**Display Object Child**

**function** addChild(child:DisplayObject):void;

**function** addChildAt(child:DisplayObject, index:int):void;

**function** contains(child:DisplayObject):Boolean;

**function** getChildAt(index:int):DisplayObject;

**function** getChildIndex(child:DisplayObject):int;

**function** removeChild(child:DisplayObject, dispose:Boolean=**false**):void;

**function** removeChildAt(index:int, dispose:Boolean=**false**):void;

**function** swapChildren(child1:DisplayObject, child2:DisplayObject):void;

**function** swapChildrenAt(index1:int, index2:int):void;

**Embed And Use Texture**

Embed(source = "background.png")]

**private** **static** **const** BackgroundBmp:Class;

**private** **var** \_background:Image;

**var** bgTexture:Texture = Texture.fromEmbeddedAsset(BackgroundBmp);

\_background = **new** Image(bgTexture);

addChild(\_background);

**Remove An Object**

msgBox.removeFromParent();

msgBox.dispose();

OR

msgBox.removeFromParent(**true**);

Pivot And Rotate Around Center

**var** image:Image = **new** Image(texture);

image.pivotX = image.width / 2.0;

image.pivotY = image.height / 2.0;

image.rotation = deg2rad(45);

or

**var** image:Image = **new** Image(texture);

image.alignPivot();

image.rotation = deg2rad(45);

or

**var** image:Image = **new** Image(texture);

image.alignPivot(Align.RIGHT, Align.BOTTOM);

image.rotation = deg2rad(45);

Texture

**public** **class** Texture

{

**static** **function** fromColor():Texture;

**static** **function** fromBitmap():Texture;

**static** **function** fromBitmapData():Texture;

**static** **function** fromEmbeddedAsset():Texture;

**static** **function** fromCamera():Texture;

**static** **function** fromNetStream():Texture;

**static** **function** fromTexture():Texture;

}

[Embed(source="mushroom.png")]

**public** **static** **const** Mushroom:Class;

**var** bitmap:Bitmap = **new** Mushroom();

**var** texture:Texture = Texture.fromBitmap(bitmap);

Beter:

[Embed(source="mushroom.png")]

**public** **static** **const** Mushroom:Class;

**var** texture:Texture = Texture.fromEmbeddedAsset(Mushroom);

Texture from Texture

**var** texture:Texture = getTexture();

**var** subTexture:Texture = Texture.fromTexture(texture, **new** Rectangle(10, 10, 41, 47));

Image from Texture

*// do NOT do this!!*

**var** image1:Image = **new** Image(Texture.fromEmbeddedAsset(Mushroom));

**var** image2:Image = **new** Image(Texture.fromEmbeddedAsset(Mushroom));

**var** image3:Image = **new** Image(Texture.fromEmbeddedAsset(Mushroom));

*// instead, create the texture once and keep a reference:*

**var** texture:Texture = Texture.fromEmbeddedAsset(Mushroom));

**var** image1:Image = **new** Image(texture);

**var** image2:Image = **new** Image(texture);

**var** image3:Image = **new** Image(texture);

Texture Atlases

XML

**<TextureAtlas** imagePath="atlas.png"**>**

**<SubTexture** name="moon" x="0" y="0" width="30" height="30"**/>**;

**<SubTexture** name="jupiter" x="30" y="0" width="65" height="78"**/>**;

...

**</TextureAtlas>**;

Using the Atlas

[Embed(source="atlas.xml", mimeType="application/octet-stream")]

**public** **static** **const** AtlasXml:Class;

[Embed(source="atlas.png")]

**public** **static** **const** AtlasTexture:Class;

Get Texture

**var** texture:Texture = Texture.fromEmbeddedAsset(AtlasTexture);

**var** xml:XML = XML(**new** AtlasXml());

**var** atlas:TextureAtlas = **new** TextureAtlas(texture, xml);

**var** moonTexture:Texture = atlas.getTexture("moon");

**var** moonImage:Image = **new** Image(moonTexture);

Render Textures

**var** renderTexture:RenderTexture = **new** RenderTexture(512, 512);

**var** brush:Sprite = getBrush();

brush.x = 40;

brush.y = 120;

brush.rotation = 1.41;

renderTexture.draw(brush);

Use Like Normal Texture

**var** image:Image = **new** Image(renderTexture);

addChild(image);

Many Texture Rendering

renderTexture.drawBundled(**function**():**void**

{

**for** (**var** i:int=0; i<numDrawings; ++i)

{

image.rotation = (2 \* Math.PI / numDrawings) \* i;

renderTexture.draw(image);

}

});

Erase parts of a render texture

brush.blendMode = BlendMode.ERASE;

renderTexture.draw(brush);

Event & EventDispatcher

**class** Dog **extends** Sprite

{

**function** advanceTime():void

{

**if** (timeToPee)

{

**var** event:Event = **new** Event("bark");

dispatchEvent(event);

}

}

}

**var** einstein:Dog = **new** Dog();

einstein.addEventListener("bark", onBark);

**function** onBark(event:Event):void

{

einstein.walk();

}

Remove the Event Listener

einstein.removeEventListener("bark", onBark);

OR

einstein.removeEventListeners("bark");

Custom Events

**public** **class** BarkEvent **extends** Event

{

**public** **static** **const** BARK:String = "bark";

**private** **var** \_reason:String;

**public** **function** BarkEvent(type:String, reason:String, bubbles:Boolean=**false**)

{

**super**(type, bubbles);

\_reason = reason;

}

**public** **function** **get** reason():Boolean { **return** \_reason; }

}

**class** Dog **extends** Sprite

{

**function** advanceTime():void

{

**var** reason:String = **this**.hungry ? "hungry" : "pee";

**var** event:BarkEvent = **new** BarkEvent(BarkEvent.BARK, reason);

dispatchEvent(event);

}

}

**var** einstein:Dog = **new** Dog();

einstein.addEventListener(BarkEvent.BARK, onBark);

**function** onBark(event:BarkEvent):void

{

**if** (event.reason == "hungry")

einstein.feed();

**else**

einstein.walk();

}

**Simplifying**

Replace the BarkEvent logic with this:

*// create & dispatch event*

**var** event:Event = **new** Event(Dog.BARK);

event.data = "hungry";

dispatchEvent(event);

*// listen to event*

einstein.addEventListener(Dog.BARK, onBark);

**function** onBark(event:Event):void

{

trace("reason: " + event.data **as** String);

}

Starling shortcuts to simplify

*// create & dispatch event*

dispatchEventWith(Dog.BARK, **false**, "hungry");

*// listen to event*

einstein.addEventListener(Dog.BARK, onBark);

**function** onBark(event:Event, reason:String):void

{

trace("reason: " + reason);

}

The simplified dispatchEventWith call is actually even more memory efficient, since Starling will pool the Event objects behind the scenes.

**Simplest:**

*// create & dispatch event*

dispatchEventWith(Dog.HOWL);

*// listen to event*

dog.addEventListener(Dog.HOWL, onHowl);

**function** onHowl():void

{

trace("hoooh!");

}

Useful Event Types

* **Event.TRIGGERED:** a button was triggered
* **Event.ADDED:** a display object was added to a container
* **Event.ADDED\_TO\_STAGE:** a display object was added to a container that is connected to the stage
* **Event.REMOVED:** a display object was removed from a container
* **Event.REMOVED\_FROM\_STAGE:** a display object lost its connection to the stage
* **Event.ENTER\_FRAME:** some time has passed, a new frame is rendered (we’ll get to that later)
* **Event.COMPLETE:** something (like a MovieClip instance) just finished

**Touch Events**

**multitouch**

Starling.multitouchEnabled = **true**;

**var** starling:Starling = **new** Starling(Game, stage);

starling.simulateMultitouch = **true**;

simulateMultitouch. If you enable it, you can simulate multitouch input with your mouse on your development computer. Press and hold the Ctrl or Cmd keys (Windows or Mac) when you move the mouse cursor around to try it out. Add Shift to change the way the alternative cursor is moving.

sprite.addEventListener(TouchEvent.TOUCH, onTouch);

**private** **function** onTouch(event:TouchEvent):void

{

**var** touch:Touch = event.getTouch(**this**, TouchPhase.BEGAN);

**if** (touch)

{

**var** localPos:Point = touch.getLocation(**this**);

trace("Touched object at position: " + localPos);

}

}

Touch Phases

|  |  |
| --- | --- |
| TouchPhase.HOVER | Only for mouse input; dispatched when the cursor moves over the object with the mouse button *up*. |
| TouchPhase.BEGAN | The finger just hit the screen, or the mouse button was pressed. |
| TouchPhase.MOVED | The finger moves around on the screen, or the mouse is moved while the button is pressed. |
| TouchPhase.STATIONARY | The finger or mouse (with pressed button) has not moved since the last frame. |
| TouchPhase.ENDED | The finger was lifted from the screen or from the mouse button. |

Multitouch

**var** touches:Vector.<Touch> = event.getTouches(**this**, TouchPhase.MOVED);

**if** (touches.length == 1)

{

*// one finger touching (or mouse input)*

**var** touch:Touch = touches[0];

**var** movement:Point = touch.getMovement(**this**);

}

**else** **if** (touches.length >= 2)

{

*// two or more fingers touching*

**var** touch1:Touch = touches[0];

**var** touch2:Touch = touches[1];

*// ...*

}

<https://github.com/Gamua/Starling-Framework/blob/master/samples/demo/src/utils/TouchSheet.as>

|  |
| --- |
| package utils |
|  | { |
|  | import flash.geom.Point; |
|  |  |
|  | import starling.display.DisplayObject; |
|  | import starling.display.Sprite; |
|  | import starling.events.Touch; |
|  | import starling.events.TouchEvent; |
|  | import starling.events.TouchPhase; |
|  |  |
|  | public class TouchSheet extends Sprite |
|  | { |
|  | public function TouchSheet(contents:DisplayObject=null) |
|  | { |
|  | addEventListener(TouchEvent.TOUCH, onTouch); |
|  | useHandCursor = true; |
|  |  |
|  | if (contents) |
|  | { |
|  | contents.x = int(contents.width / -2); |
|  | contents.y = int(contents.height / -2); |
|  | addChild(contents); |
|  | } |
|  | } |
|  |  |
|  | private function onTouch(event:TouchEvent):void |
|  | { |
|  | var touches:Vector.<Touch> = event.getTouches(this, TouchPhase.MOVED); |
|  |  |
|  | if (touches.length == 1) |
|  | { |
|  | // one finger touching -> move |
|  | var delta:Point = touches[0].getMovement(parent); |
|  | x += delta.x; |
|  | y += delta.y; |
|  | } |
|  | else if (touches.length == 2) |
|  | { |
|  | // two fingers touching -> rotate and scale |
|  | var touchA:Touch = touches[0]; |
|  | var touchB:Touch = touches[1]; |
|  |  |
|  | var currentPosA:Point = touchA.getLocation(parent); |
|  | var previousPosA:Point = touchA.getPreviousLocation(parent); |
|  | var currentPosB:Point = touchB.getLocation(parent); |
|  | var previousPosB:Point = touchB.getPreviousLocation(parent); |
|  |  |
|  | var currentVector:Point = currentPosA.subtract(currentPosB); |
|  | var previousVector:Point = previousPosA.subtract(previousPosB); |
|  |  |
|  | var currentAngle:Number = Math.atan2(currentVector.y, currentVector.x); |
|  | var previousAngle:Number = Math.atan2(previousVector.y, previousVector.x); |
|  | var deltaAngle:Number = currentAngle - previousAngle; |
|  |  |
|  | // update pivot point based on previous center |
|  | var previousLocalA:Point = touchA.getPreviousLocation(this); |
|  | var previousLocalB:Point = touchB.getPreviousLocation(this); |
|  | pivotX = (previousLocalA.x + previousLocalB.x) \* 0.5; |
|  | pivotY = (previousLocalA.y + previousLocalB.y) \* 0.5; |
|  |  |
|  | // update location based on the current center |
|  | x = (currentPosA.x + currentPosB.x) \* 0.5; |
|  | y = (currentPosA.y + currentPosB.y) \* 0.5; |
|  |  |
|  | // rotate |
|  | rotation += deltaAngle; |
|  |  |
|  | // scale |
|  | var sizeDiff:Number = currentVector.length / previousVector.length; |
|  | scaleX \*= sizeDiff; |
|  | scaleY \*= sizeDiff; |
|  | } |
|  |  |
|  | var touch:Touch = event.getTouch(this, TouchPhase.ENDED); |
|  |  |
|  | if (touch && touch.tapCount == 2) |
|  | parent.addChild(this); // bring self to front |
|  |  |
|  | // enable this code to see when you're hovering over the object |
|  | // touch = event.getTouch(this, TouchPhase.HOVER); |
|  | // alpha = touch ? 0.8 : 1.0; |
|  | } |
|  |  |
|  | public override function dispose():void |
|  | { |
|  | removeEventListener(TouchEvent.TOUCH, onTouch); |
|  | super.dispose(); |
|  | } |
|  | } |
|  | } |

Mouse Out and End Hover

**var** touch:Touch = event.getTouch(**this**);

**if** (touch == **null**)

resetButton();

EnterFrameEvent

**public** **function** CustomObject()

{

addEventListener(Event.ENTER\_FRAME, onEnterFrame);

}

**private** **function** onEnterFrame(event:Event, passedTime:Number):void

{

trace("Time passed since last frame: " + passedTime);

bird.advanceTime(passedTime);

}

Tweens

**var** tween:Tween = **new** Tween(ball, 0.5);

tween.animate("x", 20);

tween.animate("scale", 2.0);

tween.animate("alpha", 0.0);

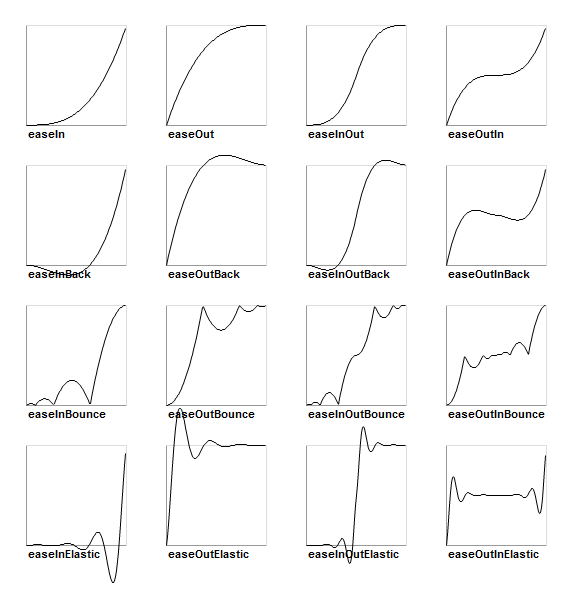
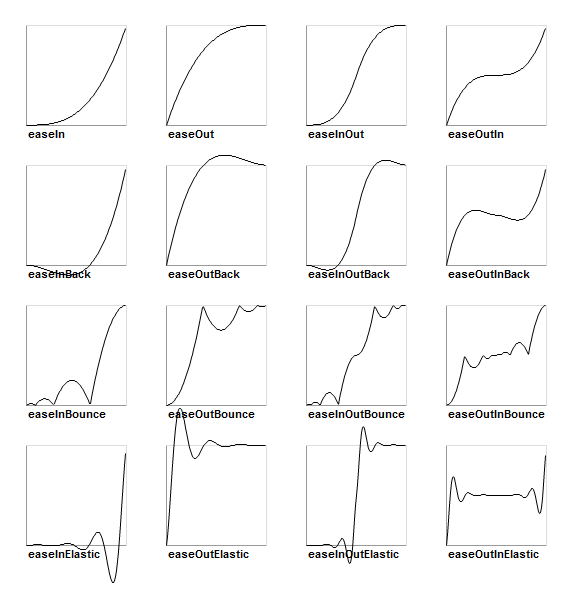
or

tween.moveTo(20, 0); *// animate "x" and "y"*

tween.scaleTo(2); *// animate "scale"*

tween.fadeTo(0); *// animate "alpha"*

Ease



The default, linear, was omitted.

Capabilities

**var** tween:Tween = **new** Tween(ball, 0.5, Transitions.EASE\_IN);

tween.onStart = **function**():**void** { */\* ... \*/* };

tween.onUpdate = **function**():**void** { */\* ... \*/* };

tween.onComplete = **function**():**void** { */\* ... \*/* };

tween.delay = 2;

tween.repeatCount = 3;

tween.reverse = **true**;

tween.nextTween = explode;

Execute Tween

ball.x = 0;

tween = **new** Tween(ball, 1.0);

tween.animate("x", 100);

tween.advanceTime(0.25); *// -> ball.x = 25*

tween.advanceTime(0.25); *// -> ball.x = 50*

tween.advanceTime(0.25); *// -> ball.x = 75*

tween.advanceTime(0.25); *// -> ball.x = 100*

or

**Juggler**

Starling.juggler.add(tween);

Custom Juggler

**public** **class** Game

{

**private** **var** \_gameArea:GameArea;

**private** **function** onEnterFrame(event:Event, passedTime:Number):void

{

**if** (activeMsgBox)

trace("waiting for user input");

**else**

\_gameArea.advanceTime(passedTime);

}

}

**public** **class** GameArea

{

**private** **var** \_juggler:Juggler;

**public** **function** advanceTime(passedTime:Number):void

{

\_juggler.advanceTime(passedTime);

}

}

Stopping Animations

tween:Tween = **new** Tween(ball, 1.5);

tween.moveTo(x, y);

Starling.juggler.add(tween);

**var** animID:uint = juggler.add(tween);

Starling.juggler.remove(tween);

Starling.juggler.removeTweens(ball);

Starling.juggler.removeByID(animID);

Starling.juggler.purge();

Automatic Removal (IAnimatable)

**public** **class** MyAnimation **extends** EventDispatcher **implements** IAnimatable

{

**public** **function** stop():void

{

dispatchEventWith(Event.REMOVE\_FROM\_JUGGLER);

}

}

Any object that implements IAnimatable can dispatch such an event; the juggler listens to those events and will remove the object accordingly.

Single-Command Tweens

juggler.tween(msgBox, 0.5, {

transition: Transitions.EASE\_IN,

onComplete: **function**():**void** { button.enabled = **true**; },

x: 300,

rotation: deg2rad(90)

});

Delayed Calls

juggler.delayCall(gameOver, 2);

OR

juggler.delayCall(dispatchEventWith, 2, "GAME\_OVER");

OR

juggler.repeatCall(spawnEnemy, 3);

(Note: delay pause if juggler pause)

Abort a delayed call

**var** animID:uint = juggler.delayCall(gameOver, 2);

juggler.removeByID(animID);

juggler.removeDelayedCalls(gameOver);

**Movie Clip**

TextureAtlas

**<TextureAtlas** imagePath="atlas.png"**>**

**<SubTexture** name="flight\_00" x="0" y="0" width="50" height="50" **/>**

**<SubTexture** name="flight\_01" x="50" y="0" width="50" height="50" **/>**

**<SubTexture** name="flight\_02" x="100" y="0" width="50" height="50" **/>**

**<SubTexture** name="flight\_03" x="150" y="0" width="50" height="50" **/>**

*<!-- ... -->*

**</TextureAtlas>**

Creating

**var** frames:Vector.<Texture> = atlas.getTextures("flight\_");

**var** movie:MovieClip = **new** MovieClip(frames, 10);

addChild(movie);

movie.play(); // It will be in "play" mode per default.

movie.pause();

movie.stop();

Starling.juggler.add(movie);

Public Properties

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Property** | | **Defined By** |
|  |  | [**currentFrame**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#currentFrame) : int  The index of the frame that is currently displayed. | MovieClip |
|  |  | [**currentTime**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#currentTime) : Number  The time that has passed since the clip was started (each loop starts at zero). | MovieClip |
|  |  | [**fps**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#fps) : Number  The default number of frames per second. | MovieClip |
|  |  | [**isComplete**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#isComplete) : Boolean  [read-only] Indicates if a (non-looping) movie has come to its end. | MovieClip |
|  |  | [**isPlaying**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#isPlaying) : Boolean  [read-only] Indicates if the clip is still playing. | MovieClip |
|  |  | [**loop**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#loop) : Boolean  Indicates if the clip should loop. | MovieClip |
|  |  | [**muted**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#muted) : Boolean  If enabled, no new sounds will be started during playback. | MovieClip |
|  |  | [**numFrames**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#numFrames) : int  [read-only] The total number of frames. | MovieClip |
|  |  | [**soundTransform**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#soundTransform) : SoundTransform  The SoundTransform object used for playback of all frame sounds. | MovieClip |
|  |  | [**totalTime**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#totalTime) : Number  [read-only] The total duration of the clip in seconds. | MovieClip |

Public Methods

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Method** | | **Defined By** |
|  |  | [**MovieClip**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#MovieClip())(textures:Vector.<[Texture](http://doc.starling-framework.org/current/starling/textures/Texture.html)>, fps:Number = 12)  Creates a movie clip from the provided textures and with the specified default framerate. | MovieClip |
|  |  | [**addFrame**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#addFrame())(texture:[Texture](http://doc.starling-framework.org/current/starling/textures/Texture.html), sound:Sound = null, duration:Number = -1):void  Adds an additional frame, optionally with a sound and a custom duration. | MovieClip |
|  |  | [**addFrameAt**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#addFrameAt())(frameID:int, texture:[Texture](http://doc.starling-framework.org/current/starling/textures/Texture.html), sound:Sound = null, duration:Number = -1):void  Adds a frame at a certain index, optionally with a sound and a custom duration. | MovieClip |
|  |  | [**advanceTime**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#advanceTime())(passedTime:Number):void  Advance the time by a number of seconds. | MovieClip |
|  |  | [**getFrameAction**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#getFrameAction())(frameID:int):Function  Returns the method that is executed at a certain frame. | MovieClip |
|  |  | [**getFrameDuration**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#getFrameDuration())(frameID:int):Number  Returns the duration of a certain frame (in seconds). | MovieClip |
|  |  | [**getFrameSound**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#getFrameSound())(frameID:int):Sound  Returns the sound of a certain frame. | MovieClip |
|  |  | [**getFrameTexture**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#getFrameTexture())(frameID:int):[Texture](http://doc.starling-framework.org/current/starling/textures/Texture.html)  Returns the texture of a certain frame. | MovieClip |
|  |  | [**pause**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#pause())():void  Pauses playback. | MovieClip |
|  |  | [**play**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#play())():void  Starts playback. | MovieClip |
|  |  | [**removeFrameAt**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#removeFrameAt())(frameID:int):void  Removes the frame at a certain ID. | MovieClip |
|  |  | [**reverseFrames**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#reverseFrames())():void  Reverses the order of all frames, making the clip run from end to start. | MovieClip |
|  |  | [**setFrameAction**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#setFrameAction())(frameID:int, action:Function):void  Sets an action that will be executed whenever a certain frame is reached. | MovieClip |
|  |  | [**setFrameDuration**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#setFrameDuration())(frameID:int, duration:Number):void  Sets the duration of a certain frame (in seconds). | MovieClip |
|  |  | [**setFrameSound**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#setFrameSound())(frameID:int, sound:Sound):void  Sets the sound of a certain frame. | MovieClip |
|  |  | [**setFrameTexture**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#setFrameTexture())(frameID:int, texture:[Texture](http://doc.starling-framework.org/current/starling/textures/Texture.html)):void  Sets the texture of a certain frame. | MovieClip |
|  |  | [**stop**](http://doc.starling-framework.org/current/starling/display/MovieClip.html#stop())():void  Stops playback, resetting "currentFrame" to zero. | MovieClip |

More Complex Movies

* [Spine](http://esotericsoftware.com/)
* [Flump](https://github.com/threerings/flump)
* [Dragon Bones](http://dragonbones.github.io/)
* [Generic Animation Format](http://gafmedia.com/)

**Asset Management**

Enqueuing

*// Enqueue an asset from a remote URL*

assets.enqueue("http://gamua.com/img/starling.jpg");

*// Enqueue an asset from disk (AIR only)*

**var** appDir:File = File.applicationDirectory;

assets.enqueue(appDir.resolvePath("sounds/music.mp3"));

*// Enqueue all contents of a directory, recursively (AIR only).*

assets.enqueue(appDir.resolvePath("textures"));

The AssetManager contains a verbose property. If enabled, all steps of the enqueuing and loading process will be traced to the console.

Texture Atlas

assets.enqueue(appDir.resolvePath("textures/atlas.xml"));

assets.enqueue(appDir.resolvePath("textures/atlas.png"));

Bitmap Fonts

assets.enqueue(appDir.resolvePath("fonts/desyrel.fnt"));

assets.enqueue(appDir.resolvePath("fonts/desyrel.png"));

Assets that are embedded

**public** **class** EmbeddedAssets

{

*/\* PNG texture \*/*

[Embed(source = "/textures/bird.png")]

**public** **static** **const** bird:Class;

*/\* ATF texture \*/*

[Embed(source = "textures/1x/atlas.atf",

mimeType = "application/octet-stream")]

**public** **static** **const** atlas:Class;

*/\* XML file \*/*

[Embed(source = "textures/1x/atlas.xml",

mimeType = "application/octet-stream")]

**public** **static** **const** atlas\_xml:Class;

*/\* MP3 sound \*/*

[Embed(source = "/audio/explosion.mp3")]

**public** **static** **const** explosion:Class;

}

**var** assets:AssetManager = **new** AssetManager();

assets.enqueue(EmbeddedAssets);

Per-Asset Configuration

**var** assets:AssetManager = **new** AssetManager();

assets.textureFormat = Context3DTextureFormat.BGRA\_PACKED;

assets.scaleFactor = 2;

assets.enqueue(EmbeddedAssets);

assets.scaleFactor = 1;

assets.enqueue(appDir.resolvePath("textures/1x"));

assets.scaleFactor = 2;

assets.enqueue(appDir.resolvePath("textures/2x"));

Loading the Assets

assets.loadQueue(**function**(ratio:Number):**void**

{

trace("Loading assets, progress:", ratio);

*// when the ratio equals '1', we are finished.*

**if** (ratio == 1.0)

startGame();

});

Accessing the Assets

**var** texture:Texture = assets.getTexture("bird");

**var** textures:Vector.<Texture> = assets.getTextures("animation");

**var** explosion:SoundChannel = assets.playSound("explosion");

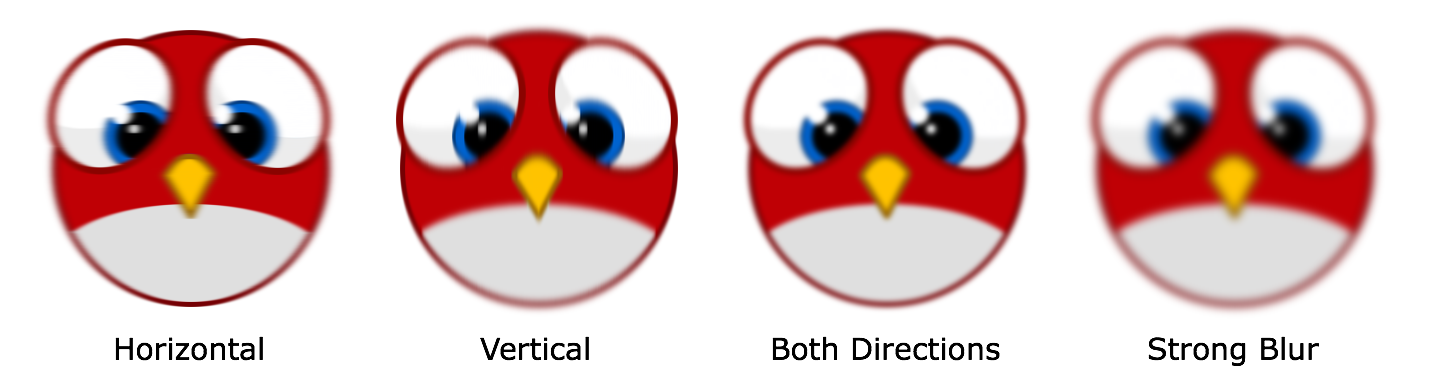
In my games, I typically store a reference to the asset manager at my root class, accessible through a static property. That makes it super easy to access my assets from anywhere in the game, simply by calling Game.assets.get…​() (assuming the root class is called Game).

**Fragment Filters**

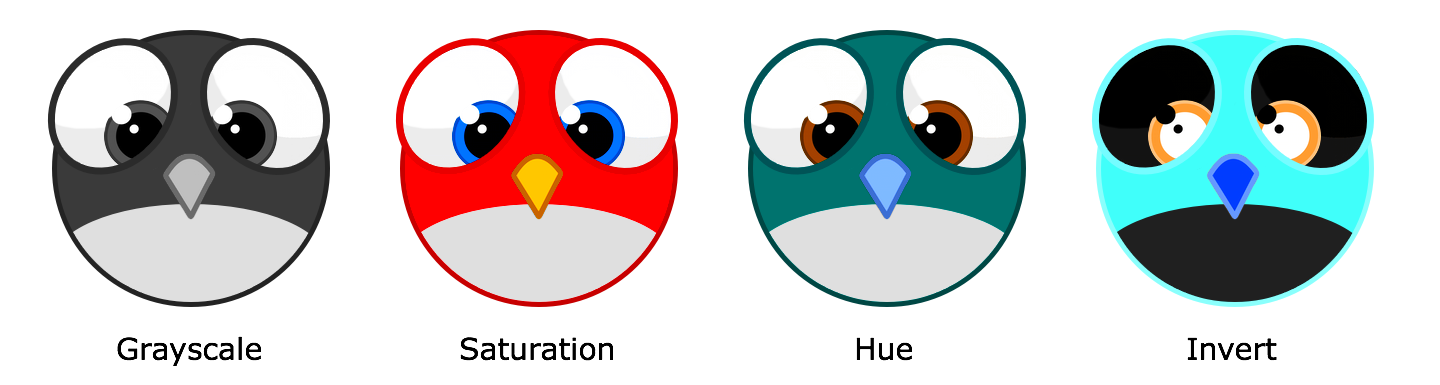
**var** filter:BlurFilter = **new** BlurFilter();

object.filter = filter;

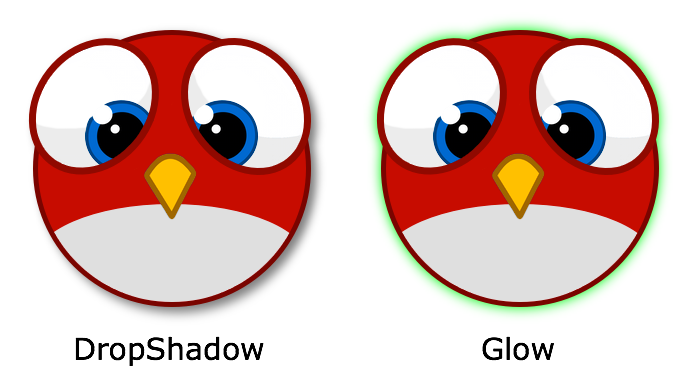
BlurFilter



ColorMatrixFilter



DropShadow- and GlowFilter DisplacementMapFilter FilterChain

Performance Tips

I mentioned it above: while the GPU processing part is very efficient, the additional draw calls make fragment filters rather expensive. However, Starling does its best to optimize filters.

* When an object does not change its position relative to the stage (or other properties like scale and color) for two successive frames, Starling recognizes this and will automatically cache the filter output. This means that the filter won’t need to be processed any more; instead, it behaves just like a single image.
* On the other hand, when the object is constantly moving, the last filter pass is always rendered directly to the back buffer instead of a texture. That spares one draw call.
* If you want to keep using the filter output even though the object is moving, call **filter.cache()**. Again, this will make the object act just like a static image. However, for any changes of the target object to show up, you must call cache again (or **uncache**).
* To save memory, experiment with the resolution and textureFormat properties. This will reduce image quality, though.

More Filters

In the meantime, you can try out filters created by other Starling developers. An excellent example is the filter collection by devon-o: <https://github.com/devon-o/Starling-Filters>