LEARNING ActionScript 3

#actionscript-3

Table of Contents

About

Chapter 1: Getting started with ActionScript 3

Remarks

ActionScript 3 is the programming language for the Adobe Flash Player and Adobe AIR runtime environments. It is object-oriented ECMAScript based language used primary for native application development on desktop (Windows/Mac) and mobile (iOS/Android) devices.

Adobe learning resources: http://www.adobe.com/devnet/actionscript/learning.html

History and more details: https://en.wikipedia.org/wiki/ActionScript

Online documentation on classes and reference:
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/package-detail.html

Versions

There is a single version of Actionscript 3, named "ActionScript 3.0"

Flash Version	Codename	Changes and Improvements	Release Date
Flash Player 9.x	Zaphod	initial release	2006-06-22
Flash Player 10.0	Astro	introduced the `Vector.<T>` type, the Adobe Pixel Bender shader filters in `flash.filters.ShaderFilter` class, and its hardware support on multiple CPUs.	2008-10-15
Flash Player 10.1	Argo	introduced `flash.events.TouchEvent` class to work with multitouch devices, and other support of mobile device hardware, such as accelerometer.	2010-06-10
Flash Player 10.2	Spicy	introduced `flash.media.StageVideo` class and the general framework to work with stage video playback in AS3.	2011-02-08
Flash Player 11	Serrano	adds H.264 support to video streaming over `NetStream` objects in both directions. Also it adds SSL/TLS support for Flash connection with `SecureSocket` class.	2011-10-04
Flash Player	Brannan	introduced `flash.system.Worker` class and the ability to delegate asynchronous work to other threads on the	2012-08-10

Flash Version	Codename	Changes and Improvements	Release Date
11.4		client.	
Flash Player 11.8	Harrison	removed hardware support (JIT compilation) for Adobe Pixel Bender shader filters, drastically reducing performance of any PB shader filter execution.	2013-05-09

Examples

Installation Overview

ActionScript 3 can be used by installing the Adobe AIR SDK or Apache Flex SDK or as part Adobe's Animate CC product *(formerly known as Flash Professional)*.

Adobe Animate CC is a professional software solution that can be used to create AS3 projects using visual tools - once installed, no further steps are necessary to begin creating AS3 projects.

The AIR SDK and Flex SDK can be used with command line tools or with various third party IDEs.

In addition to Adobe Animate CC, there are four other popular IDEs capable of working with AS3. These IDEs have their own instructions on how to get started.

- Flash Builder (By Adobe - based on Eclipse)
- IntelliJ IDEA (By Jetbrains)
- FlashDevelop
- FDT (Eclipse Plugin)

Hello World

An example document class that prints “Hello, World” to the debug console when instantiated.

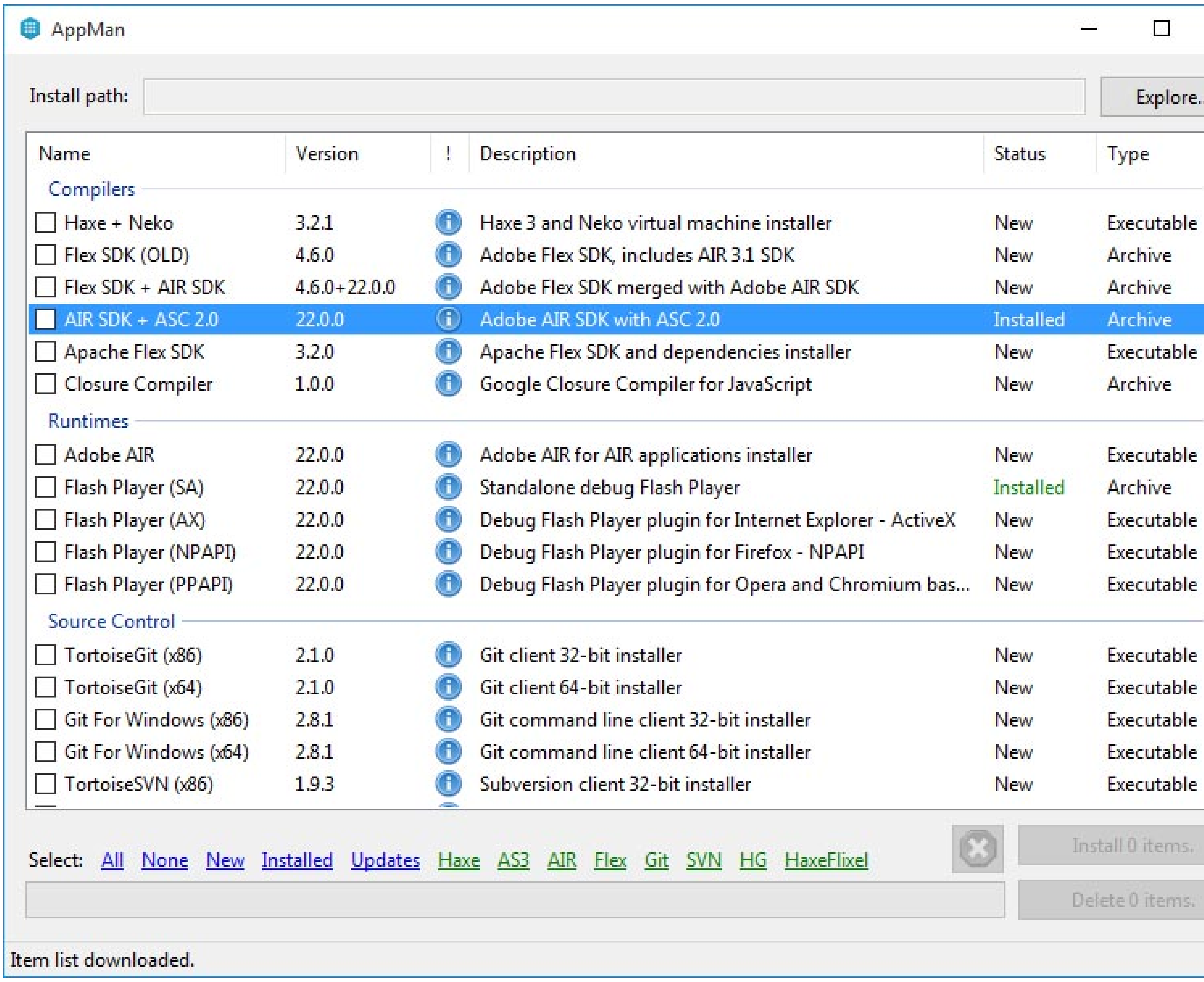
```
import flash.display.Sprite;

public class Main extends Sprite {

    public function Main() {
        super();

        trace("Hello, World");
    }

}
```

Flash Develop Installation

FlashDevelop is a multi-platform open source IDE created in 2005 for Flash developers. With no cost, it's a very popular way to get started developing with AS3.

To Install FlashDevelop:

1. Download The Installation File and run the installer
2. Once installation is complete, run FlashDevelop. On the first launch, the `App Man` window should appear asking you to pick what SDK's and tools to install.



AppMan

Install path: Explore.

Name	Version	!	Description	Status	Type
Compilers					
☐ Haxe + Neko	3.2.1	ⓘ	Haxe 3 and Neko virtual machine installer	New	Executable
☐ Flex SDK (OLD)	4.6.0	ⓘ	Adobe Flex SDK, includes AIR 3.1 SDK	New	Archive
☐ Flex SDK + AIR SDK	4.6.0+22.0.0	ⓘ	Adobe Flex SDK merged with Adobe AIR SDK	New	Archive
☐ AIR SDK + ASC 2.0	22.0.0	ⓘ	Adobe AIR SDK with ASC 2.0	Installed	Archive
☐ Apache Flex SDK	3.2.0	ⓘ	Apache Flex SDK and dependencies installer	New	Executable
☐ Closure Compiler	1.0.0	ⓘ	Google Closure Compiler for JavaScript	New	Archive
Runtimes					
☐ Adobe AIR	22.0.0	ⓘ	Adobe AIR for AIR applications installer	New	Executable
☐ Flash Player (SA)	22.0.0	ⓘ	Standalone debug Flash Player	Installed	Archive
☐ Flash Player (AX)	22.0.0	ⓘ	Debug Flash Player plugin for Internet Explorer - ActiveX	New	Executable
☐ Flash Player (NPAPI)	22.0.0	ⓘ	Debug Flash Player plugin for Firefox - NPAPI	New	Executable
☐ Flash Player (PPAPI)	22.0.0	ⓘ	Debug Flash Player plugin for Opera and Chromium bas...	New	Executable
Source Control					
☐ TortoiseGit (x86)	2.1.0	ⓘ	Git client 32-bit installer	New	Executable
☐ TortoiseGit (x64)	2.1.0	ⓘ	Git client 64-bit installer	New	Executable
☐ Git For Windows (x86)	2.8.1	ⓘ	Git command line client 32-bit installer	New	Executable
☐ Git For Windows (x64)	2.8.1	ⓘ	Git command line client 64-bit installer	New	Executable
☐ TortoiseSVN (x86)	1.9.3	ⓘ	Subversion client 32-bit installer	New	Executable

Select: All None New Installed Updates Haxe AS3 AIR Flex Git SVN HG HaxeFlixel

Install 0 items.

Delete 0 items.

Item list downloaded.

If the AppMan doesn't open automatically, or you want to add something later, open it by choosing 'Install Software' on the 'Tools' menu.

Check the **AIR SDK+ ACS 2.0** item (in the 'Compiler' section) and the **Flash Player (SA)** item in the 'Runtimes' section (plus anything else you'd like to install). Click the install button.

3. Once the SDK is installed, let's test is by creating a hello world project. Start by creating a new project (from the *Project* menu)

4. Choose **AIR AS3 Projector** from the list, and give it a name/location.

5. In the project manager panel (choose 'Project Manager' from the view menu if not already visible), expand the **src** folder, and open the `Main.as` file.

6. In the `Main.as` file, you can now create a first example program like Hello World

7. Run your project by clicking the play icon, or pressing `F5`, or `Ctrl+Enter`. The project will compile and when finished a blank window should appear (this is your application). In the FlashDevelop output window, you should see the words: **Hello World**.

You are now ready to start developing AS3 applications with FlashDevelop!

Apache Flex Installation

from http://flex.apache.org/doc-getstarted.html

1. Download the SDK installer

2. Run the SDK installer. The first question you will be asked is the installation directory.

   - on a Mac, use `/Applications/Adobe Flash Builder 4.7/sdks/4.14.0/`
   - on a PC, use `C:\Program Files(x86)\Adobe Flash Builder 4.7\sdks\4.14.0`

   You will need to create the 4.14.0 folders. Press Next. Accept SDK Licenses and Install.

IDE Specific Instructions for Apache Flex setup:

- Flash Builder
- IntelliJ IDEA
- FlashDevelop
- FDT

Building Flex or Flash projects at the command line using mxmlc

The Flex compiler (`mxmlc`) is one of the most important parts of the Flex SDK. You can edit AS3 code in any text editor you like. Create a main class file that extends from `DisplayObject`.

You can trigger builds at the command line as follows:

```
mxmlc -source-path="." -default-size [width in pixels] [height in pixels] -default-frame-rate
[fps] -o "outputPath.swf" "mainClass.as"
```

If you need to compile a Flash project (as opposed to Flex) you can add a reference to the Flash library as follows (you'll need to have the Adobe Animate IDE installed):

```
mxmlc -source-path="." -library-path+="/Applications/Adobe Animate CC 2015.2/Adobe Animate CC
2015.2.app/Contents/Common/Configuration/ActionScript 3.0/libs" -static-link-runtime-shared-
libraries=true -default-size [width in pixels] [height in pixels] -default-frame-rate [fps] -o
"outputPath.swf" "mainClass.as"
```

Or on Windows:

```
mxmlc -source-path="." -library-path+="C:\Program Files\Adobe\Adobe Animate CC
2015.2\Common\Configuration\ActionScript 3.0\libs" -static-link-runtime-shared-libraries=true
```

```
-default-size [width in pixels] [height in pixels] -default-frame-rate [fps] -o
"outputPath.swf" "mainClass.as"
```

A displayed "Hello World" example

```
package {
    import flash.text.TextField;
    import flash.display.Sprite;

    public class TextHello extends Sprite {
        public function TextHello() {
            var tf:TextField = new TextField();
            tf.text = "Hello World!"
            tf.x = 50;
            tf.y = 40;
            addChild(tf);
        }
    }
}
```

This class uses the `TextField` class to display the text.

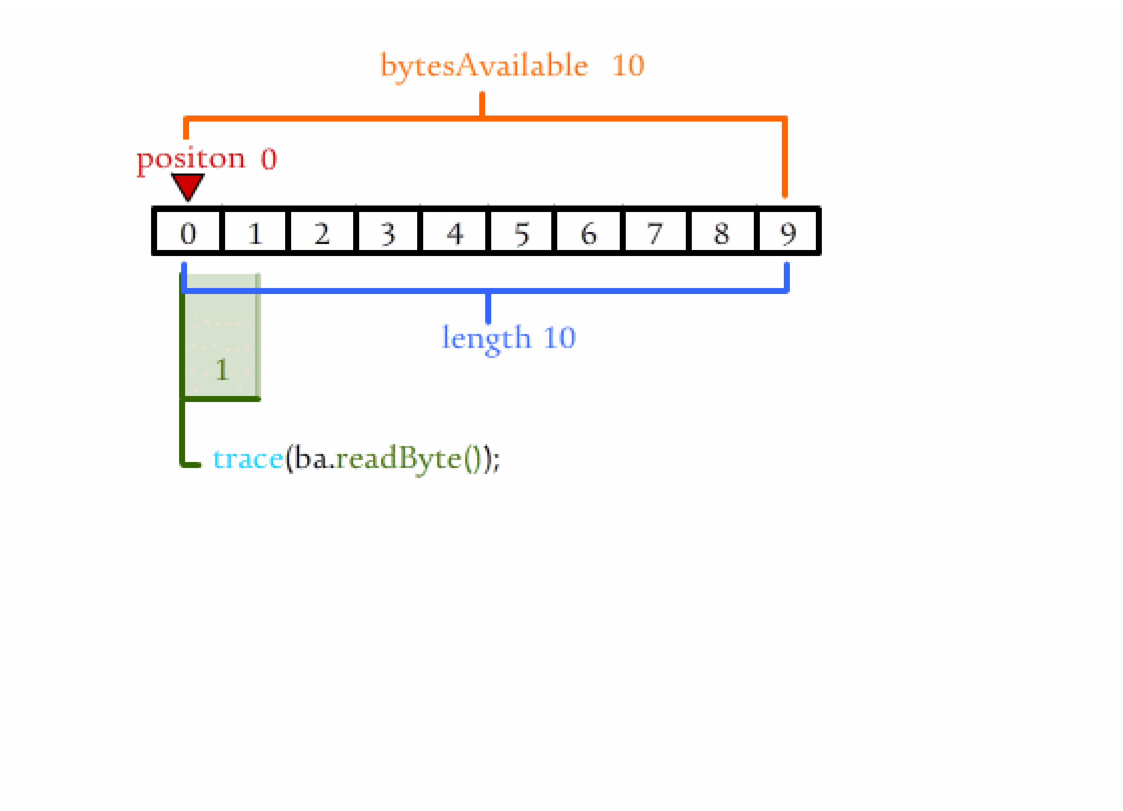
Read Getting started with ActionScript 3 online: https://riptutorial.com/actionscript-3/topic/1065/getting-started-with-actionscript-3

Chapter 2: Binary data

Examples

Reading form ByteArray throught IDataInput interface.

Animation below shows what is happening when you use IDataInput interface methods to access data form ByteArray and other classes which implement this interface.



Read Binary data online: https://riptutorial.com/actionscript-3/topic/9464/binary-data

Chapter 3: Bitmap Manipulation and Filtering

Introduction

In this topic you can learn a bit about manipulating **bitmapdata** and visual processing, working with pixels and getting started with effects filters.

Examples

Threshold (monochrome) effect

---

required:

1. understanding Bitmap and Bitmap data

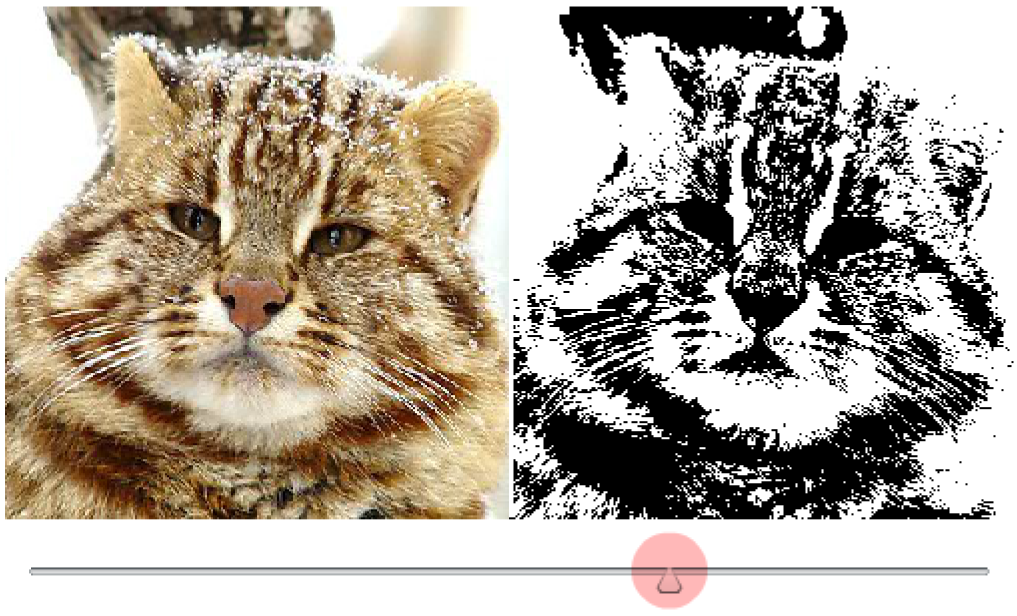
---

what is threshold

> This adjustment takes all the pixels in an image and…pushes them to either pure white or pure black

what we have to do

> here is a **Live Demo** of this example with some additional changes like using a UI to changing threshold level in runtime.



threshold in action script 3 from as3 official documentation

> Tests pixel values in an image against a specified threshold and sets pixels that pass

the test to new color values. Using the threshold() method, you can isolate and replace color ranges in an image and perform other logical operations on image pixels.

The threshold() method's test logic is as follows:

1. If ((pixelValue & mask) operation (threshold & mask)), then set the pixel to color;
2. Otherwise, if copySource == true, then set the pixel to corresponding pixel value from sourceBitmap.

i just commented the following code with exactly names as quoted description.

```
import flash.display.BitmapData;
import flash.display.Bitmap;
import flash.geom.Rectangle;
import flash.geom.Point;

var bmd:BitmapData = new wildcat(); // instantied a bitmapdata from library a wildcat
var bmp:Bitmap = new Bitmap(bmd); // our display object to previewing bitmapdata on stage
addChild(bmp);
monochrome(bmd); // invoking threshold function

/**
    @param bmd, input bitmapData that should be monochromed
*/
function monochrome(bmd:BitmapData):void {
    var bmd_copy:BitmapData = bmd.clone(); // holding a pure copy of bitmapdata for
comparation steps
    // this is our "threshold" in description above, source pixels will be compared with this
value
    var level:uint = 0xFFAAAAAA; // #AARRGGBB. in this case i used RGB(170,170,170) with an
alpha of 1. its not median but standard
    // A rectangle that defines the area of the source image to use as input.
    var rect:Rectangle = new Rectangle(0,0,bmd.width,bmd.height);
    // The point within the destination image (the current BitmapData instance) that
corresponds to the upper-left corner of the source rectangle.
    var dest:Point = new Point();
    // thresholding will be done in two section
    // the last argument is "mask", which exists in both sides of comparation
    // first, modifying pixels which passed comparation and setting them all with "color"
white (0xFFFFFFFF)
    bmd.bitmapData.threshold(bmd_copy, rect, dest, ">", level, 0xFFFFFFFF, 0xFFFFFFFF);
    // then, remaining pixels and make them all with "color" black (0xFF000000)
    bmd.bitmapData.threshold(bmd_copy, rect, dest, "<=", level, 0xFF000000, 0xFFFFFFFF);
    // Note: as we have no alpha channel in our default BitmapData (pixelValue), we left it to
its full value, a white mask (0xffffffff)
}
```

Read Bitmap Manipulation and Filtering online: https://riptutorial.com/actionscript-3/topic/8055/bitmap-manipulation-and-filtering

Chapter 4: Display List Lifecycle

Remarks

Frame-based animation in Flash and AIR implement the following lifecycle:

- `Event.ENTER_FRAME` is dispatched
- Constructor code of children display objects are executed
- `Event.FRAME_CONSTRUCTED` is dispatched
- Frame actions in the `MovieClip` symbol is executed
- Frame actions in children `MovieClip` symbols are executed
- `Event.EXIT_FRAME` is dispatched
- `Event.RENDER` is dispatched

Examples

Added and removed from stage lifecycle

```
package {
import flash.display.Sprite;
import flash.events.Event;

public class Viewport extends Sprite {

    /** Constructor */
    public function Viewport() {
        super();

        // Listen for added to stage event
        addEventListener(Event.ADDED_TO_STAGE, addedToStageHandler);
    }

    /** Added to stage handler */
    protected function addedToStageHandler(event:Event):void {
        // Remove added to stage event listener
        removeEventListener(Event.ADDED_TO_STAGE, addedToStageHandler);

        // Listen for removed from stage event
        addEventListener(Event.REMOVED_FROM_STAGE, removedFromStageHandler);
    }

    /** Removed from stage handler */
    protected function removedFromStageHandler(event:Event):void {
        // Remove removed from stage event listener
        removeEventListener(Event.REMOVED_FROM_STAGE, removedFromStageHandler);

        // Listen for added to stage event
        addEventListener(Event.ADDED_TO_STAGE, addedToStageHandler);
    }

    /** Dispose */
    public function dispose():void {
        // Remove added to stage event listener
        removeEventListener(Event.ADDED_TO_STAGE, addedToStageHandler);

        // Remove removed from stage event listener
        removeEventListener(Event.REMOVED_FROM_STAGE, removedFromStageHandler);
    }

}
}
```

Read Display List Lifecycle online: https://riptutorial.com/actionscript-3/topic/1877/display-list-lifecycle

Chapter 5: Drawing Bitmaps

Examples

Draw a display object into bitmap data

A helper function to create a bitmap copy of an object. This can be used to convert vector objects, text or complex nested Sprite's to a flattened bitmap.

```
function makeBitmapCopy(displayObj:IBitmapDrawable, transparent:Boolean = false, bgColor:uint
= 0x00000000, smooth:Boolean = true):Bitmap {

    //create an empty bitmap data that matches the width and height of the object you wish to
draw
    var bmd:BitmapData = new BitmapData(displayObj.width, displayObj.height, transparent,
bgColor);

    //draw the object to the bitmap data
    bmd.draw(displayObj, null, null, null, null, smooth);

    //assign that bitmap data to a bitmap object
    var bmp:Bitmap = new Bitmap(bmd, "auto", smooth);

    return bmp;
}
```

Usage:

```
var txt:TextField = new TextField();
txt.text = "Hello There";

var bitmap:Bitmap = makeBitmapCopy(txt, true); //second param true to keep transparency
addChild(bitmap);
```

Draw a display object with any coordinates of registration point

```
    public function drawDisplayObjectUsingBounds(source:DisplayObject):BitmapData {
        var bitmapData:BitmapData;//declare a BitmapData
        var bounds:Rectangle = source.getBounds(source);//get the source object actual size
        //round bounds to integer pixel values (to aviod 1px stripes left off)
        bounds = new Rectangle(Math.floor(bounds.x), Math.floor(bounds.y),
Math.ceil(bounds.width), Math.ceil(bounds.height));

        //to avoid Invalid BitmapData error which occures if width or height is 0
        //(ArgumentError: Error #2015)
        if((bounds.width>0) && (bounds.height>0)){
            //create a BitmapData
            bitmapData = new BitmapData(bounds.width, bounds.height, true, 0x00000000);

            var matrix:Matrix = new Matrix();//create a transform matrix
            //translate if to fit the upper-left corner of the source
            matrix.translate(-bounds.x, -bounds.y);
            bitmapData.draw(source, matrix);//draw the source
            return bitmapData;//return the result (exit point)
        }
        //if no result is created - return an empty BitmapData
        return new BitmapData(1, 1, true, 0x00000000);
    }
```

A side note: For `getBounds()` to return valid values the object has to have stage access at least once, otherwise the values are bogus. A code can be added to ensure that passed `source` has stage, and if not, it can be added to stage then removed again.

Animating a sprite sheet

A sprite sheet by definition is a bitmap that contains a certain animation. Old games use grid type sprite sheet, that is, every frame occupies an equal region, and frames are aligned by the edges to form a rectangle, probably with some spaces unoccupied. Later, in order to minimize the bitmap size, sprite sheets start to be "packed" by removing extra whitespace around the rectangle that contains each frame, but still each frame is a rectangle to simplify copying operations.

In order to animate a sprite sheet, two techniques can be used. First, you can use `BitmapData.copyPixels()` to copy a certain region of your sprite sheet to a displayed `Bitmap`, producing an animated character. This approach is better if you use a single displayed `Bitmap` that hosts the entire picture.

```
var spriteSheet:BitmapData;
var frames:Vector.<Rectangle>; // regions of spriteSheet that represent frames
function displayFrameAt(frame:int,buffer:BitmapData,position:Point):void {
    buffer.copyPixels(spriteSheet,frames[frame],position,null,null,true);
}
```

The second technique can be used if you have a lot of `Sprite`s or `Bitmap`s on the display list, and they share the same sprite sheet. Here, the target buffer is no longer a single object, but each object has its own buffer, so the proper strategy is to manipulate the bitmaps' `bitmapData` property. Prior to doing this, however, the sprite sheet should be cut apart into individual frames.

```
public class Stuff extends Bitmap {
    static var spriteSheet:Vector.<BitmapData>;
    function displayFrame(frame:int) {
        this.bitmapData=spriteSheet[frame];
    }
    // ...
}
```

Read Drawing Bitmaps online: https://riptutorial.com/actionscript-3/topic/2814/drawing-bitmaps

Chapter 6: Game Development Basics

Introduction

[![enter image description here][1]][1] **basics** of game development. ------------------------------ *Note*, this set of tutorials/articles contains many concepts which may provided as separated topics before. we have to refreshing them in the mind and learning a bit of implementing most critical parts of a video-game via actionscript-3. [1]: https://i.stack.imgur.com/CUIsz.png

Examples

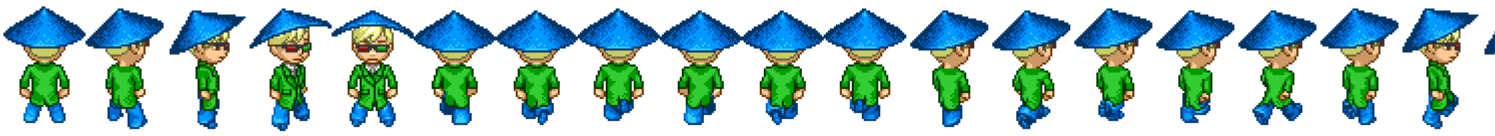
isometric character animating + movement

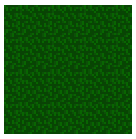
① the concepts used in this Example :

Class	Usage
`URLRequest` + `Loader` + `Event`	Loading atlas map (sprite) from external path.
`BitmapData` + `Sprite` + `beginBitmapFill` + `Matrix` + `stageWidth & stageHeight`	drawing loaded resources to bitmapdata, using tiled bitmaps, drawing with transformation.
`MovieClip` + `scrollRect` + `Bitmap` + `Rectangle`	creating and cliping character movieclip using Bitmap as timeline.
`KeyboardEvent`	detecting user inputs
`Event.EXIT_FRAME`	implementing game Loop function

---

② Resources : **(no permission for using this resources for a commercial purposes)**





③ Code and Comments :

> **Note:** FPS 15 Used for this tutorial, its recommended, but if need more, you must modify some part of code by your self.

at first we must download our resources from external urls.

```
const src_grass_tile_url:String = "https://i.stack.imgur.com/sjJFS.png";
const src_character_atlas_url:String = "https://i.stack.imgur.com/B7ztZ.png";

var loader:Loader = new Loader();
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setGround);
loader.load(new URLRequest(src_grass_tile_url));
```

---

setGround will be caled once `src_grass_tile_url` is loaded and ready for use. in follow implementing setGround to get resource and draw it as the game background

```
function setGround(e:Event):void {
    /* drawing ground */
    /* loader is a displayObject, so we can simply draw it into the bitmap data*/
    /* create an instance of Bitmapdata with same width and height as our window*/
    /* (also set transparent to false because grass image, does not contains any transparent
pixel) */
    var grass_bmd:BitmapData = new BitmapData(loader.width, loader.height, false, 0x0);
    /* time to draw */
    grass_bmd.draw(loader); // drawing loader into the bitmapData
    /* now we have to draw a tiled version of grass_bmd inside a displayObject Sprite to
displaying
       BitmapData on stage */
    var grass_sprite:Sprite = new Sprite();
    // for drawing a bitmap inside sprite, we must use <beginBitmapFill> with graphic property
of the sprite
    // then draw a full size rectangle with that Fill-Data
    // there is a repeat mode argument with true default value so we dont set it true again.
    // use a matrix for scalling grass Image during draw to be more cute!
    var mx:Matrix = new Matrix();
    mx.scale(2, 2);
    grass_sprite.graphics.beginBitmapFill(grass_bmd, mx);
    grass_sprite.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
    // now add sprite to displayobjectcontainer to be displayed
    stage.addChild(grass_sprite);

    // well done, ground is ready, now we must initialize our character
    // first, load its data, i just re-use my loader for loading new image, but with another
complete handler (setCharacter)
    // so remove existing handler, then add new one
    loader.contentLoaderInfo.removeEventListener(Event.COMPLETE, setGround);
    loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setCharacter);
    loader.load(new URLRequest(src_character_atlas_url));
}
```

the code is well damn commented, after we have done with ground, its time to implementing character. character also contains a resource which must be loaded with same way. so at the end of `setGround` we are heading to the `setCharacter` which is another complete call back.

```
function setCharacter(e:Event):void {
    // let assuming that what is really character!
    // a set of images inside a single image!
    // that images are frames of our character (also provides movement for different
directions)
    // first load this
    var character_bmd:BitmapData = new BitmapData(loader.width, loader.height, true, 0x0); //
note character is transparent
    character_bmd.draw(loader);
    // take a look at sprite sheet, how many frames you see?
    // 42 frames, so we can get width of a single frame
    const frame_width:uint = character_bmd.width / 42; // 41 pixels
    // as i show you above, to displaying a BitmapData, we have to draw it using a
DisplayObject,
    // another way is creating a Bitmap and setting its bitmapdata
    var character_bmp:Bitmap = new Bitmap(character_bmd);
    // but its not enough yet, a movieClip is necessary to cliping and animating this bitmap
(as a child of itself)
    var character_mc:MovieClip = new MovieClip();
    character_mc.addChild(character_bmp);
    character_bmp.name = "sprite_sheet"; // setting a name to character_bmp, for future
accessing
    character_mc.scrollRect = new Rectangle(0, 0, frame_width, character_bmd.height); //
cliping movieclip, to dusplaying only one frame
    character_mc.name = "character"; // setting a name to character_mc, for future accessing
    stage.addChild(character_mc); // adding it to stage.
    // well done, we have a character, but its static yet! 2 steps remaining. 1 controlling 2
animating
    // at first setting a control handler for moving character in 8 directions.
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyDown);
    stage.addEventListener(KeyboardEvent.KEY_UP, keyUp);
}
```

now character is ready to controll. its displayed well and ready for controlling. so keyboard events attached and listening for arrow keys as following code :

```
// we storing key stats inside <keys> Object
var keys:Object = {u:false, d:false, l:false, r:false};
function keyDown(e:KeyboardEvent):void {
    switch (e.keyCode) {
        case 38: //up
            keys.u = true;
            break;
        case 40: //down
            keys.d = true;
            break;
        case 37: //left
            keys.l = true;
            break;
        case 39: //right
            keys.r = true;
            break;
    }
}
function keyUp(e:KeyboardEvent):void {
    switch (e.keyCode) {
        case 38: //up
            keys.u = false;
            break;
        case 40: //down
            keys.d = false;
            break;
        case 37: //left
```

```
            keys.l = false;
            break;
        case 39: //right
            keys.r = false;
            break;
    }
}
```

`keys:Object` stores 4's boolean variable per each arrow key, moving proccess must be done inside the update (Loop) function of the game, so we must pass keyboard stats to it. lets implementing **Loop** function.

```
// initialize game Loop function for updating game objects
addEventListener(Event.EXIT_FRAME, loop);

// speed of character movement
const speed:Number = 5;
// this function will be called on each frame, with same rate as your project fps
function loop(e:Event):void {
    if (keys.u) stage.getChildByName("character").y -= speed;
    else if (keys.d) stage.getChildByName("character").y += speed;
    if (keys.l) stage.getChildByName("character").x -= speed;
    else if (keys.r) stage.getChildByName("character").x += speed;
}
```

speed is a helper constant, defines velocity of character. above code presents a simple 8 direction movement with this priority low: `Up > Down Left > Right`. so if up and down arrow are pressed in same time, the character only moves to *up* (not freezing).

well Done!!! only one step remaining, animation, the most important part of this tutorial

what is really animation? a set of keyframes which contains atleast one frame
lets creating our keyframs Object, which contains name of keyframes
and also some data about starting and ending frame of this keyframe
Note, in isometric games, each keyframe contains 8 direction (can be reduced to 5 with use of flipping)

```
var keyframs:Object = {
    idle: {up:[0,0], up_right:[1,1], right:[2,2], down_right:[3,3], down:[4,4]}, // [2,2]
means start frame is 2 and end frame is 2
    run: {up:[5,10], up_right:[11,16], right:[17,22], down_right:[23,28], down:[29,34]}
};
```

we should ignore remaining frames, this example only provides idle and run animation
for example the start frame of idle animation with direction right, is: <keyframs.idle.right[0]>
now lets implementing Animator function

```
var current_frame:uint;
function animate(keyframe:Array):void {
    // how it works
    // just called with a keyframe with direction (each frame),
    // if keyframe is what is already playing, its just moved to next frame and got updated
(or begning frame for loop)
    // other wise, just moved to begining frame of new keyframe
    if (current_frame >= keyframe[0] && current_frame <= keyframe[1]) { // check if in bound
        current_frame++;
        if (current_frame > keyframe[1]) // play back if reached
            current_frame = keyframe[0];
    } else {
        current_frame = keyframe[0]; // start new keyframe from begining
    }
    // moving Bitmap inside character MovieClip
    var character:MovieClip = stage.getChildByName("character") as MovieClip;
    var sprite_sheet:Bitmap = character.getChildByName("sprite_sheet") as Bitmap;
    sprite_sheet.x = -1 * current_frame * character.width;
}
```

read comments of above function, however main job of this function is moving *sprite_sheet* `Bitmap` inside *character* `MovieClip`.

we know that every update should be done inside the Loop function, so we will invoke this function from Loop with related keyframes. this is the updated Loop function :

```
// speed of character movement
const speed:Number = 8;
var last_keyStat:Object = {u:false, d:false, l:false, r:false}; // used to getting a backup of
previous keyboard stat for detecting correct idle direction
// this function will be called on each frame, with same rate as your project fps
function loop(e:Event):void {
    if (keys.u) stage.getChildByName("character").y -= speed;
    else if (keys.d) stage.getChildByName("character").y += speed;
    if (keys.l) stage.getChildByName("character").x -= speed;
    else if (keys.r) stage.getChildByName("character").x += speed;

    // animation detection
    if (keys.u && keys.l) { animate(keyframs.run.up_right); flip(true); }
    else if (keys.u && keys.r) { animate(keyframs.run.up_right); flip(false); }
    else if (keys.d && keys.l) { animate(keyframs.run.down_right); flip(true); }
    else if (keys.d && keys.r) { animate(keyframs.run.down_right); flip(false); }
    else if (keys.u) { animate(keyframs.run.up); flip(false); }
    else if (keys.d) { animate(keyframs.run.down); flip(false); }
    else if (keys.l) { animate(keyframs.run.right); flip(true); }
    else if (keys.r) { animate(keyframs.run.right); flip(false); }
    else {
        // if character dont move, so play idle animation
        // what is the best practice to detecting idle direction?
        // my suggestion is to sotring previous keyboard stats to determining which idle
direction is correct
        // do any better thing if possible (absolutely is possible)
        // i just simply copy it from above, and replaced (keys) with (last_keyStat) and (run)
with (idle)
        if (last_keyStat.u && last_keyStat.l) { animate(keyframs.idle.up_right); flip(true); }
        else if (last_keyStat.u && last_keyStat.r) { animate(keyframs.idle.up_right);
flip(false); }
        else if (last_keyStat.d && last_keyStat.l) { animate(keyframs.idle.down_right);
flip(true); }
        else if (last_keyStat.d && last_keyStat.r) { animate(keyframs.idle.down_right);
flip(false); }
        else if (last_keyStat.u) { animate(keyframs.idle.up); flip(false); }
        else if (last_keyStat.d) { animate(keyframs.idle.down); flip(false); }
        else if (last_keyStat.l) { animate(keyframs.idle.right); flip(true); }
        else if (last_keyStat.r) { animate(keyframs.idle.right); flip(false); }
    }
    // update last_keyStat backup
    last_keyStat.u = keys.u;
    last_keyStat.d = keys.d;
    last_keyStat.l = keys.l;
    last_keyStat.r = keys.r;
}
```

read comments, we just simply detecting a true keyframe through keyboard stats. then also do same thing for detecting idle animation. for idle animations we have no key input to using for detecting which direction character is on, so a simle helper variable could be handy to storing previous state of keyboard (last_keyStat).

---

also there is a new function `flip` which is another helper function used for simulating missing animations (left + up_left + down_left) also this funcion do some fixes which is commented below:

```
// usage of flip function is because of Movieclip registration point, its a fix
// as the registration point of MovieClip is not placed in center, when flipping animation
(for non existing directions inside spritesheet)
// character location changes with an unwanted value equal its width, so we have to prevent
this and push it back or forward during flip
function flip(left:Boolean):void {
    var character:MovieClip = stage.getChildByName("character") as MovieClip;
    if (left) {
        if (character.scaleX != -1) {
            character.scaleX = -1;
            character.x += character.width; // comment this line to see what happen without
this fix
        }
    } else {
        if (character.scaleX != 1) {
            character.scaleX = 1;
            character.x -= character.width; // comment this line to see what happen without
this fix
        }
    }
}
```

our work is ending here. special thanks for Editor's which making this tutorial more undrestandable. also Here is a Live demo of this tutorial plus an external link of complete code.

④ External References:

- **full code**
- **Live Demo**

Read Game Development Basics online: https://riptutorial.com/actionscript-3/topic/8237/game-development-basics

Chapter 7: Loading External Files

Remarks

There are some cases where your application cannot manipulate the contents of assets loaded from an external resource (e.g. transform images). I can't remember for certain what they are, but I am fairly sure it's related to cross domain content loading.

Examples

Loading External Images/SWFs With The Loader

1. Create a Loader object:

```
var loader:Loader = new Loader();   //import
```

2. Add listeners on the loader. Standard ones are complete and io/security errors

```
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete); //when the
loader is done loading, call this function
loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, loadIOError); //if the
file isn't found
loader.contentLoaderInfo.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
loadSecurityError); //if the file isn't allowed to be loaded
```

3. Load the desired file:

```
 loader.load(new URLRequest("image.png"));
```

4. Create Your Event Handlers:

```
 function loadComplete(e:Event):void {
     //load complete
     //the loader is actually a display object itself, so you can just add it to the
display list
     addChild(loader)
     //or addChild(loader.content) to add the root content of what was loaded;
 }

 function loadIOError(e:IOErrorEvent):void {
     //the file failed to load,
 }

 function loadSecurityError(e:SecurityError):void {
     //the file wasn't allowed to load
 }
```

Loading with the Loader class is asynchronous. This means after you call `loader.load` the application will continue running while the file loads. Your loader content isn't available until the loader dispatches the `Event.COMPLETE` event.

---

imports needed:

```
import flash.display.Loader;
import flash.events.Event;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLRequest;
```

Loading a text file with FileStream (AIR runtime only)

A simple example on how to read a UTF text file synchronously.

```
import flash.filesystem.File;
import flash.filesystem.FileMode;
import flash.filesystem.FileStream;
```

```
//First, get a reference to the file you want to load
var myFile:File = File.documentsDirectory.resolvePath("lifestory.txt");

//Create a FileStream object
fileStream = new FileStream();

//open the file
fileStream.open(myFile, FileMode.READ);

//read the data and assign it to a local variable
var fileText:String = fileStream.readUTF();

//close the current filestream
fileStream.close();
```

Read Loading External Files online: https://riptutorial.com/actionscript-3/topic/1694/loading-external-files

Chapter 8: Object Oriented Programming

Examples

"Overloaded" Constructor via static method

Constructor overloading is not available in As3.

In order to provide a different way to retrieve an instance of a class, a `public static` method can be provided to serve as an alternative "constructor".

An example for that is `flash.geom.Point`, which represents a 2D point object. The coordinates to define the point can be

- **cartesian** in the regular constructor

  ```
  public function Point(x:Number = 0, y:Number = 0)
  ```

  example usage:

  ```
  var point:Point = new Point(2, -.5);
  ```

- **polar** in a static method

  ```
  public static function polar(len:Number, angle:Number):Point
  ```

  example usage:

  ```
  var point:Point = Point.polar(12, .7 * Math.PI);
  ```

  Because it is not an actual constructor, there's no `new` keyword.

set & get functions

To ensure encapsulation, member variables of a class should be `private` and only be accessible to `public` via public `get`/`set` access methods. It is a common practice to prefix private fields with _

```
public class Person
{
    private var _name:String = "";

    public function get name():String{
        return _name;
        //or return some other value depending on the inner logic of the class
    }


    public function set name(value:String):void{
```

```
        //here you may check if the new value is valid
        //or maybe dispatch some update events or whatever else
        _name = value;
    }
```

Sometimes you don't even need to create a `private` field for a `get`/`set` pair.
For example in a control like a custom radio group you need to know which radio button is selected, however outside the class you need just a way to `get`/`set` only the selected value:

```
public function get selectedValue():String {
    //just the data from the element
    return _selected ? _selected.data : null;
}
public function set selectedValue(value:String):void {
    //find the element with that data
    for (var i:int = 0; i < _elems.length; i++) {
        if (_elems[i].data == value) {
            _selected = _elems[i];//set it
            processRadio();//redraw
            return;
        }
    }
}
```

Packages

Packages are bundles of classes. Every class must be declared within a package using the `package` statement. The `package` statement is followed by the name of your package, or followed by nothing in the case of adding classes to the top-level package. Sub-packages are created using dot (`.`) delimitation. The package statement is followed by a block which will contain *a single `class` definition*. Examples:

```
package {
    // The top level package.
}

package world {
    // A package named world.
}

package world.monsters {
    // A package named monsters within a package named world.
}
```

Packages should correlate to the file structure of the classes relative to the source root. Assuming you have a source root folder named `src`, the above could be correctly represented in the filesystem as:

```
src
    TopLevelClass.as

    world
        ClassInWorldPackage.as
        AnotherClassInWorldPackage.as
        monsters
            Zombie.as
```

Method overriding

When you `extend` a class, you can `override` methods that the inherited class defines using the `override` keyword:

```
public class Example {
    public function test():void {
        trace('It works!');
    }
}

public class AnotherExample extends Example {
    public override function test():void {
        trace('It still works!');
    }
}
```

Example:

```
var example:Example = new Example();
var another:AnotherExample = new AnotherExample();

example.test(); // Output: It works!
another.test(); // Output: It still works!
```

You can use the `super` keyword to reference the original method from the class being inherited. For example, we could change the body of `AnotherExample.test()` to:

```
public override function test():void {
    super.test();
    trace('Extra content.');
}
```

Resulting in:

```
another.test(); // Output: It works!
                //         Extra content.
```

Overriding class constructors is a little bit different. The `override` keyword is omitted and accessing the inherited constructor is done simply with `super()`:

```
public class AnotherClass extends Example {
    public function AnotherClass() {
        super(); // Call the constructor in the inherited class.
    }
}
```

You can also override `get` and `set` methods.

getter and setter

Getters and setters are methods that are behaved like properties. it means they have function structure but when used, they are used same as properties:

Structure of getter functions:

they should have `get` keyword after `function` keyword and before function name, with no argument, a return type specified and must return a value:

```
public function get myValue():Type{
    //anything here
    return _desiredValue;
}
```

Syntax:

to get the value from a getter,the syntax is the same as getting a value from a property(no parens `()` are used).

```
trace(myValue);
```

Structure of setter functions:

they should have `set` keyword after `function` keyword and before function name, with one argument, and no value return.

```
public function set myValue(value:Type):void{
    //anything here
    _desiredProperty=value;
}
```

Syntax:

to set the value of a setter,the syntax is the same as setting a value to a property(using equal sign `=` then value).

```
myValue=desiredValue;
```

setting a getter and setter for one value:

> Note: if you create only getter or only setter with a name, that property would be read-only or set-only.

to make a property both readable and setable, should create a getter and a setter with:

1.the same name.
2.the same type(type of return value for the getter and type of input value(argument) for the setter,

> Note: getters and setters should not have a name same as other properties or methods

.

Usage of getters and setters:

Using getters and setters rather than normal properties has many pros:

1.**making read-only or set-only properties:**
for example number of children in a display object. it can't be setable.

2.**accessing private properties:**
an example:

```
private var _private:Type=new Type();
//note that function name "private" is not same as variable name "_private"
public function get private():Type{
    return _private;
}
```

3.**when some change is required after setting a value:**
in this example, changing this property must be notified:

```
public static function set val:(input:Type):void{
    _desiredProperty=input;
    notifyValueChanged();
}
```

and many other usages

Read Object Oriented Programming online: https://riptutorial.com/actionscript-3/topic/2042/object-oriented-programming

Chapter 9: Optimizing Performance

Examples

Vector based graphics

Vector based graphics are represented by a plethora of data that must be computed by the CPU (vector points, arcs, colors, etc). Anything other than simple shapes with minimal points and straight lines will consume vast amounts of CPU resource.

There is a "Cache as Bitmap" flag that can be turned on. This flag stores the result of drawing the vector based DisplayObject for much faster redraws. The pitfall of this is that if there are any transformations applied to the object, the entire thing needs to be redrawn and re-cached. This can be slower than not turning it on at all if there are frame-by-frame transformations applied (rotation, scaling, etc).

Generally, rendering graphics using bitmaps is far more performant than using vector graphics. Libraries such as flixel take advantage of this for rendering sprites on a "canvas" without reducing framerate.

Text

Rendering text consumes a lot of CPU. Fonts are rendered in a fashion similar to vector graphics and contain many vector points for every character. Altering text frame-by-frame *will* degrade performance. The "Cache as bitmap" flag is extremely useful if used correctly, meaning you must avoid:

- Altering the text frequently.
- Transforming the text field (rotating, scaling).

Simple techniques like wrapping text updates in an `if` statement will make a major difference:

```
if (currentScore !== oldScore) {
    field.text = currentScore;
}
```

Text can be rendered using the anti-aliased renderer built into Flash, or using "device fonts". Using "device fonts" makes text render much faster, although it makes text appear jagged (aliased). Also, device fonts requires the font to be pre-installed by your end user, or the text may "disappear" on the user's PC although it appears fine on yours.

```
field.embedFonts = false; // uses "device fonts"
```

Vector and for each vs arrays and for

Using the `Vector.<T>` type and the `for each` loop is more performant than a conventional array and `for` loop:

Good:

```
var list:Vector.<Sprite> = new <Sprite>[];

for each(var sprite:Sprite in list) {
    sprite.x += 1;
}
```

Bad:

```
var list:Array = [];

for (var i:int = 0; i < list.length; i++) {
    var sprite:Sprite = list[i];

    sprite.x += 1;
}
```

Fast array item removal

If you do not require an array to be in any particular order, a little trick with `pop()` will afford you enormous performance gains compared to `splice()`.

When you `splice()` an array, the index of subsequent elements in that array needs to be reduced by 1. This process can consume a large chunk of time if the array is large and the object you are removing is nearer the beginning of that array.

If you do not care about the order of the elements in the array, you can instead replace the item you want to remove with an item that you `pop()` from the end of the array. This way, the indexes of all the other items in the array remains the same and the process does not degrade in performance as the length of your array grows.

Example:

```
function slowRemove(list:Array, item:*):void {
    var index:int = list.indexOf(item);

    if (index >= 0) list.splice(index, 1);
}

function fastRemove(list:Array, item:*):void {
    var index:int = list.indexOf(item);

    if (index >= 0) {
        if (index === list.length - 1) list.pop();

        else {
            // Replace item to delete with last item.
            list[index] = list.pop();
        }
    }
}
```

Vectors instead of Arrays

Flash Player 10 introduced the Vector.<*> generic list type that was faster than the Array. However, this is not entirely true. Only the following Vector types are faster than the Array counterparts, due to the way they are implemented in Flash Player.

- `Vector.<int>` - Vector of 32-bit integers
- `Vector.<uint>` - Vector of 32-bit unsigned integers
- `Vector.<Double>` - Vector of 64-bit floats

In all other cases, using an Array will be more performant than using Vectors, for all operations (creation, manipulation, etc). However, if you wish to "strongly type" your code then you can use Vectors despite the slowdown. FlashDevelop has a syntax that enables code completion drop-downs to work even for Arrays, by using `/*ObjectType*/Array`.

```
var wheels:Vector.<Wheel> // strongly typed, but slow

var wheels:/*Wheel*/Array // weakly typed, but faster
```

Reusing and pooling graphics

Creating and configuring `Sprite` and `TextField` objects at runtime can be costly if you are creating hundreds of thousands of these on a single frame. Therefore a common trick is "pooling" these objects for later reuse. Remember we are not just trying to optimize the creation time (`new Sprite()`) but also the configuration (setting of default properties).

Lets say we were building a list component using hundreds of TextField objects. When you need to create a new object, check if an existing object can be reused.

```
var pool:Array = [];

if (pool.length > 0){

    // reuse an existing TextField
    var label = pool.pop();

}else{
    // create a new TextField
    label = new TextField();

    // initialize your TextField over here
    label.setDefaultTextFormat(...);
    label.multiline = false;
    label.selectable = false;
}

// add the TextField into the holder so it appears on-screen
// you will need to layout it and set its "text" and other stuff seperately
holder.addChild(label);
```

Later, when you are destroying your component (or removing it from screen), remember to add unused labels back into the pool.

```
foreach (var label in allLabels){
    label.parent.removeChild(label); // remove from parent Sprite
    pool.push(label); // add to pool
}
```

In most cases it is best to create a pool per usage instead of a global pool. Disadvantages to a creating a global pool is you need to re-initialize the object everytime to retrieve it from the pool, to negate the settings done by other functions. This is equally costly and pretty much negates the performance boost of using pooling in the first place.

Read Optimizing Performance online: https://riptutorial.com/actionscript-3/topic/2215/optimizing-performance

Chapter 10: Random Value Generation

Examples

Random number between 0 and 1

```
Math.random();
```

produces an evenly distributed random number between 0 (inclusive) and 1 (exclusive)

Example output:

- 0.22282187035307288
- 0.3948539895936847
- 0.9987191134132445

Random number between min and max values

```
function randomMinMax(min:Number, max:Number):Number {
    return (min + (Math.random() * Math.abs(max - min)));
}
```

This function is called by passing a range of minimum and maximum values.

Example:

```
randomMinMax(1, 10);
```

Example outputs:

- 1.661770915146917
- 2.5521070677787066
- 9.436270965728909

Random angle, in degrees

```
function randomAngle():Number {
    return (Math.random() * 360);
}
```

Example outputs:

- 31.554428357630968
- 230.4078639484942
- 312.7964010089636

Random value from an array

Assuming we have an array `myArray`:

```
var value:* = myArray[int(Math.random() * myArray.length)];
```

Note we use `int` to cast the result of `Math.random()` to an int because values like `2.4539543` would not be a valid array index.

Random point inside a circle

First define the circle radius and its center:

```
var radius:Number = 100;
var center:Point = new Point(35, 70);
```

Then generate a random angle in *radians* from the center:

```
var angle:Number = Math.random() * Math.PI * 2;
```

Then generate an effective radius of the returned point, so it'll be inside given `radius`. A simple `Math.random()*radius` won't do, because with this distribution the produces points will end up in the inner circle of half radius half of the time, but the square of that circle is a quarter of original. To create a proper distribution, the function should be like this:

```
var rad:Number=(Math.random()+Math.random())*radius; // yes, two separate calls to random
if (rad>radius) { rad=2*radius-rad; }
```

This function produces a value that has its probability function linearly increasing from 0 at zero to maximum at `radius`. It happens because a sum of random values has a probability density function equal to convolution of all the random values' individual density functions. This is some extended maths for an average grade person, but a kind GIF is presented to draw a graph of convolution function of two uniformed distribution density functions explained as "box signals". The `if` operator folds the resultant function over its maximum, leaving only a sawtooth-shaped graph.

This function is selected because the square of a circle strip located between `radius=r` and `radius=r+dr` increases linearly with increasing `r` and very small constant `dr` so that `dr*dr<<r`. Therefore, the amount of points generated close at the center is smaller than the amount of points generated at the edge of the circle by the same margin as the radius of center area is smaller than the radius of the whole circle. So overall, points are evenly distributed across the entire circle.

Now, get your random position:

```
var result:Point = new Point(
    center.x + Math.cos(angle) * rad,
    center.y + Math.sin(angle) * rad
);
```

To get a random point ON the circle (on the edge of the circle of a given radius), use `radius` instead of `rad`.

PS: The example ended up being overloaded by explanation of maths.

Random angle, in radians

```
function randomAngleRadians():Number
{
    return Math.random() * Math.PI * 2;
}
```

Example outputs:

- 5.490068569213088
- 3.1984284719180205
- 4.581117863808207

Determining the success of a "percent chance" operation

If you need to roll for a `true` or `false` in an "x% chance" situation, use:

```
function roll(chance:Number):Boolean {
    return Math.random() >= chance;
}
```

Used like:

```
var success:Boolean = roll(0.5); // True 50% of the time.
var again:Boolean = roll(0.25); // True 25% of the time.
```

Create a random color

To get *any* random color:

```
function randomColor():uint
{
    return Math.random() * 0xFFFFFF;
}
```

If you need more control over the red, green and blue channels:

```
var r:uint = Math.random() * 0xFF;
var g:uint = Math.random() * 0xFF;
var b:uint = Math.random() * 0xFF;

var color:uint = r << 16 | g << 8 | b;
```

Here you can specify your own range for `r`, `g` and `b` (this example is from 0-255).

Randomly loop through alphabet

```
var alphabet:Vector.<String> = new <String>[ "A", "B", "C", "D", "E", "F", "G",
                                             "H", "I", "J", "K", "L", "M", "N",
                                             "O", "P", "Q", "R", "S", "T", "U",
                                             "V", "W", "X", "Y", "Z" ];

while (alphabet.length > 0)
{
    var letter:String = alphabet.splice(int(Math.random() *
                                            alphabet.length), 1)[0];
    trace(letter);
}
```

Example output:

> V, M, F, E, D, U, S, L, X, K, Q, H, A, I, W, N, P, Y, J, C, T, O, R, G, B, Z

Randomize An Array

```
var alphabet:Array = [ "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N",
"O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z" ];

for (var i:int=alphabet.length-1;i>0;i--) {
    var j:int=Math.floor(Math.random()*(i+1));
    var swap=alphabet[j];
    alphabet[j]=alphabet[i];
    alphabet[i]=swap;
}
trace(alphabet);
```

Example output

> B,Z,D,R,U,N,O,M,I,L,C,J,P,H,W,S,Q,E,K,T,F,V,X,Y,G,A

This method is known as Fisher-Yates array shuffle.

Read Random Value Generation online: https://riptutorial.com/actionscript-3/topic/1441/random-value-generation

Chapter 11: Responsive Application Design

Examples

Basic Responsive Application

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;

    public class Main extends Sprite
    {
        //Document Class Main Constructor
        public function Main()
        {
            //Sometimes stage isn't available yet, so if not, wait for it before proceeding
            if (!stage) {
                addEventListener(Event.ADDED_TO_STAGE, stageReady);
            }else {
                stageReady();
            }
        }

        protected function stageReady(e:Event = null):void {
            //align the stage to the top left corner of the window/container
            stage.align = StageAlign.TOP_LEFT;
            //don't scale the content when the window resizes
            stage.scaleMode = StageScaleMode.NO_SCALE;

            //listen for when the window is resized
            stage.addEventListener(Event.RESIZE, stageResized);
        }

        protected function stageResized(e:Event):void {
            //use stage.stageWdith & stage.stageHeight to repostion and resize items
        }
    }
}
```

Performing lengthy processes and not get unresponsive application

There are situations when you need to calculate something really large in your Flash application, while not interrupting the user's experience. For this, you need to devise your lengthy process as a multi-step process with saved state between iterations. For example, you need to perform a background update of a lot of internal objects, but if you desire to update them all at once with a simple `for each (var o in objects) { o.update(); }`, Flash briefly (or not as briefly) becomes unresponsive to the user. So, you need to perform one or several updates per frame.

```
private var processing:Boolean;      // are we in the middle of processing
private var lastIndex:int;           // where did we finish last time
```

```
var objects:Vector.<UpdatingObject>; // the total list of objects to update
function startProcess():Boolean {
    if (processing) return false; // already processing - please wait
    startProcessing=true;
    lastIndex=0;
    if (!hasEventListener(Event.ENTER_FRAME,iterate))
        addEventListener(Event.ENTER_FRAME,iterate); // enable iterating via listener
}
private function iterate(e:Event):void {
    if (!processing) return; // not processing - skip listener
    objects[lastIndex].update(); // perform a quantum of the big process
    lastIndex++; // advance in the big process
    if (lastIndex==objects.length) {
        processing=false; // finished, clear flag
    }
}
```

Advanced processing can include using `getTimer()` to check elapsed time and allowing another iteration if time didn't run out, splitting `update()` into several functions if updating is too long, displaying progress elsewhere, adjusting iteration process to adapt to changes of the list of objects to process, and many more.

Read Responsive Application Design online: https://riptutorial.com/actionscript-3/topic/1615/responsive-application-design

Chapter 12: Sending and Receiving Data From Servers

Examples

Making a request from Flash

The `URLRequest` and `URLLoader` classes work together to make requests from Flash to external resources. The `URLRequest` defines information about the request e.g. the request body and the request method type, and the `URLLoader` references this to perform the actual request and provide a means of being notified when a response is received from the resource.

Example:

```
var request:URLRequest = new URLRequest('http://stackoverflow.com');
var loader:URLLoader = new URLLoader();

loader.addEventListener(Event.COMPLETE, responseReceived);
loader.load(request);

function responseReceived(event:Event):void {
    trace(event.target.data); // or loader.data if you have reference to it in
                              // this scope.
}
```

Adding variables to your request

The `URLVariables` class allows you to define data to be sent along with a `URLRequest`.

Example:

```
var variables:URLVariables = new URLVariables();

variables.prop = "hello";
variables.anotherProp = 10;

var request:URLRequest = new URLRequest('http://someservice.com');
request.data = variables;
```

You can either send the request via a `URLLoader` or open the request URL with the variables attached in the querystring using `navigateToURL`.

Altering the HTTP method (GET, POST, PUT, etc)

The `URLRequestMethod` class contains constants for the various request types you can make. These constants are to be allocated to `URLRequest`'s `method` property:

```
var request:URLRequest = new URLRequest('http://someservice.com');
```

```
request.method = URLRequestMethod.POST;
```

> Note that only `GET` and `POST` are available outside the AIR runtime.

My response data is always null, what does "asynchronous" mean?

When Flash makes a request for data from an external source, that operation is *asynchronous*. The most basic explanation of what this means is that the data loads "in the background" and triggers the event handler you allocate to `Event.COMPLETE` when it is received. This can happen at any point in the lifetime of your application.

Your data **WILL NOT** be available immediately after calling `load()` on your `URLLoader`. You **must** attach an event listener for `Event.COMPLETE` and interact with the response there.

```
var request:URLRequest = new URLRequest('http://someservice.com');
var loader:URLLoader = new URLLoader();

loader.addEventListener(Event.COMPLETE, responseReceived);
loader.load(request);

trace(loader.data); // Will be null.

function responseReceived(event:Event):void {
    trace(loader.data); // Will be populated with the server response.
}

trace(loader.data); // Will still be null.
```

You cannot get around this with any little tricks like using `setTimeout` or similar:

```
setTimeout(function() {
    trace(loader.data); // Will be null if the data hasn't finished loading
                        // after 1000ms (which you can't guarantee).
}, 1000);
```

Cross-domain requests

Flash will not load data from a domain other than the one your application is running on unless that domain has an XML crossdomain policy either in the root of the domain (e.g. `http://somedomain.com/crossdomain.xml`) or somewhere that you can target with `Security.loadPolicyFile()`. The crossdomain.xml file is where you can specify domains that are able to ask your server for data from a Flash application.

Example of the *most permissive* crossdomain.xml:

```
<?xml version="1.0" ?>
<cross-domain-policy>
  <allow-access-from domain="*"/>
  <allow-http-request-headers-from domain="*" headers="*"/>
</cross-domain-policy>
```

Note this example **should not be used in production environments**, use a more restrictive instance.

A more restrictive specific crossdomain.xml will look like this for example:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-
policy.dtd">
<cross-domain-policy>
    <site-control permitted-cross-domain-policies="master-only" />

    <allow-access-from domain="*.domain.com" to-ports="80,843,8011" />
    <allow-access-from domain="123.123.123.123" to-ports="80,843,8011" />
</cross-domain-policy>
```

Resources:

- The crossdomain policy file specification.

Read Sending and Receiving Data From Servers online: https://riptutorial.com/actionscript-3/topic/1893/sending-and-receiving-data-from-servers

Chapter 13: Singleton Pattern

Remarks

The singleton pattern has the goal to allow only one instance of a class to exists at any given time.

Preventing the direct instantiation via constructor is usually prevent by making it private. However, this is not possible in As3 and thus other ways to control the number of instances have to be used.

Examples

Singleton enforcer via private instance

In this approach, the single is accessed via the static method:

```
Singleton.getInstance();
```

To enforce only one instance of the singleton, a private static variable retains the instance, while any additional attempts to instantiate an instance are enforced within the constructor.

```
package {

public class Singleton {

    /** Singleton instance */
    private static var _instance: Singleton = new Singleton();

    /** Return singleton instance. */
    public static function getInstance():Singleton {
        return _instance;
    }

    /** Constructor as singleton enforcer. */
    public function Singleton() {
        if (_instance)
            throw new Error("Singleton is a singleton and can only be accessed through
Singleton.getInstance()");
    }

}
}
```

Read Singleton Pattern online: https://riptutorial.com/actionscript-3/topic/1437/singleton-pattern

Chapter 14: Types

Examples

Type Casting

Type casting is done with either the `as` operator:

```
var chair:Chair = furniture as Chair;
```

Or by wrapping the value in `Type()`:

```
var chair:Chair = Chair(furniture);
```

If the cast fails with `as`, the result of that cast is `null`. If the cast fails by wrapping in `Type()`, a `TypeError` is thrown.

The Function type

Functions are of the type `Function`:

```
function example():void { }
trace(example is Function); // true
```

They can be referenced by other variables with the type `Function`:

```
var ref:Function = example;
ref(); // ref.call(), ref.apply(), etc.
```

And they can be passed in as arguments for parameters whose type is `Function`:

```
function test(callback:Function):void {
    callback();
}

test(function() {
    trace('It works!');
}); // Output: It works!
```

The Class type

References to class declarations are typed `Class`:

```
var spriteClass:Class = Sprite;
```

You can use variables typed `Class` to instantiate instances of that class:

```
var sprite:Sprite = new spriteClass();
```

This can be useful for passing an argument of type `Class` to a function that might create and instance of the provided class:

```
function create(type:Class, x:int, y:int):* {
    var thing:* = new type();

    thing.x = x;
    thing.y = y;

    return thing;
}

var sprite:Sprite = create(Sprite, 100, 100);
```

Annotating types

You can tell the compiler the type of a value by annotating it with `:Type`:

```
var value:int = 10; // A property "value" of type "int".
```

Function parameters and return types can also be annotated:

```
// This function accepts two ints and returns an int.
function sum(a:int, b:int):int {
    return a + b;
}
```

Attempting to assign a value with a mismatching type will result in a `TypeError`:

```
var sprite:Sprite = 10; // 10 is not a Sprite.
```

Checking types

You can use the `is` operator to validate whether a value is of a certain type:

```
var sprite:Sprite = new Sprite();

trace(sprite is Sprite); // true
trace(sprite is DisplayObject); // true, Sprite inherits DisplayObject
trace(sprite is IBitmapDrawable); // true, DisplayObject implements IBitmapDrawable
trace(sprite is Number); // false
trace(sprite is Bitmap); // false, Bitmap inherits DisplayObject
                         // but is not inherited by Sprite.
```

There is also an `instanceof` operator (deprecated) which works almost identical to `is` except that it *returns `false` when checking for implemented interfaces* and int/uint types.

The `as` operator can also by used just like `is` operator. This is especially usefull if you use some smart IDE like FlashDevelop which will give you a list of all possible properties of explicit object type. Example:

```
for (var i:int = 0; i < a.length; i++){
    var d:DisplayObject = a[i] as DisplayObject;
    if (!d) continue;
    d.//get hints here
    stage.addChild(d);
}
```

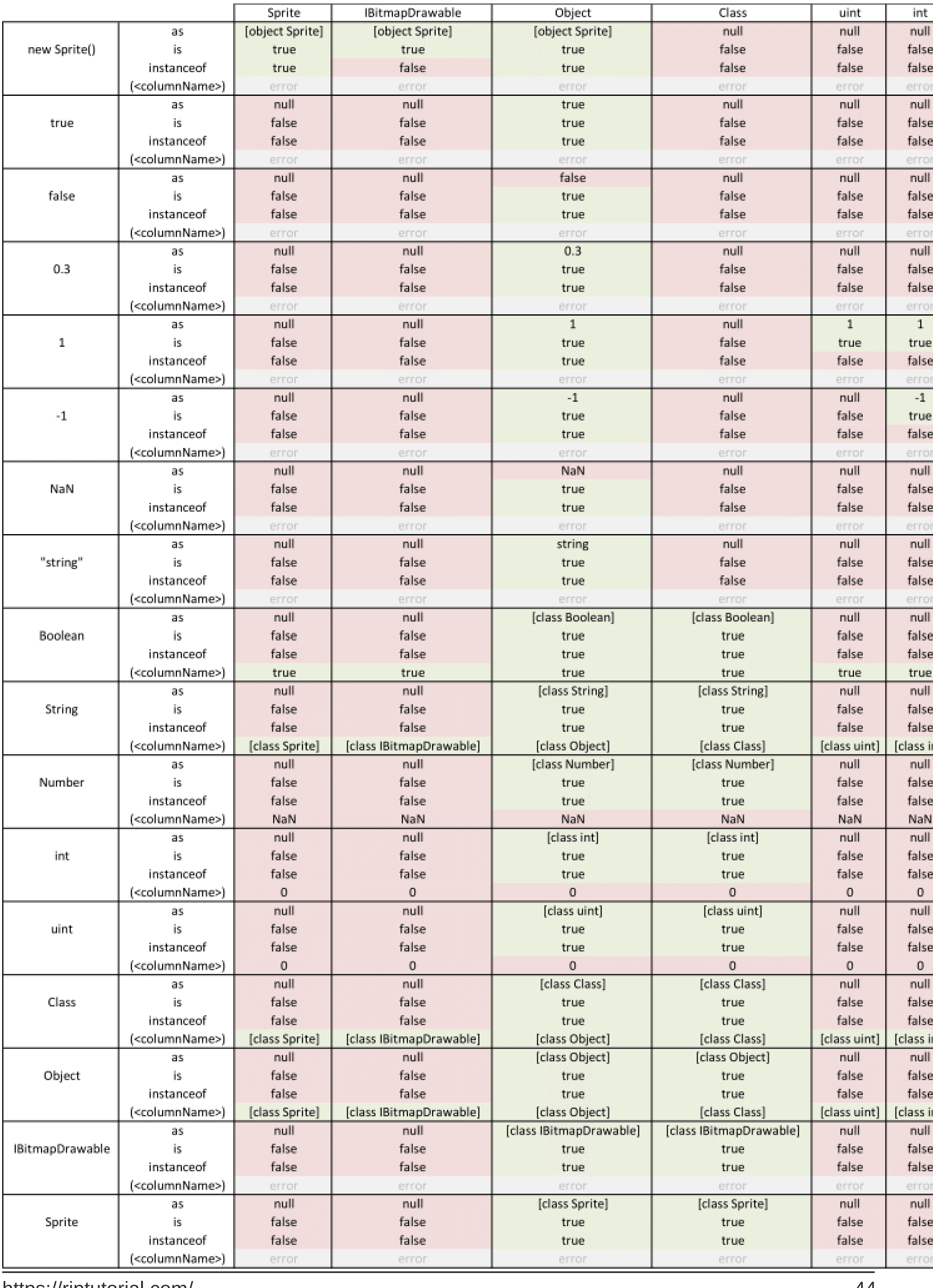
To get the same effect with `is` you would write (sligthly less convenient):

```
for (var i:int = 0; i < a.length; i++){
    if (a[i] is DisplayObject != true) continue;
    var d:DisplayObject = a[i] as DisplayObject;
    stage.addChild(d);
}
```

Just keep in mind that when checking coditions with `as` operator, given value will be fist converted to specified type and then result of that operation will be checked if not false, so be careful when using it with possible false/NaN values:

```
if(false as Boolean) trace("This will not be executed");
if(false as Boolean != null) trace("But this will be");
```

Below table shows some basic values and types with result of type operators. Green cells will evaluate to true, red to false and greay will cause compile/runtime errors.



		Sprite	IBitmapDrawable	Object	Class	uint	int
new Sprite()	as	[object Sprite]	[object Sprite]	[object Sprite]	null	null	null
	is	true	true	true	false	false	false
	instanceof	true	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
true	as	null	null	true	null	null	null
	is	false	false	true	false	false	false
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
false	as	null	null	false	null	null	null
	is	false	false	true	false	false	false
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
0.3	as	null	null	0.3	null	null	null
	is	false	false	true	false	false	false
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
1	as	null	null	1	null	1	1
	is	false	false	true	false	true	true
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
-1	as	null	null	-1	null	null	-1
	is	false	false	true	false	false	true
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
NaN	as	null	null	NaN	null	null	null
	is	false	false	true	false	false	false
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
"string"	as	null	null	string	null	null	null
	is	false	false	true	false	false	false
	instanceof	false	false	true	false	false	false
	(<columnName>)	error	error	error	error	error	erro
Boolean	as	null	null	[class Boolean]	[class Boolean]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	true	true	true	true	true	true
String	as	null	null	[class String]	[class String]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	[class Sprite]	[class IBitmapDrawable]	[class Object]	[class Class]	[class uint]	[class i
Number	as	null	null	[class Number]	[class Number]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	NaN	NaN	NaN	NaN	NaN	NaN
int	as	null	null	[class int]	[class int]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	0	0	0	0	0	0
uint	as	null	null	[class uint]	[class uint]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	0	0	0	0	0	0
Class	as	null	null	[class Class]	[class Class]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	[class Sprite]	[class IBitmapDrawable]	[class Object]	[class Class]	[class uint]	[class i
Object	as	null	null	[class Object]	[class Object]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	[class Sprite]	[class IBitmapDrawable]	[class Object]	[class Class]	[class uint]	[class i
IBitmapDrawable	as	null	null	[class IBitmapDrawable]	[class IBitmapDrawable]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	error	error	error	error	error	erro
Sprite	as	null	null	[class Sprite]	[class Sprite]	null	null
	is	false	false	true	true	false	false
	instanceof	false	false	true	true	false	false
	(<columnName>)	error	error	error	error	error	erro

Typed arrays

- You receive compile-type `TypeError`s if you attempt to insert non-T values into the collection.
- IDEs provide useful type hinting information for the objects inside a `Vector.<T>` instance.

Examples of creating a `Vector.<T>`:

```
var strings:Vector.<String> = new Vector.<String>(); // or
var numbers:Vector.<Number> = new <Number>[];
```

---

[1] Vectors actually only provide notable performance improvements over arrays when working with primitive types (`String`, `int`, `uint`, `Number`, etc).

Read Types online: https://riptutorial.com/actionscript-3/topic/2803/types

Chapter 15: Understanding the "Error 1009: Cannot access a property or method of a null object reference"

Introduction

An error 1009 is a general error that arises when you are trying to receive a value out of a variable or property that has a value of `null`. The examples provided expose various cases where this error arises, together with some recommendations on how to mitigate the error.

Remarks

The dreaded and often asked "Error 1009: Cannot access a property or method of a null object reference" is a signal that some of the data appears null, but is tried to be used as a populated object. There are pretty many types of issues that can cause this behavior, and each one should be tested against the code where the error arised.

Examples

Stage is unavailable

Sometimes developers write some code that desires access to `stage`, or Flash stage, to add listeners. It can work for the first time, then all of a sudden fail to work and produce the error 1009. The code in question can even be on the timeline, as it's the first initiative to add code there, and many tutorials that still exist use timeline code layer to place the code.

```
public class Main extends MovieClip {
    public function Main() {
        stage.addEventListener(Event.ENTER_FRAME,update); // here
```

The reason this code doesn't work is simple: A display object is first instantiated, then added to the display list, and while it's off the display list, `stage` is null.

Worse if the code like this:

```
stage.addEventListener(Event.ENTER_FRAME,update); // here
```

is placed on the timeline. It can even work for some time, while the `Main` object is slapped to stage via GUI. Then, their SWF is loaded from another SWF, and all of a sudden the code breaks. This happens because the `Main`'s frames are constructed in a different way when the SWF is loaded directly by the player and when the loading is processed asynchronously. The solution is to use `Event.ADDED_TO_STAGE` listener, and put all code that addresses stage in it, and put the listener itself into an AS file instead of the timeline.

Invalid typecast

```
function listener(e:Event):void {
    var m:MovieClip=e.target as MovieClip;
    m.x++;
}
```

If such a listener is attached to an object that's not a `MovieClip` descendant (for example, a `Sprite`), the typecast will fail, and any subsequent operations with its result will throw the 1009 error.

Uninstantiated object

```
var a:Object;
trace(a); // null
trace(a.b); // Error 1009
```

Here, an object reference is declared, but is never assigned a value, be it with `new` or assignment of a non-null value. Requesting its properties or method results in a 1009 error.

Multi-tiered expression

```
x=anObject.aProperty.anotherProperty.getSomething().data;
```

Here, any object before the dot can end up being null, and using methods that return complex objects only increases the complication to debug the null error. Worst case if the method is prone to extraneous failures, say retrieving data over the network.

Unprocessed function result

```
s=this.getChildByName("garbage");
if (s.parent==this) {...}
```

`getChildByName()` is one of the many functions that can return null if an error occurred when processing its input. Therefore, if you are receiving an object from any function that can possibly return null, check for null first. Here, a property is instantly queried without first checking if `s` is null, this will generate the 1009 error.

Forgotten event listener

```
addEventListener(Event.ENTER_FRAME,moveChild);
function moveChild(e:Event):void {
    childMC.x++;
    if (childMC.x>1000) {
        gotoAndStop(2);
    }
}
```

This example will move the `childMC` (added to `Main` at design time) but will instantly throw a 1009 as soon as `gotoAndStop()` is invoked, if that `childMC` does not exist on frame 2. The primary reason for this is that whenever a playhead passes a key frame (a frame which doesn't inherit the previous frame's object set), either by using `gotoAndStop()`, `gotoAndPlay()` with destination frame being separated from the current frame by a keyframe, or by normal play if the SWF is an animation, the current frame's contents are **destroyed** and the new contents are created using the data stored from GUI. So, if the new frame does not have a child named `childMC`, the property request will return null and 1009 will be thrown.

The same principle applies if you add two event listeners, but remove only one, or add a listener to one object, but try removing from another. The `removeEventListener` call won't warn you if the object did not have a respective event listener attached, so read the code that adds and removes event listeners carefully.

Note also: Using `Timer` objects, calling `setInterval()` and `setTimeout()` also creates event listeners, and these should also be cleared properly.

Invalidated reference to a frame-based object

Sometimes `gotoAndStop()` is called in the middle of the code that refers some frame-based properties. But, **right after the frame is changed** all links to properties that existed on the current frame are invalidated, so any processing that involves them should be immediately terminated.

There are two general scenarios of such processing to occur: First, a loop doesn't end after `gotoAndStop()` call, like here:

```
for each (bullet in bullets) {
    if (player.hitTestObject(bullet)) gotoAndStop("gameOver");
}
```

Here, a 1009 error means that the `player` MC was destroyed while processing `gotoAndStop()` call, but the loop continues, and refers the now-null link to get `hitTestObject()` from. If the condition would say `if (bullet.hitTestObject(player))` instead, the error would be #2007 "Parameter hitTestObject must not be null". The solution is to put a `return` statement right after calling `gotoAndStop()`.

Second case is multiple event listeners on the same event. Like this:

```
stage.addEventListener(Event.ENTER_FRAME,func1);
stage.addEventListener(Event.ENTER_FRAME,func2);
function func1(e:Event):void {
    if (condition()) {
        gotoAndStop(2);
    }
}
```

Here, if `condition()` is true, the first listener would perform `gotoAndStop()`, but the second listener would still be executed, and if that one references objects on the frame, a 1009 error will be thrown. The solution is to avoid multiple listeners on a single event, in a single object, it's better to have one listener that handles all situations on that event and can properly terminate if a frame change is needed.

Read Understanding the "Error 1009: Cannot access a property or method of a null object reference" online: https://riptutorial.com/actionscript-3/topic/2098/understanding-the--error-1009--cannot-access-a-property-or-method-of-a-null-object-reference-

Chapter 16: Using the Proxy Class

Introduction

First, I have to say. There **is** a reason this Proxy thing, despite its seeming usefulness, is not highlighted enough on the Internet.

You **cannot** use it to watch a random property of a random class/instance. You are only allowed to use this technique by subclassing the **Proxy** class.

Some operations do not expect exceptions so you need to completely understand what are you doing and why are you doing it, and your code **must** be absolutely clean and error-less.

Examples

Implementation

The other thing about Proxy class, and why it is not so popular, is that it is rather difficult to fathom a problem that exactly needs a dynamic class with a controllable access to its dynamic properties and methods as a most appropriate solution. Every time I tried to use Proxy, I have ended up resorting to something else, simpler and more controllable.

However, let us not be discouraged. I like the idea of addressing the last array elements by [-1], [-2], etc indices in **Python**. Might be not a big feat, but it feels nice to use that rather than long and clumsy **someArray[someArray.length - 1]**. Lets see what we can do about it.

```
package
{
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;

    /**
     * Pyaray the Tentacled Whisperer of Impossible Secrets.
     */

    dynamic public class PyArray extends Proxy
    {
        private var data:Array;

        public function PyArray(...args:Array)
        {
            if (args.length == 0)
            {
                data = new Array;
            }
            else if ((args.length == 1) && (args[0] is Array))
            {
                data = args[0];
            }
            else
            {
```

```
            data = args;
        }
    }

    // This is a getter proxy to all the available Array
    // elements and properties and, sometimes, methods.
    override flash_proxy function getProperty(name:*):*
    {
        var anIndex:int = name;

        // Handle the int indices of the Array elements.
        if (anIndex == name)
        {
            // Handle the -1, -2, etc indexing.
            if (anIndex < 0) anIndex += data.length;

            if (anIndex >= data.length) return null;
            if (anIndex < 0) return null;

            return data[anIndex];
        }

        // Handle the existing public Array properties.
        if (data.hasOwnProperty(name)) return data[name];

        // Handle the Array methods addressed via ["member"] access.
        try
        {
            if (data[name] is Function) return data[name];
            else throw new Error;
        }
        catch (fail:Error)
        {
            trace("[PyArray] is unable to resolve property \"" + name + "\".");
        }

        return null;
    }

    // This will set either elements, or settable properties.
    override flash_proxy function setProperty(name:*, value:*):void
    {
        var anIndex:int = name;

        // Handle the int indices of the Array elements.
        if (anIndex == name)
        {
            // Handle the -1, -2, etc indexing.
            if (anIndex < 0) anIndex += data.length;

            // In case the element index is out of range,
            // the PyArray will extend its data Array.
            // if (anIndex >= data.length) return;
            if (anIndex < 0) return;

            data[anIndex] = value;

            return;
        }

        // Handle the existing (or dynamic) public Array properties.
```

```
            try
            {
                data[name] = value;
            }
            catch (fail:Error)
            {
                trace("[PyArray] is unable to set property \"" + name + "\".");
            }

            return;
        }

        // This allows to delete PyArray elements with "delete" operator.
        override flash_proxy function deleteProperty(name:*):Boolean
        {
            var anIndex:int = name;

            // Handle the int indices of the Array elements.
            if (anIndex == name)
            {
                // Handle the -1, -2, etc indexing.
                if (anIndex < 0) anIndex += data.length;

                if (anIndex >= data.length) return false;
                if (anIndex < 0) return false;

                data.splice(anIndex, 1);

                return true;
            }

            // Handle the dynamic public Array properties.
            try
            {
                delete data[name];
                return true;
            }
            catch (fail:Error)
            {
                trace("[PyArray] is unable to delete the \"" + name + "\" property.");
            }

            return false;
        }

        // This proxies any attempt to call a method on PyArray directly to data Array, thus
        // all Array methods (including "toString" method called through trace) are available.
        override flash_proxy function callProperty(name:*, ...rest):*
        {
            try
            {
                return (data[name] as Function).apply(data, rest);
            }
            catch (fail:Error)
            {
                trace("[PyArray] is unable to resolve method \"" + name + "\".");
                return null;
            }
        }

        // This allows PyArray to handle for..in and for..each..in loops.
        // The initial call starts with zero, so we need to do this +1 -1 magic
        // in order for enumeration to work correctly. I'm not happy with this either.
        override flash_proxy function nextNameIndex(index:int):int
        {
            if (index >= data.length) return 0;
            else return index + 1;
        }

        // This method handles the for..in loop.
        override flash_proxy function nextName(index:int):String
        {
            return (index - 1).toString();
        }

        // This method handles the for..each..in loop.
        override flash_proxy function nextValue(index:int):*
        {
            return data[index - 1];
        }
    }
}
```

Usage

```
package
{
    import flash.display.Sprite;

    /**
     * Daemonette of Slaanesh.
     *
     * It is minor female demon, vaguely human-like, but with crab-like pincers instead of
hands.
     * She wears a rather indecent skimpy leather bikini, moves quickly and casts deadly
spells!
     */

    public class Slaanesh extends Sprite
    {
        public function Slaanesh()
        {
            // Lets initialize the PyArray.
            var PA:PyArray = new PyArray(1,2,3,4,5,4,3,2,1,"Foo");

            // Basic check: get the last element.
            trace(PA[-1]);
            // output:
            // Foo

            // This will map to the 0-based third element.
            trace(PA[2.0]);
            // output:
            // 3

            // This should not get us anywhere.
            trace(PA[2.1]);
            // output:
            // [PyArray] is unable to resolve property "2.1".
            // null
            // This should return the length of the data Array.
            trace(PA["length"]);
            // output:
            // 10

            // This should return the length of the data Array.
            // This will not compile unless PyArray class is marked "dynamic".
            trace(PA.length);
            // output:
            // 10

            // This will map to indexOf method of data Array via getProperty method.
            trace(PA["indexOf"]);
            // output:
            // function Function() {}

            // This will map to indexOf method of data Array via getProperty method.
            // This will not compile unless PyArray class is marked "dynamic".
            trace(PA.indexOf);
            // output:
            // function Function() {}

            // This is a try to access a non-existent property.
            // This will not compile unless PyArray class is marked "dynamic".
            trace(PA.P124);
            // output:
            // [PyArray] is unable to resolve property "P124".
            // null

            // This is a try to call a non-existent method via callProperty method.
            // This will not compile unless PyArray class is marked "dynamic".
            trace(PA.P124());
            // output:
            // [PyArray] is unable to resolve method "P124".
            // null

            // Basic check: calling a proxied method via callProperty method.
            trace(PA.indexOf(5));
            // output:
            // 4

            // An attempt to replace an Array method with a random value.
            // This will not compile unless PyArray class is marked "dynamic".
            PA.indexOf = 123;
            // output:
            // [PyArray] is unable to set property "indexOf".

            // An attempt to assign a random value to a random property.
            // It will succees because Array, as a dynamic class, allows so.
            // This will not compile unless PyArray class is marked "dynamic".
            PA.indexOfz = 123;
            trace(PA.indexOfz);
            // output:
            // 123

            // An attempt to assign an Array element via negative indexing.
            // This trace works fine because toString method is also proxied.
            PA[-3] = "Hello";
            trace(PA);
            // output:
```

```
            // 1,2,3,4,5,4,3,Hello,1,Foo

            // An attempt to delete Array elements via normal and negative indexing.
            // This operation is mapped via deleteProperty method.
            delete PA[-4]; // deletes "3" before "Hello"
            delete PA[0];  // deletes "1" at the start.
            trace(PA);
            // output:
            // 2,3,4,5,4,Hello,1,Foo

            // An attempt to delete a non-dynamic method reference.
            // There's no error output, AS3 must be handling this internally.
            delete PA.indexOf;
            trace(PA.indexOf);
            // output:
            // function Function() {}

            // An attempt to set an element out of index range.
            PA[10] = "123abc";
            trace(PA);
            // output:
            // 2,3,4,5,4,Hello,1,Foo,,,123abc

            var aText:String;

            // This is a test of for..in loop, Array elements are
            // enumerated via nextName and nextNameIndex methods.
            aText = ""

            for (var aKey:String in PA)
                aText += aKey + ":" + PA[aKey] + " ";

            trace(aText);
            // output:
            // 0:2 1:3 2:4 3:5 4:4 5:Hello 6:1 7:Foo 8:undefined 9:undefined 10:123abc

            // This is a test of for..each..in loop, Array elements are
            // enumerated via nextValue and nextNameIndex methods.
            aText = "";

            for each (var aValue:* in PA)
                aText += aValue + " ";

            trace(aText);
            // output:
            // 2 3 4 5 4 Hello 1 Foo undefined undefined 123abc
        }
    }
}
```

Read Using the Proxy Class online: https://riptutorial.com/actionscript-3/topic/10631/using-the-proxy-class

Chapter 17: Working with Date and Time

Examples

Ante meridiem (AM) or Post meridiem (PM) for 12-hour clock

```
function meridiem(d:Date):String {
    return (d.hours > 11) ? "pm" : "am";
}
```

Number of days in the specified month

```
/**
 * @param year   Full year as int (ex: 2000).
 * @param month  Month as int, zero-based (ex: 0=January, 11=December).
 */
function daysInMonth(year:int, month:int):int {
    return (new Date(year, ++month, 0)).date;
}
```

Whether the specified year is leap year

```
function isLeapYear(year:int):Boolean {
    return daysInMonth(year, 1) == 29;
}
```

Whether daylight savings time is currently observed

```
function isDaylightSavings(d:Date):Boolean {
    var months:uint = 12;
    var offset:uint = d.timezoneOffset;
    var offsetCheck:Number;

    while (months--) {
        offsetCheck = (new Date(d.getFullYear(), months, 1)).timezoneOffset;

        if (offsetCheck != offset)
            return (offsetCheck > offset);
    }

    return false;
}
```

Today's start date, at midnight

```
function today(date:Date = null):Date {
    if (date == null)
        date = new Date();
    return new Date(date.fullYear, date.month, date.date, 0, 0, 0, 0);
}
```

Read Working with Date and Time online: https://riptutorial.com/actionscript-3/topic/1926/working-with-date-and-time

Chapter 18: Working With Display Objects

Syntax

1. `addChild(child)` - adds a new item to this object's child tree as the topmost element.
2. `addChildAt(child, index)` - adds a new item to this object's child tree at a specified position. The bottom-most item has index of 0.
3. `getChildAt(index)` - returns a child with given index.
4. `getChildIndex(child)` returns the index of a *direct* child of this object. Otherwise an exception is thrown.
5. `removeChild(child)` - removes the specified direct child from this object's child tree. Throws exception if the supplied child's parent is not equal to `this`.
6. `removeChildAt(index)` - removes a child selected by index instead of reference. Throws exception if the child tree is not this wide.
7. `removeChildren(beginIndex:int = 0, endIndex:int = 0x7fffffff))` - added in Flash Player 11, removes a subset of children by index range, or all children if called with no parameters.
8. `setChildIndex(child,index)` - changes the child's index to the new value, shifting all children in between to occupy the released spot.
9. `swapChildren(child1,child2)` - swaps the two children's positions in display list, not affecting positions of other children.
10. `swapChildrenAt(index1,index2)` - swaps children located by their indexes.

Remarks

The display list is actually a tree, and is visualized with depth first algorithm. Any object listed earlier will be displayed earlier, and might be obscured by objects listed later. All techniques that can be used against a tree can be applied to working with display list.

Examples

Introduction To The Display List

In AS3, display assets are not visible until they are added to the Display List.

The AIR/Flash runtime has a hierarchical display structure (parent child relationship where children can have their own children), with the `stage` being the top level parent.

To add something to the display list, you use `addChild` or `addChildAt`. Here is a basic example of drawing a circle and adding it to the display list:

```
var myCircle:Shape = new Shape();

myCircle.graphics.beginFill(0xFF0000); //red
myCircle.graphics.drawCircle(25, 25, 50);
myCircle.graphics.endFill();
```

```
this.addChild(myCircle); //add the circle as a child of `this`
```

To see the object in the example above, `this` (the context of the code) also must be on the display list, as well any parents it may have. In AS3, the `stage` is the top most parent.

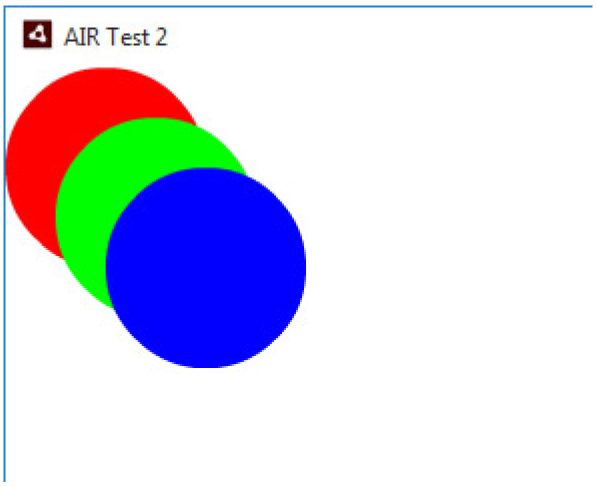
> Display objects can only have one parent. So if a child already has a parent, and you add it to another object, it will be removed from it's previous parent.

Z-Order / Layering

Let's say you replicated the code from the previous example so you had 3 circles:

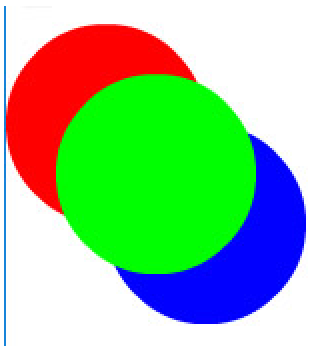
```
var redCircle:Shape = new Shape();
redCircle.graphics.beginFill(0xFF0000); //red
redCircle.graphics.drawCircle(50, 50, 50); //graphics.endFill is not required

var greenCircle:Shape = new Shape();
greenCircle.graphics.beginFill(0x00FF00); //green
greenCircle.graphics.drawCircle(75, 75, 50);

var blueCircle:Shape = new Shape();
blueCircle.graphics.beginFill(0x0000FF); //blue
blueCircle.graphics.drawCircle(100, 100, 50);

this.addChild(redCircle);
this.addChild(greenCircle);
this.addChild(blueCircle);
```

Since the `addChild` method adds the child on top of everything else in the same parent, you'll get this result with the items layered in the same order you use addChild:



If you wanted a child layered different relative to it's siblings, you can use `addChildAt`. With `addChildAt`, you pass in another parameter that indicates the index (z-order) the child should be at. `0` being the bottom most position/layer.

```
this.addChild(redCircle);
this.addChild(greenCircle);
this.addChildAt(blueCircle,0); //This will add the blue circle at the bottom
```



Now the blue circle is under it's siblings. If later on, you want to change the index of a child, you can use the `setChildIndex` method (on the child's parent).

```
this.setChildIndex(redCircle, this.numChildren - 1); //since z-index is 0 based, the top most
position is amount of children less 1.
```

This will rearrange the red circle so it's above everything else. The code above produces the exact same result as `this.addChild(redCircle)`.

Removing Display Objects

To remove objects, you have the converse `removeChild` and `removeChildAt` methods as well as the `removeChildren` method.

```
removeChild(redCircle); //this will take redCircle off the display list

removeChildAt(0); //this will take the bottom most object off the display list

removeChildren(); //this will clear all children from the display list

removeChildren(1); //this would remove all children except the bottom most

removeChildren(1,3); //this would remove the children at indexes 1, 2 & 3
```

Events

When a child is added to the display list, some events are fired on that child.

- `Event.ADDED`
- `Event.ADDED_TO_STAGE`

Conversely, there are also the remove events:

- `Event.REMOVED`
- `Event.REMOVED_FROM_STAGE`

Adobe Animate / Flash Professional

When dealing with FlashProfessional/Adobe Animate timelines, adding something to the timeline handles the display list nuances automatically. They added and removed from the display list automatically by the timeline.

However, it's good to keep in mind that:

> If you manipulate through code the parentage of a display object created by the timeline (by using addChild/setChildIndex), that child will no longer be removed automatically by the timeline and will need to be removed via code.

Layering

There can be situations when you decide that one set of display objects should always be above another set of objects, for example, arrows over heads, explosions over something that just exploded, etc. To perform this as simple as possible, you need to designate and create a set of `Sprite`s, arrange them in order from bottom to top, then just add all objects of "above" set to a layer above the one used for objects of "below" set.

```
var monsters:Vector.<Monster>;
var bullets:Vector.<Bullet>; // desired: bullets strictly above monsters
var monsterLayer:Sprite=new Sprite();
var bulletLayer:Sprite=new Sprite();
addChild(monsterLayer);
addChild(bulletLayer);
```

Then, whenever you add a `Monster` to the display list, add it to `monsterLayer`, and whenever you add a `Bullet`, add to `bulletLayer` to achieve the desired effect.

Remove all objects from the display list

If targeting Flash Player 11+, the built-in removeChildren method is the best way to remove all children:

```
removeChildren(); //a start and end index can be passed
```

For legacy applications, the same can be accomplished with a loop:

```
while (numChildren > 0) {
    removeChildAt(0);
}
```

Transiting from frames to manual content switching

Early on a Flash developer uses frames, as they are natively available in Flash player, to host various screens of their application (most often it's a game). Eventually they might stumble upon an issue that something goes wrong exactly because they have used frames, and overlooked the difficulties that arise from this, and seek ways to both retain their frame structure but also remove the obstacle of using frames with its complications. The solution is to use `Sprite` descendant classes, or exported frames as `MovieClip`s with a single frame (to those that design in Adobe Flash CS), and manually switch the contents with `addChild()` and `removeChild()`.

The manager class should have all of its child frame classes ready, and whenever a transition is called, a function similar to this can be used:

```
var frames:Vector.<DisplayObject>; // this holds instances to ALL children
var currentFrame_alt:int; // current frame. Can't use the property
function changeFrame(frame:int):void {
    removeChild(frames[currentFrame_alt]);
    addChild(frames[frame]);
    currentFrame_alt=frame;
}
```

All children can both dispatch and listen to events with `Event.ADDED_TO_STAGE` used as an entry point for whatever happens after `gotoAndStop()` targetting that frame, and any outgoing transitions can be coded as events based on strings, which are being listened in `Main` class, which then performs the transition.

```
frames[0].addEventListener("startGame",startGame); // assuming frame 0 is a "Play" button
function startGame(e:Event):void {
    changeFrame(1); // switch to frame 1 - will display frames[1]
}
```

Of course, the set of strings should be predefined, for example, intro screen could have two buttons to start the game, "Start game" and "Start muted" for example, and the buttons should dispatch different events, which will then be handled differently in manager class.

This pattern can go as deep as you need to. If any frame of the project contains a MovieClip with multiple frames, it can also be decoupled into sprites with this method.

Read Working With Display Objects online: https://riptutorial.com/actionscript-3/topic/1628/working-with-display-objects

Chapter 19: Working with Events

Remarks

Events are pieces of data that a program can create, exchange and react upon. The asynchronous event flow is dispatched over display list by Flash engine as a reaction on external events, such as mouse movements or another frame being displayed. Every other event flow and all event processing is synchronous, so if a piece of code has generated an event, all reactions on it are processed before the next line of code is executed, also if there are several listeners of an event, all of them would have run before the next event could be processed.

There are several major events associated with Flash programming. `Event.ENTER_FRAME` is generated before Flash draws another frame, it signals the entire display list to prepare to be drawn, and can be used as a synchronous timer. `MouseEvent.CLICK` and its siblings can be used to receive mouse input from the user, and `TouchEvent.TOUCH_TAP` is an analogue for touch screens. `KeyboardEvent.KEY_DOWN` and `KEY_UP` provide means to receive user input from the keyboard, however, their usage in mobile department is almost impossible due to devices having no physical keyboard. Finally, `Event.ADDED_TO_STAGE` is dispatched once a display object receives access to stage, and is included in the global display list that receives the entirety of events that can bubble up and down the display list.

Most events in Flash are component specific. If you are designing your own component that will use Flash events, use a `flash.events.Event` descendant class and its static `String` properties to create your component's event set.

Examples

Custom events with event data

```
package
{
    import flash.events.Event;

    public class CustomEvent extends Event
    {
        public static const START:String = "START";
        public static const STOP:String  = "STOP";

        public var data:*;

        public function CustomEvent(type:String, data:*,
                                    bubbles:Boolean=false, cancelable:Boolean=false)
        {
            super(type, bubbles, cancelable);

            if (data)
                this.data = data;
        }
    }
}
```

To dispatch a custom event:

```
var dataObject:Object = {name: "Example Data"};

dispatchEvent(new CustomEvent(CustomEvent.START, dataObject))
```

To listen for custom events:

```
addEventListener(CustomEvent.STOP, stopHandler);

function stopHandler(event:CustomEvent):void
{
    var dataObject:* = event.data;
}
```

Event handling off the display list

```
package {
import flash.events.EventDispatcher;

public class AbstractDispatcher extends EventDispatcher {

    public function AbstractDispatcher(target:IEventDispatcher = null) {
        super(target);
    }

}
}
```

To dispatch an event on an instance:

```
var dispatcher:AbstractDispatcher = new AbstractDispatcher();
dispatcher.dispatchEvent(new Event(Event.CHANGE));
```

To listen for events on an instance:

```
var dispatcher:AbstractDispatcher = new AbstractDispatcher();
dispatcher.addEventListener(Event.CHANGE, changeHandler);

function changeHandler(event:Event):void
{
}
```

Basic event handling

Flash dispatches `Events` for most of its objects. One of the most basic event is `ENTER_FRAME`, which is dispatched (at the framerate of the SWF) on every display list object.

```
import flash.display.Sprite;
import flash.events.Event;

var s:Sprite = new Sprite();
s.addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(e:Event)
{
    trace("I am called on every frame !");
}
```

This function will be called asynchronously on every frame. This means the function that you assign as the `onEnterFrame` event handler is processed before any other ActionScript code that is attached to the affected frames.

Add your own events

You can create your own events and dispatch them, by extending the `Event` class.

```
import flash.events.Event;

class MyEvent extends Event
{
    var data: String;

    static public var MY_EVENT_TYPE = "my_event_my_event_code";

    public function MyEvent(type: String, data: String)
    {
        this.data = data;
    }

    override public function clone():Event
    {
        return new MyEvent(type, data);
    }
}
```

You can then dispatch it and listen to it, using an `EventDispatcher`. Note that most flash objects are event dispatchers.

```
import flash.events.EventDispatcher;

var d = new EventDispatcher();
d.addEventListener(MyEvent.MY_EVENT_TYPE, onType);

function onType(e: MyEvent)
{
    trace("I have a string: "+e.data);
}

d.dispatchEvent(new MyEvent(MyEvent.MY_EVENT_TYPE, "Hello events!"));
```

Note that the `clone` method is required if you want to redispatch your event.

Simple Mouse Event Structure

Through the use of `event types` you can easily reduce code bloat that often occurs when defining events for many objects on stage by filtering events in 1 function rather than defining many event handling functions.

Imagine we have 10 objects on stage named `object1`, `object2` ... `object10`

You could do the following:

```
var i: int = 1;
while(getChildByName("object"+i) != null){
    var obj = getChildByName("object"+i)
    obj.addEventListener(MouseEvent.CLICK, ObjectMouseEventHandler);
    obj.addEventListener(MouseEvent.MOUSE_OVER, ObjectMouseEventHandler);
    obj.addEventListener(MouseEvent.MOUSE_OUT, ObjectMouseEventHandler);
    obj.alpha = 0.75;
    i++;
}

function ObjectMouseEventHandler(evt:Event)
{
    if(evt.type == "click")
    {
        trace(evt.currentTarget + " has been clicked");
    }
    else
    {
        evt.currentTarget.alpha = evt.type == "mouseOver" ? 1 : 0.75;
    }
}
```

Benefits to this method include:

1. Not needing to specify the quantity of objects to apply events to.
2. Not needing to know specifically what object was interacted with yet still apply functionality.
3. Easily applying events in bulk.

Read Working with Events online: https://riptutorial.com/actionscript-3/topic/1925/working-with-events

Chapter 20: Working with Geometry

Examples

Getting the angle between two points

Using vanilla mathematics:

```
var from:Point = new Point(100, 50);
var to:Point = new Point(80, 95);

var angle:Number = Math.atan2(to.y - from.y, to.x - from.x);
```

Using a new vector representing the difference between the first two:

```
var difference:Point = to.subtract(from);

var angle:Number = Math.atan2(difference.y, difference.x);
```

> Note: `atan2()` returns radians, not degrees.

Getting the distance between two points

Using vanilla mathematics:

```
var from:Point = new Point(300, 10);
var to:Point = new Point(75, 40);

var a:Number = to.x - from.x;
var b:Number = to.y - from.y;

var distance:Number = Math.sqrt(a * a + b * b);
```

Using inbuilt functionality of `Point`:

```
var distance:Number = to.subtract(from).length; // or
var distance:Number = Point.distance(to, from);
```

Converting radians to degrees

```
var degrees:Number = radians * 180 / Math.PI;
```

Converting degrees to radians

```
var radians:Number = degrees / 180 * Math.PI;
```

The value of a circle in degrees and radians

- A whole circle is `360` degrees or `Math.PI * 2` radians.
- Half of those values follows to be `180` degrees or `Math.PI` radians.
- A quarter is then `90` degrees or `Math.PI / 2` radians.

To get a segment as a percentage of a whole circle in radians:

```
function getSegment(percent:Number):Number {
    return Math.PI * 2 * percent;
}

var tenth:Number = getSegment(0.1); // One tenth of a circle in radians.
```

Moving a point along an angle

Assuming you have the angle you'd like to move in and an object with `x` and `y` values you want to move:

```
var position:Point = new Point(10, 10);
var angle:Number = 1.25;
```

You can move along the `x` axis with `Math.cos`:

```
position.x += Math.cos(angle);
```

And the `y` axis with `Math.sin`:

```
position.y += Math.sin(angle);
```

You can of course multiply the result of `Math.cos` and `Math.sin` by the distance you want to travel:

```
var distance:int = 20;

position.x += Math.cos(angle) * distance;
position.y += Math.sin(angle) * distance;
```

> Note: The input angle must be in radians.

Determine if a point is inside a rectangle area

You can test whether a point is inside a rectangle using `Rectangle.containsPoint()`:

```
var point:Point = new Point(5, 5);
var rectangle:Rectangle = new Rectangle(0, 0, 10, 10);

var contains:Boolean = rectangle.containsPoint(point); // true
```

Read Working with Geometry online: https://riptutorial.com/actionscript-3/topic/3201/working-with-geometry

Chapter 21: Working with numerical values

Examples

Whether a number is an even value

```
function isEven(n:Number):Boolean {
    return ((n & 1) == 0);
}
```

Examples:

```
isEven(1); // false
isEven(2); // true

isEven(1.1); // false
isEven(1.2); // false
isEven(2.1); // true
isEven(2.2); // true
```

Whether a number is an odd value

```
function isOdd(n:Number):Boolean {
    return ((n & 1) == 1);
}
```

Examples:

```
isOdd(1); // true
isOdd(2); // false

isOdd(1.1); // true
isOdd(1.2); // true
isOdd(2.1); // false
isOdd(2.2); // false
```

Rounding to nearest X

To round a value to the nearest multiple of x:

```
function roundTo(value:Number, to:Number):Number {
    return Math.round(value / to) * to;
}
```

Example:

```
roundTo(8, 5); // 10
roundTo(17, 3); // 18
```

Roundoff errors of floating point numbers

```
/**
 * @param n Number to be rounded.
 * @param precision Decimal places.
 * @return Rounded Number
 */
function roundDecimal(n:Number, precision:Number):Number {
    var factor:int = Math.pow(10, precision);
    return (Math.round(n * factor) / factor);
}
```

Examples:

```
trace(0.9 - 1); // -0.09999999999999998

trace(roundDecimal(0.9 - 1, 1));     // -0.1
trace(roundDecimal(0.9 - 1, 2));     // -0.1

trace(roundDecimal(0.9 - 1.123, 1)); // -0.2
trace(roundDecimal(0.9 - 1.123, 2)); // -0.22
trace(roundDecimal(0.9 - 1.123, 3)); // -0.223
```

Displaying numbers with required precision

```
var a:Number=0.123456789;
trace(a);                      // 0.123456789
trace(a.toPrecision(4));       // 0.1235
trace(a.toFixed(4));           // 0.1235
trace(a.toExponential(4));     // 1.2345e-1
trace(a.toString(16));         // 0 - works for integer part only
var b:Number=12345678.9876543; // a bigger number to display rounding
trace(b);                      // 12345678.9876543
trace(b.toPrecision(4));       // 1.235e+7
trace(b.toFixed(4));           // 12345678.9877
trace(b.toExponential(4));     // 1.2345e+7
trace(b.toString(16));         // bc614e
b=1.0e+16;
trace(b.toString(36));         // 2qgpckvng1s
```

Read Working with numerical values online: https://riptutorial.com/actionscript-3/topic/1899/working-with-numerical-values

Chapter 22: Working with Sound

Syntax

- Sound.play(startTime:Number = 0, loops:int = 0, sndTransform:flash.media:SoundTransform = null):SoundChannel // Plays a loaded sound, returns a SoundChannel

Examples

Stop Playing a Sound

```
import flash.net.URLRequest;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.events.Event;

var snd:Sound; = new Sound();
var sndChannel:SoundChannel
var sndTimer:Timer;

snd.addEventListener(Event.COMPLETE, soundLoaded);
snd.load(new URLRequest("soundFile.mp3")); //load after adding the complete event

function soundLoaded(e:Event):void
{
    sndChannel = snd.play();

    //Create a timer to wait 1 second
    sndTimer = new Timer(1000, 1);
    sndTimer.addEventListener(TimerEvent.TIMER, stopSound, false, 0, true);
    sndTimer.start();
}

function stopSound(e:Event = null):void {
    sndChannel.stop(); //Stop the sound
}
```

Infinite looping a sound

```
import flash.net.URLRequest;
import flash.media.Sound;
import flash.events.Event;

var req:URLRequest = new URLRequest("filename.mp3");
var snd:Sound = new Sound(req);

snd.addEventListener(Event.COMPLETE, function(e: Event)
{
    snd.play(0, int.MAX_VALUE); // There is no way to put "infinite"
}
```

You also don't need to wait for sound to load before calling `play()` function. So this will do the same job:

```
snd = new Sound(new URLRequest("filename.mp3"));
snd.play(0, int.MAX_VALUE);
```

And if you really want to loop sound inifinite time for some reason (`int.MAX_VALUE` will loop 1s sound for about 68 years, not counting the pause an mp3 causes...) you can write something like this:

```
var st:SoundChannel = snd.play();
st.addEventListener(Event.SOUND_COMPLETE, repeat);
function repeat(e:Event) {
    st.removeEventListener(Event.SOUND_COMPLETE, repeat);
    (st = snd.play()).addEventListener(Event.SOUND_COMPLETE, repeat);
}
```

`play()` function returns new instance of `SoundChannel` object each time it's called. We assgin it to variable and listen for its SOUND_COMPLETE event. In the event callback, listener is removed from current `SoundChannel` object and new one is created for new `SoundChannel` object.

Load and play an external sound

```
import flash.net.URLRequest;
import flash.media.Sound;
import flash.events.Event;

var req:URLRequest = new URLRequest("click.mp3");
var snd:Sound = new Sound(req);

snd.addEventListener(Event.COMPLETE, function(e: Event)
{
    snd.play();
}
```

Read Working with Sound online: https://riptutorial.com/actionscript-3/topic/2043/working-with-sound

Chapter 23: Working with Timeline

Examples

Referencing the main timeline or document class from within other MovieClips

In the timeline of any `DisplayObject` that is attached as a descendant of the display tree, you can utilise the `root` property. This property points to the main timeline in the case of no custom document class, or the document class if you do define one.

Because `root` is typed `DisplayObject`, the compiler will not allow you to access custom methods or properties defined on the main timeline or within your document class as:

```
root.myCustomProperty = 10;
root.myCustomMethod();
```

To get around this, you can typecast `root` to your document class in the case where you have a document class:

```
(root as MyDocumentClass).myCustomMethod();
```

Or `MovieClip` in the case of no document class:

```
(root as MovieClip).myCustomMethod();
```

The reason casting to `MovieClip` works here is because `MovieClip` is `dynamic`. This means that the compiler allows runtime properties and method to be declared on it, preventing compile-time errors when attempting to access properties or methods that are not explicitly defined on `MovieClip`. The downside to this is that you lose all compile-time type safety. You are much better off declaring a document class and casting to that.

Read Working with Timeline online: https://riptutorial.com/actionscript-3/topic/2459/working-with-timeline

Chapter 24: Working with Timers

Examples

Countdown timer example

```
package {
import flash.events.TimerEvent;
import flash.utils.Timer;

public class CountdownTimer extends Timer {

    public var time:Number = 0;

    public function CountdownTimer(time:Number = Number.NEGATIVE_INFINITY, delay:Number =
1000) {
        super(delay, repeatCount);

        if (!isNaN(time))
            this.time = time;

        repeatCount = Math.ceil(time / delay);

        addEventListener(TimerEvent.TIMER, timerHandler);
        addEventListener(TimerEvent.TIMER_COMPLETE, timerCompleteHandler);
    }

    override public function start():void {
        super.start();
    }

    protected function timerHandler(event:TimerEvent):void {
        time -= delay;
    }

    protected function timerCompleteHandler(event:TimerEvent):void {
    }

    override public function stop():void {
        super.stop();
    }

    public function dispose():void {
        removeEventListener(TimerEvent.TIMER, timerHandler);
        removeEventListener(TimerEvent.TIMER_COMPLETE, timerCompleteHandler);
    }

}
}
```

This `CountdownTimer` extends `Timer`, and is used exactly the same, except that time counts down.

Example usage:

```
var timer:CountdownTimer = new CountdownTimer(5000);
```

```
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.addEventListener(TimerEvent.TIMER_COMPLETE, completeHandler);
timer.start();

function timerHandler(event:TimerEvent):void {
    trace("Time remaining: " + event.target.time);
}

function completeHandler(event:TimerEvent):void {
    trace("Timer complete");
}
```

Above example would output:

```
[trace] Time remaining: 4000
[trace] Time remaining: 3000
[trace] Time remaining: 2000
[trace] Time remaining: 1000
[trace] Time remaining: 0
[trace] Timer complete
```

Intervals and timeouts

```
import flash.utils.*;
var intervalId:uint=setInterval(schroedingerCat,1000);
// execute a function once per second and gather interval ID
trace("Cat's been closed in the box.");
function schroedingerCat():void {
    if (Math.random()<0.04) {
        clearInterval(intervalId); // stop repeating by ID
        trace("Cat's dead.");
        return;
    }
    trace("Cat's still alive...");
}

var bombId:uint;
function plantBomb(seconds:Number):uint {
    trace("The bomb has been planted, and will blow in "+seconds.toFixed(3)+" seconds!");
    var id:uint=setTimeout(boom,seconds*1000); // parameter is in milliseconds
    return id;
}
function defuseBomb(id:uint):void {
    clearTimeout(id);
    trace("Bomb with id",id,"defused!");
}
function boom():void {
    trace("BOOM!");
}
```

`setInterval()` is used to perform repeated tasks asynchronously as specified intervals. Internal `Timer` object is used, the returned value of type `uint` is its internal ID, by which you can access and stop the repeating by calling `clearInterval()`. `setTimeout()` and `clearTimeout()` work similarly, but the call to supplied function is done only once. You can supply additional arguments to both set functions, these will be passed to the function in order. The number of arguments and their type is not checked at compile time, so should you supply a weird combination of arguments, or a function that requires them and receives none, an error "Error #1063: Argument count mismatch" is raised.

You can perform both activities of `setInterval` and `setTimeout` with regular `Timer` objects, by either using 0 or 1 for `repeatCount` property, 0 for indefinite repeats, 1 for one.

Random timer example

```
package {
    import flash.events.TimerEvent;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class RandomTimer extends Timer {

        public var minimumDelay:Number;
        public var maximumDelay:Number;
        private var _count:uint = 0;
        private var _repeatCount:int = 0;

        public function RandomTimer(min:Number, max:Number, repeatCount:int = 0) {
            super(delay, repeatCount);

            minimumDelay = min;
            maximumDelay = max;
            _repeatCount = repeatCount;
        }

        override public function start():void {
            delay = nextDelay();
            addEventListener(TimerEvent.TIMER, timerHandler);
            super.start();
        }

        private function nextDelay():Number {
            return (minimumDelay + (Math.random() * (maximumDelay - minimumDelay)));
        }

        override public function stop():void {
            removeEventListener(TimerEvent.TIMER, timerHandler);
            super.stop();
        }

        protected function timerHandler(event:TimerEvent):void {
            _count++;
            if ((_repeatCount > 0) && (_count >= _repeatCount)) {
                stop();
                dispatchEvent(new TimerEvent(TimerEvent.TIMER_COMPLETE));
            }
            delay = nextDelay();
        }

        override public function reset():void {
            _count = 0;
            super.reset();
        }
    }
}
```

This `RandomTimer` extends `Timer`, and is used exactly the same, except that it dispatches at random intervals.

Example usage, dispatching randomly between 1 and 5-seconds:

```
var t:int = getTimer();

var timer:RandomTimer = new RandomTimer(1000, 5000);
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.start();

function timerHandler(event:TimerEvent):void {
    trace("Time since last dispatch: " + (getTimer() - t));
    t = getTimer();
}
```

Above example would output:

```
[trace] Time since last dispatch: 1374
[trace] Time since last dispatch: 2459
[trace] Time since last dispatch: 3582
[trace] Time since last dispatch: 1335
[trace] Time since last dispatch: 4249
```

Read Working with Timers online: https://riptutorial.com/actionscript-3/topic/2798/working-with-timers

Chapter 25: Working with Video

Examples

Load and play external video file

reference : `NetConnection` , `NetStream` , `Video`

related topics : Working with Sound

Basic example of playing an external video file (FLV, MP4, F4V). Code will also play M4A audio files.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);

var myVideo:Video = new Video();
addChild(myVideo);

myVideo.attachNetStream(ns);

ns.play("http://www.yourwebsite.com/somefile.mp4");
```

Notice the code used a `nc.connect.null`? This is because, in this case, there is no need to create a two-way peer to peer connection (eg: as expected in a video chat app) since we are playing a stored file.

By setting a `nc.connect.null` it is required to provide a link to a file that is either on a web server or one that is local (same location/folder) to the running SWF.

- For a **web** file use : `ns.play("http://www.yourwebsite.com/somefile.mp4");`
- For a **local** file use : `ns.play("somefile.mp4");`

With NetStatusEvent

```
package {
    import flash.events.NetStatusEvent;
    import flash.net.NetStream;
    import flash.net.NetConnection;
    import flash.events.Event;
    import flash.media.Video;
    import flash.display.Sprite;
    public class VideoWithNetStatus extends Sprite {
        private var video:Video = new Video();
        private var nc:NetConnection;
        private var ns:NetStream;

        public function VideoWithNetStatus() {
            nc = new NetConnection();
            nc.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
            nc.connect(null);//or media server url
        }

         private function onStatus(e:NetStatusEvent):void{
             switch(e.info.code){
                 case 'NetConnection.Connect.Success':
                     connectStream();
                 break;
                 default:
                     trace(e.info.code);//to see any unhadled events
             }
         }
         private function connectStream():void{
             ns = new NetStream(nc);
             ns.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
             addChild(video);
             video.attachNetStream(ns);
             ns.play('url/to/video.flv');
         }
    }
}
```

Read Working with Video online: https://riptutorial.com/actionscript-3/topic/2406/working-with-video

Credits

S. No	Chapters	Contributors
1	Getting started with ActionScript 3	BadFeelingAboutThis, Community, Jason Sturges, joshtynjala, Kit Grose, null, Programmer Dancuk, Vesper
2	Binary data	Paweł Audionysos
3	Bitmap Manipulation and Filtering	payam_sbr, VC.One
4	Display List Lifecycle	Jason Sturges, mnoronha
5	Drawing Bitmaps	BadFeelingAboutThis, Marty, Vesper, www0z0k
6	Game Development Basics	payam_sbr
7	Loading External Files	BadFeelingAboutThis, Marty
8	Object Oriented Programming	HITMAN, Marty, null, www0z0k
9	Optimizing Performance	Community, Marty
10	Random Value Generation	alebianco, BadFeelingAboutThis, HITMAN, Jason Sturges, Marty, mnoronha, null, Vesper, xims
11	Responsive Application Design	BadFeelingAboutThis, null, Vesper
12	Sending and Receiving Data From Servers	Marty, null, xims
13	Singleton Pattern	commovere, Jason Sturges, mnoronha, null
14	Types	BadFeelingAboutThis, joshtynjala, Marty, Paweł Audionysos
15	Understanding the "Error 1009: Cannot access a property or method of a null object reference"	Vesper, www0z0k

16	Using the Proxy Class	Organis
17	Working with Date and Time	Jason Sturges, mnoronha
18	Working With Display Objects	BadFeelingAboutThis, Jason Sturges, Vesper
19	Working with Events	blue112, Jason Sturges, mnoronha, null, VC.One, Vesper, Zze
20	Working with Geometry	Marty, mnoronha, www0z0k
21	Working with numerical values	Jason Sturges, Jonny Henly, Marty, Vesper
22	Working with Sound	BadFeelingAboutThis, blue112, Paweł Audionysos, VC.One
23	Working with Timeline	Marty
24	Working with Timers	Jason Sturges, mnoronha, Vesper, www0z0k
25	Working with Video	VC.One, www0z0k