

constrainRectangular left,top,right,bottom
set the constrainRectangular of btn "myButton" to 100,100,250,300
set the constrainRectangular of btn "myButton" to the rect of grc "test"

Allows a drag of a control in a rectangle specified by left,top,right,bottom. The control can not be dragged outside of that rectangle. The first variant of that script has been posted to the use-revolution list by Scott Rossi.

Attention! You need to make sure that the following messages are not trapped in the target control:
mouseDown, mouseUp, mouseRelease, mouseMove. If you need to use these messages make sure that you pass them.

You might use the following messages instead to avoid conflicts:
mouseDown / touchStart - constrainRectangularInit
mouseUp /touchEnd - constrainRectangularExit
mouseRelease / touchRelease - constrainRectangularExit
mouseMove / touchMove - constrainRectangularCallback

Multitouch constraining:
Previous versions of AE relied on mouse messages being sent to constrained controls using either the constrainLinear, constrainCircular, constrainRectangular or constrainElliptical properties. Major parts of the library have been reworked to support constrained dragging of controls on mobile devices. The implementation in animationEngine now allows dragging multiple controls at the same time on mobile devices, as long as multiTouch messages are supported. No changes to the syntax were necessary. You can simply set any of the constrain properties and have all of them react on the device

spiral cx,cy,step,windings(,accuracy)
(graphics only)
set the spiral of grc "yourGrafic" to 200,200,1,2
set the spiral of grc "yourGrafic" to 200,200,-0.07,5,"draft"

if you set the spiral of a graphic it will become a spiral.
cX is the X location of the centerpoint
cY is the yLocation of the centerpoint
step should be a number between -1 and 1
windings is the number of windings of the spiral
accuracy is a number between 1 and 36 or one of this words: best/high/low/draft

superpath m,xScale,yScale
(graphics only)
set the superpath of grc "myGraphic" to 7,100,100
set the superpath of grc "myGraphic" to 3,50,50

is a simplified version of the superShape property. The graphics created can be used as a path for movepolygonal.

supershape m,n1,n2,n3,xScale,yScale,iterations
(graphics only)
set the supershape of grc "myGraphic" to 7,1,1,1,100,100,36000
set the supershape of grc "myGraphic" to 3,1,1.289,0.879,50,50,360

It allows you to create beautyful graphics by script.