

Using Fuse



A complete animation, sequencing & BitmapFilter toolset for Flash

MosesSupposes.com

What is the Fuse Kit?

The necessity of animating with code can often result in reams of script to accomplish simple things.

Fuse proposes clear and compact syntax with timeline-like sophistication and great runtime flexibility.

This sample Fuse moves a box in a looping square, with bubble-curved motion paths:

```
var f:Fuse = new Fuse( { _x:'100',  controlY:'-50' },
                      { _y:'100',  controlX:'50'  },
                      { _x:'-100', controlY:'50'  },
                      { _y:'-100', controlX:'-50' });
f.target = box_mc;
f.start();
```

Beginners: if this looks difficult, try out Fuse's Simple Syntax & Shortcuts to get you started.

Fuse is smart and parses the contents of each action, filling in things you may have omitted – note how complete bezier curves have been generated simply by adding a single control point parameter, and relative positioning has been specified by setting values to strings.

You will of course create more involved Fuses, but the code you write will always be as legible.

Tell me more...

MosesSupposes.com's Fuse Kit is an example of Open Source in action, bringing together ideas of Moses Gunesch from the States, Ladislav Zigo of Slovakia and the authors of FMP from Germany. Fuse is now being used in over sixty nations by some of the world's top firms – RGA, Avenue A, Organic, Euro RSCG, Chopping Block, Schematic, Firstborn, Frog Design, NBC and many others.

Adobe is currently updating its animation classes for Flash 9 using AS3 which will lead to new ways of scripting animation. In recent years however, developers have been left to overcome some significant pitfalls of AS2 such as its slower processing speed. A popular tween engine by Ladislav Zigo was rewritten as the core of Fuse because of its efficiency in handling many tweens at once across a whole SWF published to FP8 or lower.

Pains were taken in building the Fuse Kit to provide standards-leaning coders a professional solution – the kit is MTASC-strict compliant and prototypes are not altered except on demand – while designer-friendly features have been maintained as add-on options. The kit is a Swiss army knife – but only the features you pick are published to the SWF.

The main bits are :

ZigoEngine – Handles all scripted animation across your SWF.

FuseFMP – Simplifies BitmapFilters. Use alone, or tween filters with the engine.

Fuse – Sequencer that extends Array. Use alone or with other elements.

Other goodies include :

FuseItem – Registering this class lets you tween from the engine with *Object Syntax*.

Shortcuts – Extend prototypes or add/remove shortcuts like *alphaTo* to individual targets.

PennerEasing – R. Penner's original full set of easing styles (or use *mx.transitions.easing*)

For more information visit mosessupposes.com/Fuse -or- osflash.org/fuse

Getting set up

Component bundles are offered for IDE users, and most classes can be used immediately with a single import. You can then use one of two setup commands to optionally link the classes with ZigoEngine.

Developers: Features are added by registering classes to ZigoEngine.

```
import com.mosesSupposes.fuse.*;
ZigoEngine.register(FuseItem);
```

Designers: To use Shortcut features (aka 'tween prototypes'), use *simpleSetup* or components instead.

```
import com.mosesSupposes.fuse.*;
ZigoEngine.simpleSetup(Shortcuts, PennerEasing, Fuse, FuseFMP);
```

Again the idea is "mix and match." One project may call for ZigoEngine or FuseFMP alone, others may call for the full meal deal if you're doing complex sequencing with filter tweens.

For easing you may use the *mx.transitions.easing*, your own custom equations, or register PennerEasing for shortcut strings like "easeInOutCubic".

Designers or those seeking easy access usually opt for Shortcuts – *DropShadow_distanceTo*, etc.



Making Stuff Move

The ZigoEngine class supports a wide variety of pseudo properties – *_brightness*, *_bezier_*, *_frame*, *_tintPercent* - that may be tweened in addition to standard properties like *_x*. Any custom property or variable in any object can also be tweened. Once FuseFMP is registered you can tween any BitmapFilter property using the convention *Blur_blurX*, *DropShadow_distance*, etc.

doTween (params are: target, properties, end-values, seconds, easing, delay, callback)

```
ZigoEngine.doTween(box_mc, "_alpha", 0, 2, "easeInExpo");
```

doTween using Object Syntax (FuseItem should be registered):

```
ZigoEngine.doTween(({target:box_mc, _alpha:0, seconds:2, ease:"easeInExpo"}));
```

Shortcut example (params vary per shortcut):

```
box_mc.alphaTo(0, 2, "easeInExpo"); // 2-second fadeout w/extreme easing
```

Any of these could use *mx.transitions.Strong.easeInOut*.

ZigoEngine contains a great number of features not shown above.

- Controls like pausing & removing tweens have options to specify multiple targets/properties
- Easy to use built-in start, update, and end callbacks
- Events for start, update, and end plus a special *onTweenInterrupt* event dispatched by engine
- *cycles* param yoyos tweens back and forth a specific number of times or infinitely
- *skipLevel* param controls how motionless or durationless tweens are handled

Know your Relatives

In the Fuse Kit, using a String value such as "100" instead of a 100 yields *relative* end-values.

Instead of sliding to 100 px, tweening `_x` to "100" would move the clip 100px right of its current position.

This trick has a lot of valuable applications:

- Any "hudge" animation
- Rotate a clip counter-clockwise – for 45 degrees CCW, use "-45".
- Affect a motionless tween with "0" – a trick that can come in handy.
- Bezier tweens are easy to write with relative values. (params `x`, `y`, `controlX`, `controlY`)

```
ZigoEngine.doTween(box_mc,"_bezier_", {x:"100",controlY:"-300"});  
box_mc.bezierTo("100","0","0","-300"); // shortcut syntax
```



Easy BitmapFilters with FuseFMP

FuseFMP's methods can generate and remove filters, retrieve filters from a target, and quickly set filter properties using strings like "Blur" to indicate a BlurFilter and `Blur_blurX` to target properties. You can use FuseFMP without anything else from the kit. Example of standalone usage :

```
FuseFMP.writeFilter(box_mc, "Blur", { blurX:10, blurY:0 });
```

Shortcuts can be optionally added :

```
FuseFMP.simpleSetup(); // one-time setup not needed if using component or ZigoEngine  
  
box_mc.Blur_blurX = 10;  
box_mc.Blur_blurY = 0;  
box_mc.writeFilter("DropShadow");
```

To tween filters easily using ZigoEngine or Fuse, be sure FuseFMP is included in your setup command :

```
import com.mosesSupposes.fuse.*;  
ZigoEngine.register(FuseFMP); // one-time setup
```

FuseFMP + ZigoEngine :

```
FuseFMP.writeFilter(box_mc, "Blur", { blur:0 });  
ZigoEngine.doTween(box_mc, "Blur_blurX", 10, 2, "easeInExpo");
```

FuseFMP + Object Syntax :

```
ZigoEngine.doTween({target:box_mc, DropShadow_distance:10, ease:"easeOutBounce"});
```

FuseFMP Shortcuts + ZigoEngine :

```
box_mc.DropShadow_distanceTo(10, 2, "easeOutBounce");
```



Fuse : When to use?

- Fuse is a sequencer. It can be used alone to generate sequences of timed method-calls and events, or combined with the engine to handle animation.
- Examples might include : the initial rendering of a website's screen layout. Or a custom slideshow player that loads assets dynamically then plays an effects sequence for each slide.
- Fuse can also be used to create motion graphics, but with a coder's edge – building layouts dynamically around dynamically loaded content and so on.

For small numbers of tweens, or for generic UI animation that's interrupted a lot by interactivity, using the ZigoEngine without Fuse is a much better choice.

Fuse Simple Syntax

Fuse's Simple Syntax option is great for two audiences:

1. Beginning coders – with a few extra commands you can build sequences of tweens.

```
var f = Fuse.open();
box1_mc.tween("_y", "10", .5);
Fuse.openGroup();
  box1_mc.tween("_x", "40", .5, "easeInOutBack");
  box2_mc.tween("_x", "50", .5, "easeInOutBack", .1);
Fuse.closeGroup();
box2_mc.tween("_y", "20", .5);
Fuse.closeAndStart();
```

2. Programmers – for enforcing event order w/ optional timers, this syntax outlines method-calls neatly. Short delays let bound classes instantiate assets and initial rendering to complete at each step.

```
Fuse.open();
Fuse.addCommand(this, "traceOutput", "Start setup...");
Fuse.addCommand(_oMainMenu, "init", _mcScope, menuItemNodes);
Fuse.addCommand("delay", .25);
Fuse.addCommand(_mcContentPage, "create", _mcScope, "contentPage_mc");
Fuse.addCommand("delay", .25);
Fuse.addCommand(_oController, "setupAndBegin", _oMainMenu, _mcContentPage);
Fuse.addCommand(this, "traceOutput", "Setup complete.");
Fuse.closeAndStart();
```

Fuse Object Syntax

Fuse's smart parser reads loose parameters from generic objects. Parameters that you *imply but omit* are filled in automatically, which further simplifies and condenses the code you need to write. Each action is interpreted at runtime in the sequence and you may query for values on-the-fly.

Example 1 – an animation action

```
{  target : box_mc,
  start_visible : true,
  start_alpha : 0,
  start_x : "-100",
  rotation : 360,
  seconds : .5,
  delay : 1,
  ease : "easeInOutQuint" // or you use mx.transitions.Strong
}
```

Translation: *After a 1-second delay, turn visibility on and fade in box_mc sliding from 100px left of where it sits now back to its current position and flipping it around, in half a second, with strong in-out easing style.*

What do you think: Is the Fuse code actually easier to read than the description?

Some of Fuse's smart-parsing rules for animation :

- You may omit underscores for known properties, unless you've turned that feature off.
- You may set start values by adding the prefix "start_" to any property
- You may omit start or end values and they will be assumed.
- Props like *_alpha* will assume their 'natural' end value such as 100
- Props like *_x* will return to their current position if only given a start value
- Flexible targeting (Fuse instance defaults, action-overrides and action-addTargets)
- Boolean values may be set (at start & end of tweens)
- Actions can contain a *label* property that can be used during *skipTo* calls

Example 2 – firing callbacks, custom events

Besides the ability to fire callbacks, a Fuse may be used quite effectively as an event-dispatching utility.

```
{  delay : 1,
  // fire a local callback
  scope : this,
  func : "drawSubMenu",
  args : [menu1_mc, submenuItems, true]
  // dispatch a custom event from the Fuse instance
  event : "onSubMenuDrawn",
  eventparams : { submenu : menu1_mc }
}
```

Related capabilities include:

- Define a default scope for a Fuse instance and it will be adopted by all callbacks
- Tweens may be mixed in with callbacks and events in the same action
- Start, update and end callbacks are grouped to the action regardless of how many targets, props
- Inline play control using the command parameter such as `{ command : "start" }` to loop a Fuse.

Building Sequences with Object Syntax

Fuse extends Array and each Fuse is a playable actions list.

Since Fuse extends Array this part is easy. All applicable array methods are supported.

```
var f = new Fuse();
f.push({ target:box_mc, start_scale:0, delay:.5 });
f.push({ target:box2_mc, brightOffset:100 });
f.start();
```

Play commands include start, stop, pause, resume, and skipTo.

Fuse dispatches the events onStart, onAdvance, onStop, onPause, onResume, and onComplete.

```
var f = new Fuse();
f.label = "examplefuse"; // label is a convenience for traceItems() calls.
f.autoClear = true; // set instance to self-destroy after completion
f.scope = this;
f.target = box_mc;

f.push({ delay:2, func:"trace", args:"First item done."});
f.push({ start_scale:0, delay:.5 });
f.push({ brightOffset:100 });

f.addEventListener("onComplete", myListenerObject);
f.traceItems(); // dump Fuse to output window or logger
f.start(true);
```

Start properties can be set on all or specific items using *setStartProps* or by passing parameters to *start*. Indices use a 0 base since Fuses are Arrays; or pass the *label* property assigned to an action as in one of the following:

```
f.setStartProps(); // set all start properties
f.setStartProps(0,3,"menufade"); // specify indices
f.start(true); // set all start properties
f.start(0,3,"menufade"); // specify indices
```

Grouping actions and triggering advance

Block actions with a simple generic Array square braces. To jump to the next Fuse action before all items have completed, use a *trigger* parameter.

```
var f:Fuse = new Fuse(
[// begin a group of 3 actions using Array hard-bracket
{ start_x : "-100", seconds : 1, delay : .25, ease : "easeOutQuint" },
{ scope : FuseFMP, func : "writeFilter", args : [box_mc, "Blur", { blur : 0 } ] },
{ Blur_blurX : 20, seconds : .5, ease : "easeInQuad", trigger : true }
], // end group
// The next action happens after the trigger in the previous group.
{ Blur_blurX : 0, seconds : .3, ease : "easeOutQuad" }
);
f.target = box_mc;
f.start();
```

Querying values at runtime

This is achieved by setting a value to a function. That function is queried at runtime in sequence play.

```
{  target : function(){ return _aCurrentTargets; },
  start_x : function(){ return (Math.random()*Stage.width) },
  start_y : function(){ return (Math.random()*Stage.height) },
  start_alpha : 0,
  ease : getRandomEasing, // a function elsewhere in your code
  seconds : function() { return (Math.random()*2); }
}
```

Applying reused actions

This example illustrates the reuse of a common fade action using the *action* property.

```
var slowFadeOut:Object = { alpha:0, duration:3, ease:"easeInQuint" };
someFuse.push({target:nav_mc, action:slowFadeOut, delay:.5 });
someOtherFuse.push({target:page_mc, action:slowFadeOut });
```

Nesting Fuses

You may include other Fuse instances within a Fuse.

```
var f1:Fuse = new Fuse(action1, action2, action3);
var f2:Fuse = new Fuse(action4, f1, action5);
f2.start();
```

For more information visit mosessupposes.com/Fuse -or- osflash.org/fuse