Quickstart Guide for Browser-Based Experiences

What's New in Flash Player 11

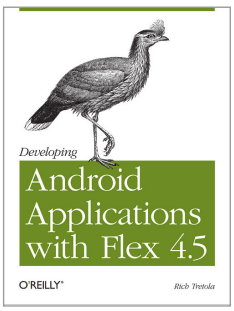
O'REILLY®

Adobe Developer Library

Joseph Labrecque

O'REILLY®

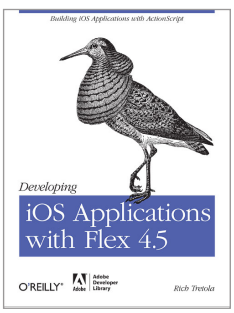
Related Ebooks



Developing Android Applications with Adobe Flex 4.5

By Rich Tretola
Released: May 2011
Ebook: **$16.99**

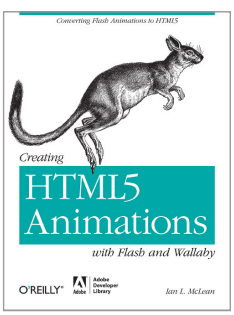
Buy Now



Developing iOS Applications with Flex 4.5

By Rich Tretola
Released: August 2011
Ebook: **$12.99**

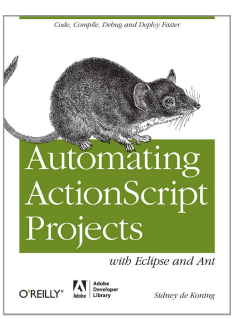
Buy Now



Creating HTML5 Animations with Flash and Wallaby

By Ian L. McLean
Released: September 2011
Ebook: **$12.99**

Buy Now



Automating ActionScript Projects with Eclipse and Ant

By Sidney de Koning
Released: October 2011
Ebook: **$9.99**

Buy Now

What's New in Flash Player 11

Joseph Labrecque

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

What's New in Flash Player 11
by Joseph Labrecque

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Mary Treseler
Production Editor: Dan Fauxsmith
Proofreader: O'Reilly Production Services
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Revision History for the First Edition:

See *http://oreilly.com/catalog/errata.csp?isbn=9781449311094* for release details.

ISBN: 978-1-449-31109-4

[LSI]

1323195824

Adobe Developer Library, a copublishing partnership between O'Reilly Media Inc., and Adobe Systems, Inc., is the authoritative resource for developers using Adobe technologies. These comprehensive resources offer learning solutions to help developers create cutting-edge interactive web applications that can reach virtually anyone on any platform.

With top-quality books and innovative online resources covering the latest tools for rich-Internet application development, the *Adobe Developer Library* delivers expert training straight from the source. Topics include ActionScript, Adobe Flex®, Adobe Flash®, and Adobe Acrobat®.

Get the latest news about books, online resources, and more at *http://adobedeveloperlibrary.com*.

Table of Contents

Preface ..... **ix**

1. Improvements to the MovieClip and Drawing APIs **1**
Cubic Bezier curves 1
DisplayObjectContainer.removeChildren() 3
MovieClip.isPlaying 5

2. External Image Capabilities **9**
Enhanced High-Resolution Bitmap Support 9
Asynchronous Bitmap Decoding 11
JPEG-XR Support 14

3. Stage3D: High Performance Visuals **17**
Stage3D Accelerated Graphics Rendering 17
Elements of Stage3D 18
Stage3D Example Using Away3D 20
Stage3D Example Using Starling 22
Tooling Support for Stage3D 26

4. Audio and Video Enhancements **29**
H.264/AVC Software Encoding for Cameras 29
Encoding H.264 within Flash Player 11 30
Reading an H.264 Stream into Flash Player 11 32
G.711 Audio Compression for Telephony 35

5. Data Transfer Additions **39**
Native JSON (JavaScript Object Notation) Support 39
JSON.parse() 40
JSON.stringify() 42
Socket Progress Events 45

6. Runtime Enhancements 49
Native 64-bit Support 49
High-Efficiency SWF Compression Support 49
Garbage Collection Advice 50

7. Flash Player Security 55
Protected HTTP Dynamic Streaming and Flash Access Content Protection Support for Mobile 55
Secure Random Number Generator 56
Secure Sockets Support 59

Appendix: Additional Resources 63

Preface

Introduction to Adobe Flash Player 11

This book will detail the various enhancements, new functionalities, and general improvements available in this new version of Adobe Flash Player. Each item is explained in detail, and when possible, a series of screen captures and a full code example will be provided, enabling you to both grasp the new feature in a visual way, and integrate the feature into your own code quickly, based upon example.

During the development cycle between Flash Player 10 and Flash Player 10.1, Adobe rewrote much of the underlying code in order to lay a solid foundation that not only benefited traditional web experiences, but could also be brought over into new areas such as mobile and television. This foundation has served to make Flash Player 10.1–10.3 very stable while allowing Adobe to begin adding small features upon each incremental release. In contrast to these incremental versions, with Flash Player 11 we begin to see the rapid evolution of the Flash runtime into something not only great at interactive, gaming, media distribution, and enterprise applications...but into something that pushes all these areas way beyond their previous limitations.

With the recent rise of expanding web technologies like *HTML5* (including *HTML/CSS/JavaScript*), it is very important that the Flash Player evolves in a way which not only showcases why it is still relevant, but also why it is still (in many cases) the ideal technology platform for advanced interaction on the Web and beyond. With Adobe ramping up the Flash Player release schedule along with more iterative tooling support in Flash Professional and Flash Builder, not to mention a number of new community partnerships in support of the platform from both independent framework and third-party tooling support, we can expect great things in future incremental releases of Flash Player 11 and within the entire platform ecosystem.

Who This Book Is For

This book is written for both veteran Flash Platform developers curious about enhancements in Flash Player 11, as well as those who are entirely new to the platform.

The reader will acquire a solid overview of new features along with usable code examples.

Who This Book Is Not For

This book is not an in-depth study of ActionScript or Flash Player internals. Neither is this meant to be an exhaustive overview of complex new features such as `Stage3D`. Entire books will be written which cover such advanced topics. This book will simply provide the reader with a holistic foundation to be built upon using other resources.

Conventions Used in This Book

The following typographical conventions are used in this book:

Menu options
: Menu options are shown using the→character, such as File→Open.

Italic
: Italic indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: This is used for program listings, as well as within paragraphs, to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: This shows commands or other text that should be typed literally by you.

`Constant width italic`
: This shows text that should be replaced with user-supplied values or by values determined by context.

This Book's Example Files

You can download the example files for this book from this location:

http://examples.oreilly.com/0636920021698/

All code examples are written using pure ActionScript 3, when possible, and are not tied to any framework or IDE. This is to allow the reader to implement the code examples in whichever environment he/she chooses.

The examples are all ActionScript 3 (`AS3`) class files which can be compiled to *SWF* using Flash Professional, Flash Builder, FDT, FlashDevelop, or any other IDE which can be configured to process and output Flash content for Flash Player 11.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*What's New in Flash Player 11* by Joseph Labrecque (O'Reilly). Copyright 2012 Fractured Vision Media, LLC, 978-1-449-31110-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

How to Use This Book

Development rarely happens in a vacuum. In today's world, email, Twitter, blog posts, co-workers, friends, and colleagues all play a vital role in helping you solve development problems. Consider this book yet another resource at your disposal to help you solve the development problems you will encounter. The content is arranged in such a way that solutions should be easy to find and easy to understand. However, this book does have a big advantage: it is available anytime of the day or night.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://shop.oreilly.com/product/0636920021698.do

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

I'd first like to thank my wife, Leslie, and our daughters, Paige and Lily, for being so understanding of the work that I do. It's strange stuff, I know.

Thanks also to Rich Tretola, Chris Griffith, Michelle Yaiser, Brian Rinaldi, Richard Galvan, O'Reilly Media, Adobe Systems, and the Adobe Education Leader and Adobe Community Professional organizations.

CHAPTER 1

Improvements to the MovieClip and Drawing APIs

Flash Player began life in the mid-1990s as a web-based media animation and display technology. For much of its history, it has been relied on for graphically intense, functional, and beautiful image rendering and manipulation. With Flash Player 11, the graphics and vector drawing technology which is so core to Flash Player is extended and improved upon in some rather useful ways.

Cubic Bezier curves

We have an addition to the graphics drawing APIs in this release of Flash Player which allows the simple creation of *Cubic Bezier Curves* without having to do a lot of complex equations on your own, each time you want to draw a new curve. The new `cubicCurveTo()` method takes six arguments to function correctly; a set of `x` and `y` coordinates for the first control point, a similar set for the second control point, and a set of coordinates for the anchor point.

Bezier curves are widely used in computer graphics to model smooth curves through the use of four distinct points: a start point, an end point, and two anchor points which inform the direction and pull of the drawn curve.

The curve will begin wherever the current line is – we can use the `moveTo()` method to precisely position the start point just as is done on other graphics API calls. The two control points influence the curve of the line, and the anchor point will be the end of the drawn curve. This is illustrated visually in the following figure.

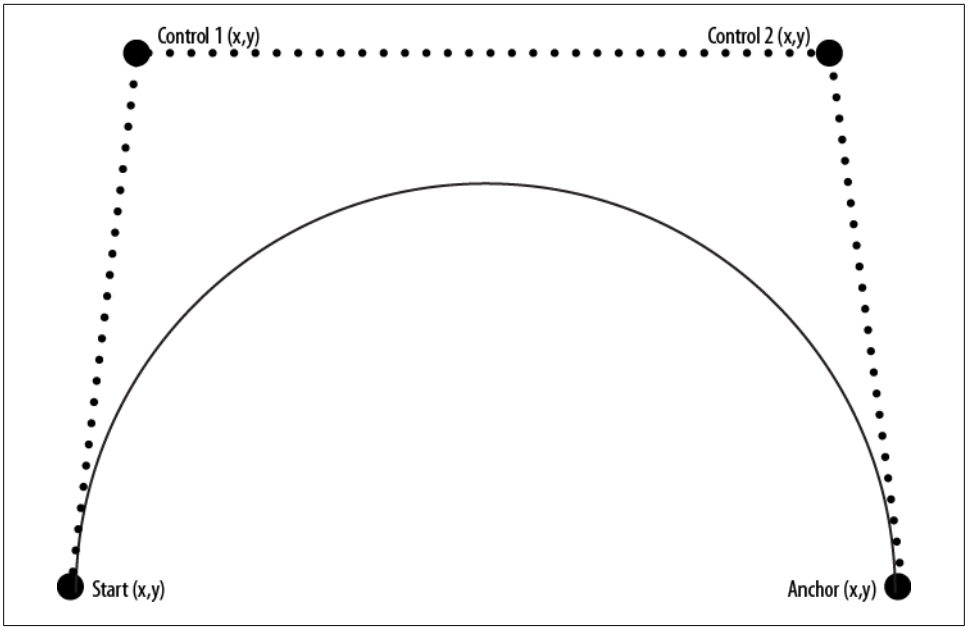


Figure 1-1. How Cubic Bezier curves work

In the example below, we create a *Sprite* within which the new `cubicCurveTo()` method is invoked in order to draw a cubic Bezier arc across the stage.

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

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class CubicBezierCurve extends Sprite {

        private var drawingHolder:Sprite;

        public function CubicBezierCurve() {
            generateDisplayObjects();
        }

        protected function generateDisplayObjects():void {
            drawingHolder = new Sprite();
            drawingHolder.graphics.moveTo(20, stage.stageHeight-20);
            drawingHolder.graphics.lineStyle(5,0x000000);

            drawingHolder.graphics.cubicCurveTo(50, 50, stage.stageWidth-50, 50,
stage.stageWidth-20, stage.stageHeight-20);
            addChild(drawingHolder);
        }

    }
}
```

This will render a SWF similar in appearance to Figure 1-2.

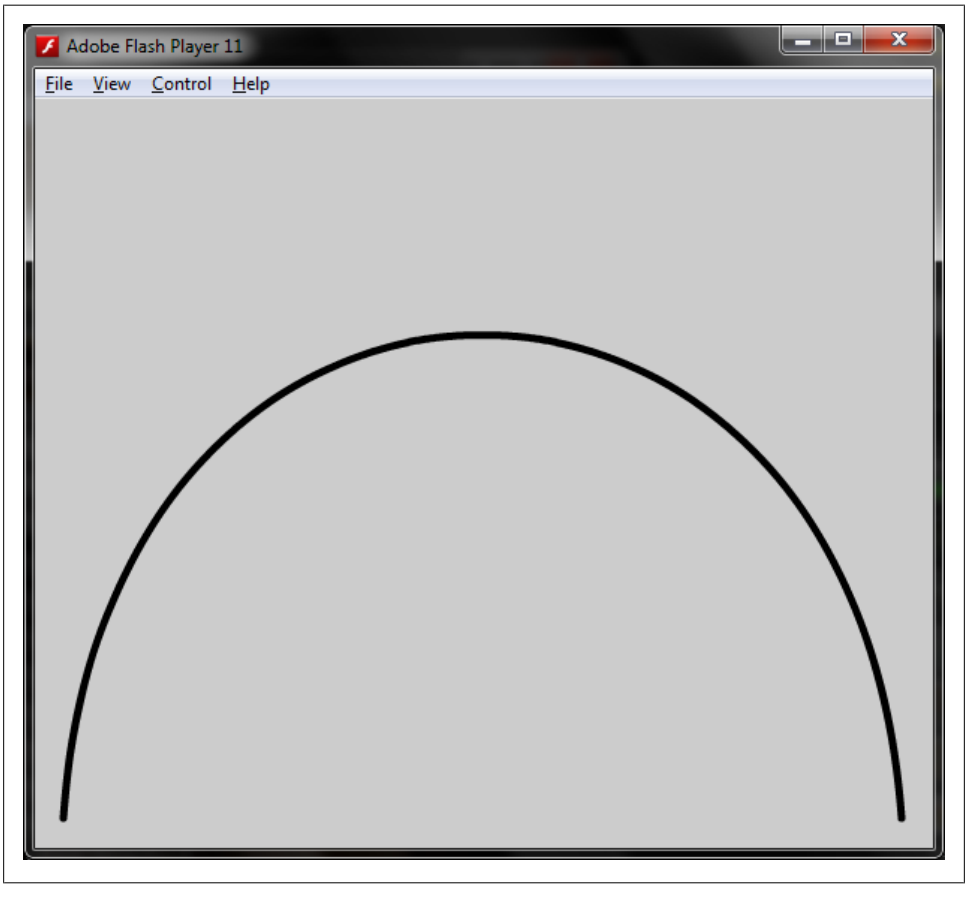
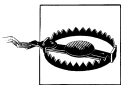


Figure 1-2. Cubic Bezier curve

DisplayObjectContainer.removeChildren()

Previous to Flash Player 11, if a developer wanted to remove all children from a container object, it was necessary to first determine how many children were present through `DisplayObjectContainer.numChildren` and then loop over each of these child objects, removing them one at a time.

With the `DisplayObjectContainer.removeChildren()` method, one simple command can be used to remove all children of a parent container, making them all available for garbage collection.



You'll want to be sure to remove any event listeners or other references to these children before invoking `removeChildren`, else the garbage collector may not be able to totally free the memory allocated to these objects.

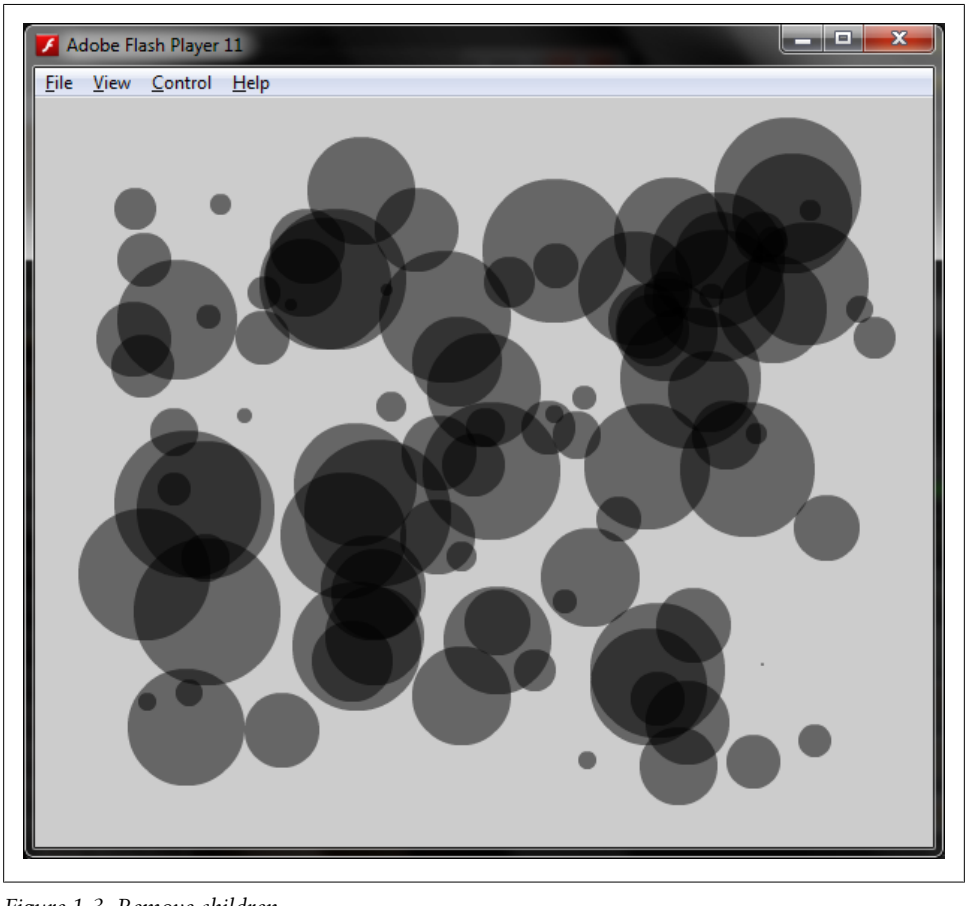


Figure 1-3. Remove children

In the following example, we will generate a number of dynamic `MovieClip` symbols upon the `Stage`. We add an event listener to the `Stage` as well, listening for a simple `MouseEvent.CLICK` event – which then invokes a method to remove all of these `MovieClips` with one simple command: `stage.removeChildren()`.

```
package {
    import flash.display.MovieClip;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]
```

```
    public class RemoveAllChildren extends Sprite {
        public function RemoveAllChildren() {
            generateDisplayObjects();
        }

        protected function generateDisplayObjects():void {
            for(var i:int=100; i>0; i--){
                var childMC:MovieClip = new MovieClip();
              var randX:Number = Math.floor(Math.random() * (1+stage.stageWidth-100))
+ 50;
               var randY:Number = Math.floor(Math.random() * (1+stage.stageHeight-100))
+ 50;
                var randD:Number = Math.floor(Math.random() * 50-10) + 10;
                childMC.x = randX;
                childMC.y = randY;
                childMC.graphics.beginFill(0x000000, 0.5);
                childMC.graphics.drawCircle(0, 0, randD);
                childMC.graphics.endFill();
                this.addChild(childMC);
            }
            stage.addEventListener(MouseEvent.CLICK, removeAllChildren);
        }

        protected function removeAllChildren(e:MouseEvent):void {
            stage.removeChildren();
        }
    }
}
```

MovieClip.isPlaying

It's actually sort of amazing that we haven't had this property in older versions of Flash Player. `MovieClip` instances are unique in that they contain their own timeline, independent from the main timeline. Often, a developer will want to know whether or not a specific `MovieClip` instance is actually playing or not, and this has traditionally involved monitoring the current frame of the `MovieClip` to determine whether or not it is changing over time.

Making use of this new functionality is very direct, as `MovieClip.isPlaying` is simply a property of every `MovieClip` instance, which, when invoked, returns a `Boolean` value of `true` for playing and `false` for stopped. In the following example; we create a `MovieClip`, add it to the `DisplayList`, and then write the `isPlaying` property out onto a `TextField`.

```
package {
    import flash.display.MovieClip;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.text.TextField;
    import flash.text.TextFormat;
```

```
    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class CheckPlaying extends Sprite {

        private var face:MovieClip;
        private var traceField:TextField;

        public function CheckPlaying() {
            generateDisplayObjects();
        }

        protected function generateDisplayObjects():void {
            face = new AngryFace() as MovieClip;
            face.x = stage.stageWidth/2;
            face.y = stage.stageHeight/2;
            face.stop();
            face.addEventListener(MouseEvent.CLICK, toggleFacePlaying);
            addChild(face);

            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 26;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.autoSize = "left";
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);

            stage.addEventListener(Event.ENTER_FRAME, checkPlaying);
        }

        protected function toggleFacePlaying(e:MouseEvent):void {
            if(face.isPlaying){
                face.stop();
            }else{
                face.play();
            }
        }

        protected function checkPlaying(e:Event):void {
            traceField.text = "MovieClip is playing? => " + face.isPlaying;
        }
    }
}
```

Figure 1-4. MovieClip.isPlaying

The result of this code can be seen fully rendered in Figure 1-5. When clicking upon the `MovieClip`, its playback is toggled, and the `isPlaying` Boolean is measured and written onto the screen.

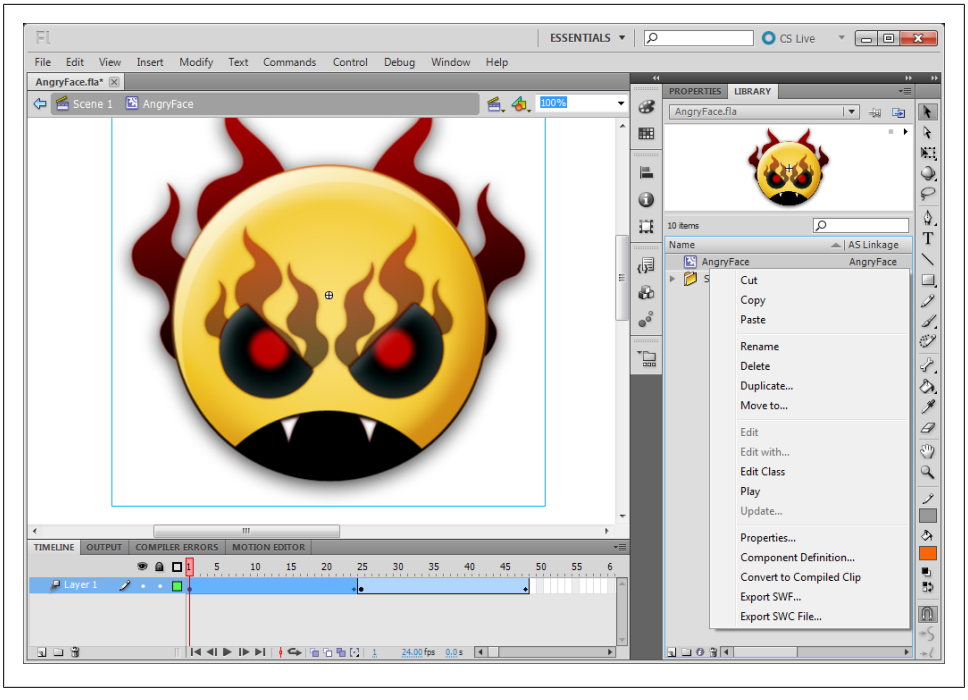


Figure 1-5. Export SWC from Flash Professional

Note that in this example, we are employing a `MovieClip` object that was animated in Flash Professional CS5.5, exported as part of a SWC, and linked into Flash Builder 4.5. There are other ways of doing this, but this method is very direct if you are not working within Flash Professional already.

CHAPTER 2

External Image Capabilities

With Flash Player's focused ability to readily handle vector drawing objects, it is often overlooked how capable the platform is at utilizing bitmap data through embedded or external image files. Whether using *PNG*, *JPG*, *GIF*, or the new *JPEG-XR* filetype, there is no denying that this imaging technology is extended and improved upon in some rather spectacular ways.

Enhanced High-Resolution Bitmap Support

Loaded `BitmapData` objects have historically been limited to 8,191 total pixels along any side with a total supported resolution of 16,777,215 pixels...which isn't a whole lot when dealing with high resolution images. With the megapixel count of consumer digital cameras breaking well past 10, the need for greater resolution is easily apparent. With Flash Player 11, these restrictions have been lifted, making this is a feature that can be leveraged through a multitude of project types.

1 megapixel is equal to **1,000,000** pixels.

Flash Player 10 supports up to **16.777** megapixels.

Flash Player 11 includes no such restrictions.

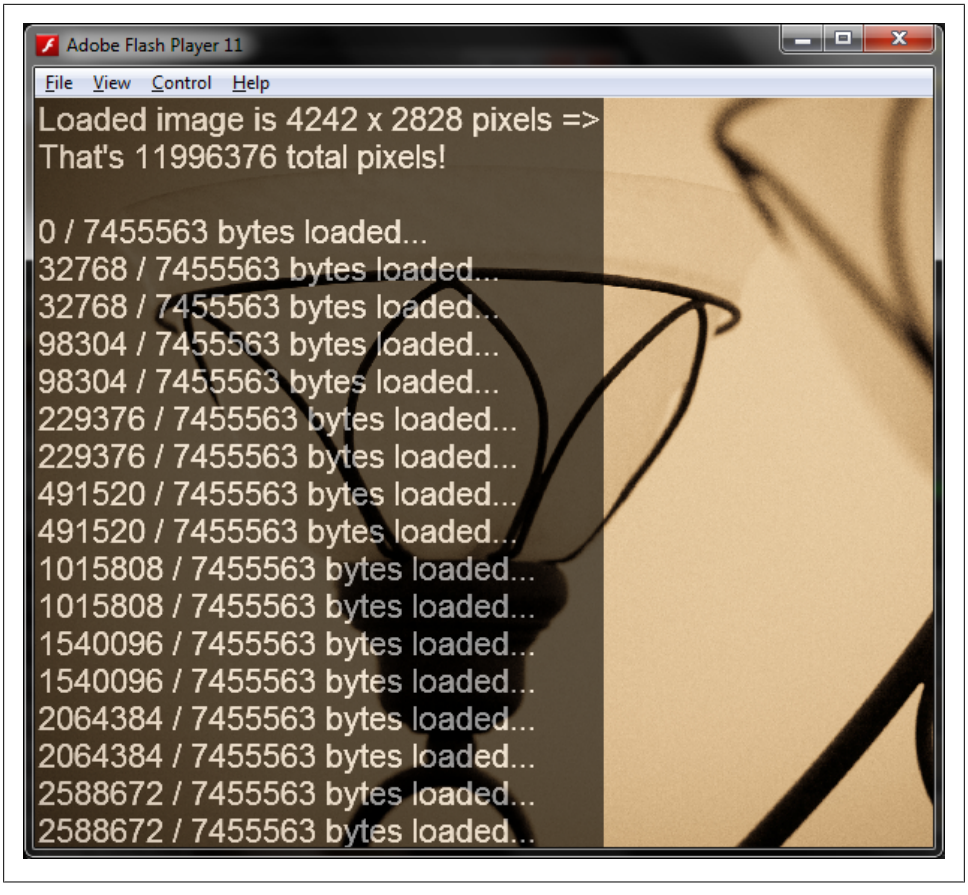


Figure 2-1. High-resolution bitmap

We don't actually need to do anything to enable support for this behavior, as it is built into Flash Player itself. In the following example, we'll use the `Loader` class to bring a high-resolution image into a Flash project:

```
package {
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.events.ProgressEvent;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class HighRes extends Sprite {

        private var imageLoader:Loader;
```

```
        private var traceField:TextField;

        public function HighRes() {
            generateDisplayObjects();
        }

        protected function generateDisplayObjects():void {
            imageLoader = new Loader();
          imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoaded);
            imageLoader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
imageProgress);
            imageLoader.load(new URLRequest("assets/highres.jpg"));
            addChild(imageLoader);

            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 22;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.6;
            traceField.autoSize = "left";
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function imageProgress(e:ProgressEvent):void {
            traceField.appendText(e.bytesLoaded + " / " + e.bytesTotal + " bytes
loaded...\n");
        }

        protected function imageLoaded(e:Event):void {
            imageLoader.height = stage.stageHeight;
            imageLoader.scaleX = imageLoader.scaleY;

            traceField.text = "Loaded image is " + e.target.width + " x " +
e.target.height + " pixels =>\nThat's " + e.target.width*e.target.height + " total
pixels!\n\n" + traceField.text;
        }

    }
}
```

Asynchronous Bitmap Decoding

When loading large images within Flash Player, we now have control over when the image is actually decoded. Previous to Flash Player 11, loading large images or other files could adversely impact performance and responsiveness of the general user interface. We can now offload this process to a separate thread and make some choices around the image decode process by using the `flash.system.ImageDecodingPolicy` class.

This is set as the `imageDecodingPolicy` property of the `flash.system.LoaderContext` class and has two potential values. These values are defined by the constants `ImageDecodingPolicy.ON_LOAD` and `ImageDecodingPolicy.ON_DEMAND`. The `ON_LOAD` setting will actually decode the image even before the complete event fires. If `ON_DEMAND` is specified as the developer's intended behavior, the image will not be decoded until it is needed by the runtime.

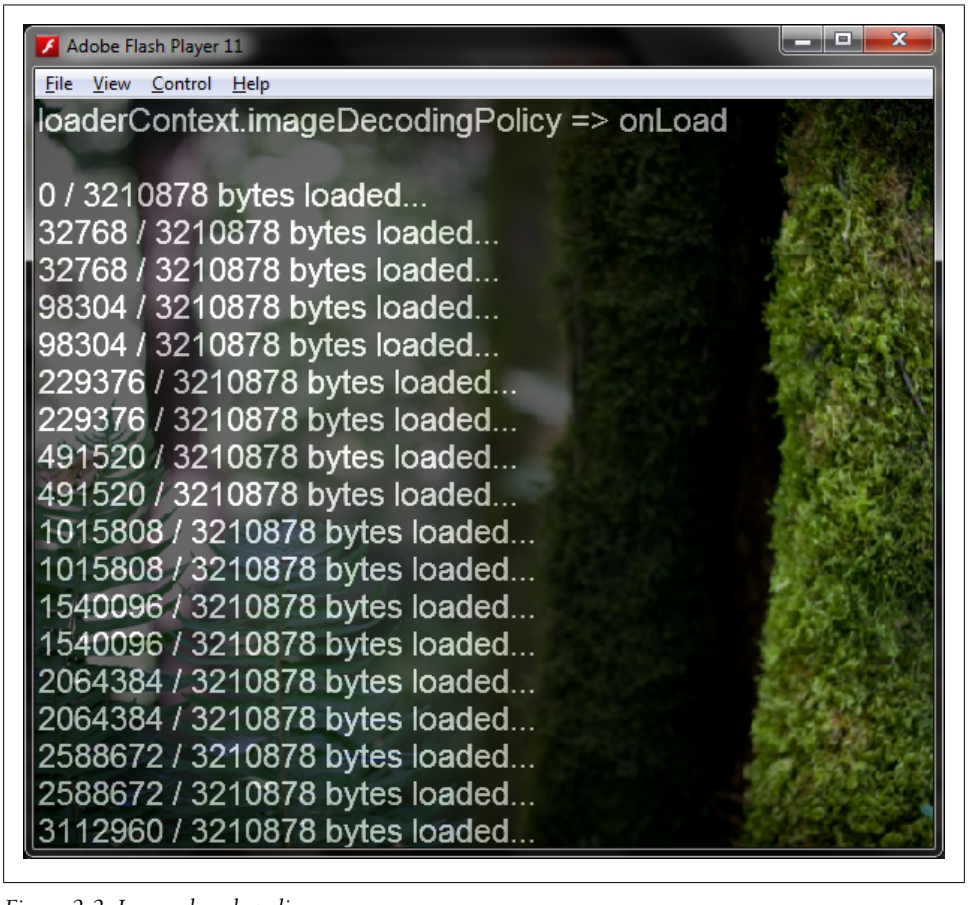


Figure 2-2. Image decode policy

In this example, we load a high-resolution image into a `Loader` class and decode using this new behavior.

```
package {
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.system.LoaderContext;
```

```
    import flash.system.ImageDecodingPolicy;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.events.ProgressEvent;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class ImageDecoding extends Sprite {

        private var imageLoader:Loader;
        private var loaderContext:LoaderContext;
        private var traceField:TextField;

        public function ImageDecoding() {
            generateDisplayObjects();
        }

        protected function generateDisplayObjects():void {
            loaderContext = new LoaderContext();

            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD;

            imageLoader = new Loader();
          imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoaded);
            imageLoader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
imageProgress);
            imageLoader.load(new URLRequest("assets/decode.jpg"), loaderContext);
            addChild(imageLoader);

            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 22;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.6;
            traceField.autoSize = "left";
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function imageProgress(e:ProgressEvent):void {
            traceField.appendText(e.bytesLoaded + " / " + e.bytesTotal + " bytes
loaded...\n");
        }

        protected function imageLoaded(e:Event):void {
            imageLoader.height = stage.stageHeight;
            imageLoader.scaleX = imageLoader.scaleY;

            traceField.text = "loaderContext.imageDecodingPolicy => " +
loaderContext.imageDecodingPolicy + "\n\n" + traceField.text;
        }
```

```
        }
    }
```

JPEG-XR Support

Flash Player 11 includes expanded support for still image file formats. Previous versions of Flash Player include support for the following image file formats: *GIF*, *JPEG*, and *PNG* – with any other files relying upon external code libraries for interpretation. The recent addition of *JPEG-XR* (International Standard ISO/IEC 29199-2) brings a new image file format to Flash Player which boasts more efficient compression than *JPG*, along with both lossy and lossless compression options. Like the *PNG* format, *JPEG-XR* also includes a full alpha channel.

You may be wondering how to generate *JPEG-XR* files, since many popular tools (including Adobe Photoshop) do not support the export or conversion to *.JXR* natively. I've found the Windows-only tool **Paint.NET** (*http://paint.net/*) along with the **JPEG XR plugin** (*http://pdnjpegxrplugin.codeplex.com/*) to be most useful in converting images to *JPEG-XR*.

Many conversion programs actually leave out certain bytes which are necessary for the file to load into the runtime, due to security concerns.

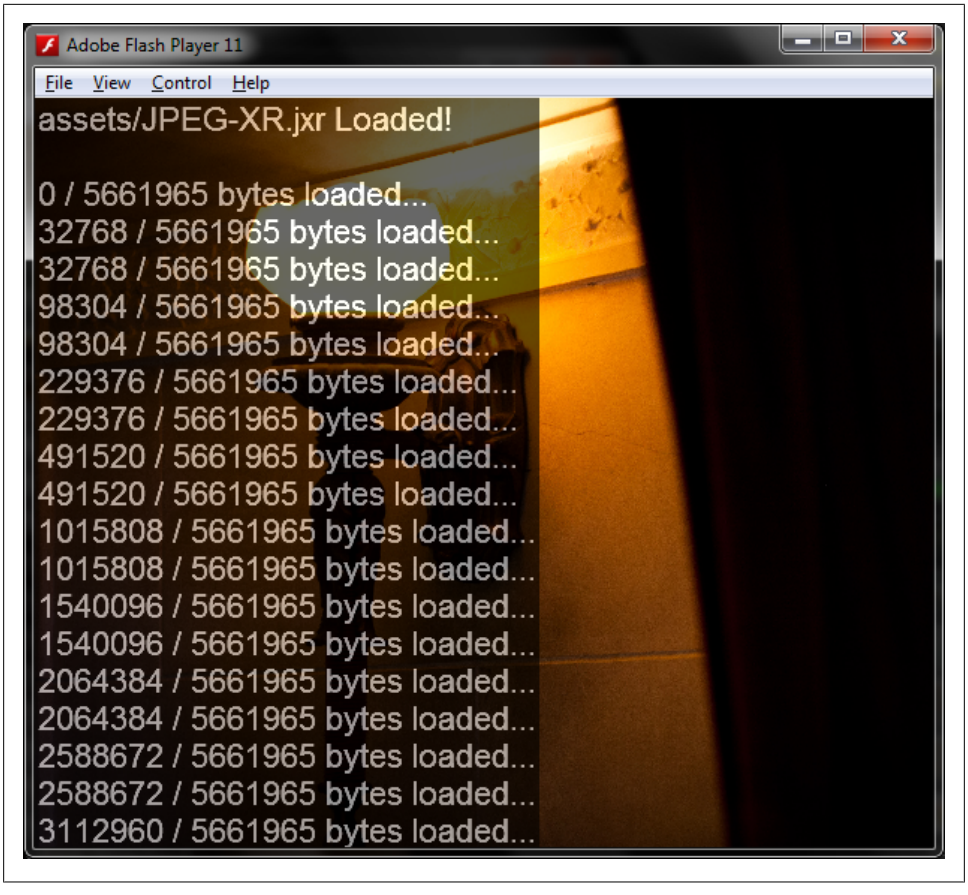


Figure 2-3. JPEG-XR support

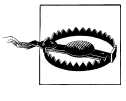
To load a *JPEG-XR* file into Flash Player, you perform the same set of actions that are necessary for any external image to be loaded into a project:

```
package {
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.ProgressEvent;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFormat;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class JPEGXR extends Sprite {

        private var imageLoader:Loader;
        private var traceField:TextField;
```

```
        private const JXR_PATH:String = "assets/JPEG-XR.jxr";

        public function JPEGXR() {
            generateDisplayObjects();
        }

        protected function generateDisplayObjects():void {
            imageLoader = new Loader();
          imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoaded);
            imageLoader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
imageProgress);
            imageLoader.load(new URLRequest(JXR_PATH));
            addChild(imageLoader);

            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 22;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.6;
            traceField.autoSize = "left";
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function imageProgress(e:ProgressEvent):void {
            traceField.appendText(e.bytesLoaded + " / " + e.bytesTotal + " bytes
loaded...\n");
        }

        protected function imageLoaded(e:Event):void {
            imageLoader.height = stage.stageHeight;
            imageLoader.scaleX = imageLoader.scaleY;

            traceField.text = JXR_PATH + " Loaded!\n\n" + traceField.text;
        }

    }
}
```

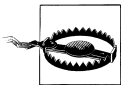
CHAPTER 3
Stage3D: High Performance Visuals

The single most written about feature of Flash Player 11 would definitely be the new accelerated graphics rendering engine available through *Stage3D* (previously known by the codename "*Molehill*"). This advanced rendering architecture can be used in rendering both 2D and 3D visual objects within Flash Player through direct use of the APIs or by implementation of one of the many engines and frameworks that have been built on top of these APIs.



To use `Stage3D` in Flash Player, we must set the `wmode` to `direct` if embedding within a web browser.

The main benefit of using the `Stage3D` APIs is that everything rendered using `Stage3D` on supported system configurations will be rendered directly through the system *GPU* (Graphics Processing Unit). This allows the GPU to assume total responsibility for these complex visual rendering tasks, while the *CPU* (Central Processing Unit) remains available for other functions.



In those cases where rendering `Stage3D` using hardware is not available on a particular system, the `Stage3D` view will be rendered using software as a fallback.

Stage3D Accelerated Graphics Rendering

The new `flash.display.Stage3D` class works very similar to `flash.media.StageVideo` in how it behaves as a display object within Flash Player. Just like `StageVideo`, `Stage3D` is never added to the Flash `DisplayList` but rather exists separately from that stack of objects. As in the case of `StageVideo` usage, the `DisplayList` appears above `Stage3D` in the visual stacking order.

It's important to note that `Stage3D` does not in any way deprecate or interfere with the "2.5D" capabilities introduced in Flash Player 10. Those APIs are used with objects added to the traditional `DisplayList`, while the new `Stage3D` APIs are entirely separated from that.

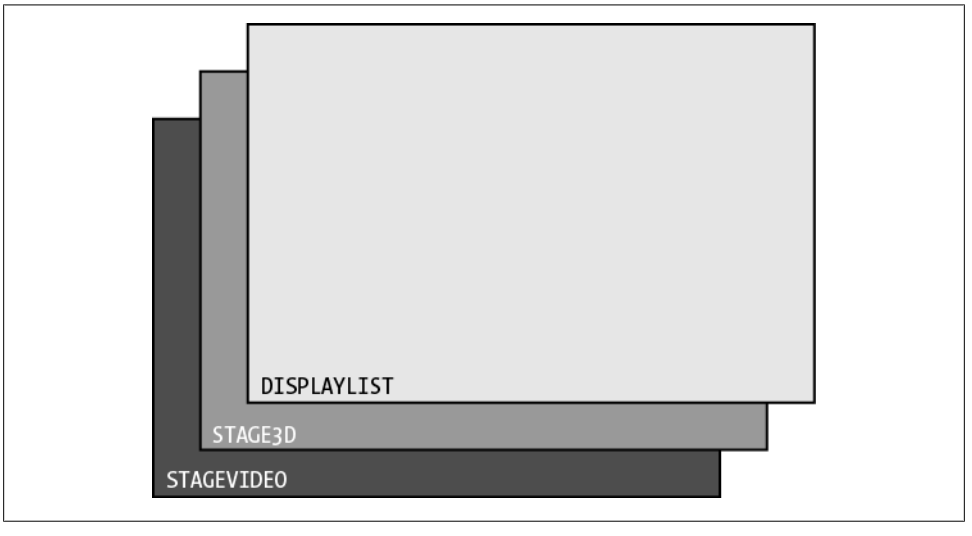
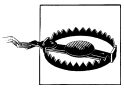


Figure 3-1. Stage3D sits between StageVideo and the traditional DisplayList

This will, no doubt remain one of the most deep and complex sets of classes that a Flash developer will come across for some years to come. Thankfully, Adobe has made the wise decision of providing early access to these new APIs to both rendering engine and tooling product creators.



`Stage3D` is currently supported on the desktop only. Mobile `Stage3D` will be supported in a future Flash Player release.

Elements of Stage3D

As mentioned above, `Stage3D` itself is rather low level in its implantation and quite difficult to work with for most ActionScript developers because of this. If you haven't worked in a 3D programming environment before, many of the terms and objects that are necessary to get this working will seem quite foreign in relation to your normal workflow.

For an example of how to leverage these raw APIs, we suggest that the reader visit Thibault Imbert's website at *http://www.bytearray.org/* for a number of `Stage3D` examples and a much deeper information pool than we will get into here.

To get a simple example of `Stage3D` set up and rendering within Flash Player, there are a number of core classes to import, as can be seen below:

```
import flash.display.Stage3D;
import flash.display3D.Context3D;
import flash.display3D.Context3DProgramType;
import flash.display3D.Context3DTriangleFace;
import flash.display3D.Context3DVertexBufferFormat;
import flash.display3D.IndexBuffer3D;
import flash.display3D.Program3D;
import flash.display3D.VertexBuffer3D;
import flash.geom.Matrix3D;
```

When working in `Stage3D`, we have to work with *vertex* and *fragment shaders* in order to render anything upon the `Stage3D` view. For those unfamiliar with the term, *shaders* are low-level software instructions that are used to calculate rendering effects on the system GPU. In fact, these instructions are used to directly program the graphics rendering pipeline or the GPU. *Vertex shaders* affect the direct appearance of a visual element while *fragment shaders* manage element surface details.

Adobe Pixel Bender 3D allows the production of `vertex` and `fragment shaders` that run on 3D hardware to generate output images. These kernels operate on 3D objects and affect their appearance.

To actually create and render any `shaders`, you'll also need to use a new language called *AGAL* (Adobe Graphics Assembly Language). *AGAL* is very, very low level and not for the faint of heart. Traditional Flash developers will most likely struggle with *AGAL*, but those familiar with working in other environments such as *OpenGL* or any general *Assembly* language should feel right at home. In either case, the recommended approach to working with `Stage3D` is to use one of the many higher-level frameworks that are available.

While `Stage3D` has a large number of 3D frameworks which utilize it in the creation and rendering of complex 3D graphics within Flash Player, the rendering surface can actually be used for any 3D or even 2D content which utilizes it in enabling an accelerated visual experience.

The basic setup for getting `Stage3D` working in an ActionScript project is to perform the following actions:

- Request a `Context3D` object through the `stage.stage3Ds` array.
- Once the `Context3D` object is ready, we can then set up `Context3D` to whatever specifications we have, including our `IndexBuffer3D` and `VertexBuffer3D` objects.
- We then use *AGAL* to create our various `shaders` to use within a `Program3D` object.
- Finally, all of this is processed through a render loop (`Event.ENTER_FRAME`) and rendered to the `Stage3D` object via `Context3D` and a set of `Program3D` and `Matrix3D` object controls.

If this sounds complicated, that's because it is! The process outlined above and the array of complexities associated with it are really meant for those who wish to build their own frameworks and engines upon a `Stage3D` foundation. In the next section, we'll have a look at how to actually use one of these 3D frameworks to render some content within Flash Player.

There is a project hosted on **Google Code** called **EasyAGAL** which aims to simplify the creation of *AGAL* for *Stage3D*. The project can be acquired from *http://code.google.com/p/easy-agal/*

Stage3D Example Using Away3D

Thankfully, we don't need to deal with direct APIs and *AGAL* unless we actually want to. There are a number of very robust, complete 3D frameworks that can be used as high-level alternatives to the Flash Player `Stage3D` APIs. In this example, we will have a look at a simple implementation using `Away3D` to render an animated primitive using `Stage3D`.

I would encourage those who are curious to perform a basic rendering like this using the direct APIs first, and then compare that with the `Away3D` implementation. The differences will be quite apparent in how simple a framework like `Away3D` distills the APIs into a highly usable form.

Before running the example below, you will want to be sure to download the proper Away3D framework code from *http://away3d.com/* for use within your project.

As can be seen in the code below, all we need to do for this to work is to create an instance of `View3D`, generate objects such as the `WireframeCube` primitive, and add these objects to the `View3D.scene` property. Now all we must do is render the `View3D`. This is normally done by creating what is known as a render loop using `Event.ENTER_FRAME` and then executing `View3D.render()` within a method invoked by that event. Upon every iteration of the render loop, we have the opportunity to adjust our object properties.

In our example, we adjust the `rotationX` and `rotationY` properties of our `WireframeCube` primitive to create 3D animation.

```
package {
    import away3d.containers.View3D;
    import away3d.primitives.WireframeCube;

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

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class SimpleAway3D extends Sprite {

        private var view3D:View3D;
        private var wireframeCube:WireframeCube;

        public function SimpleAway3D() {
            generate3D();
        }

        private function generate3D():void {
            var size:Number = 250;
            wireframeCube = new WireframeCube(size, size, size, 0x24ff00, 5);

            view3D = new View3D();
            view3D.scene.addChild(wireframeCube);

            addChild(view3D);
            addEventListener(Event.ENTER_FRAME, renderLoop);
        }

        protected function renderLoop(e:Event):void {
            wireframeCube.rotationX += 1;
            wireframeCube.rotationY += 3;
            view3D.render();
        }

    }
}
```

Running the above code will produce a wireframe cube slowing rotating along the `x` and `y` axis. `Away3D` comes packaged with a lot of different primitives and materials that can be used in rendering 3D content. This example just scratches the surface of what one might do with such an extensive framework.

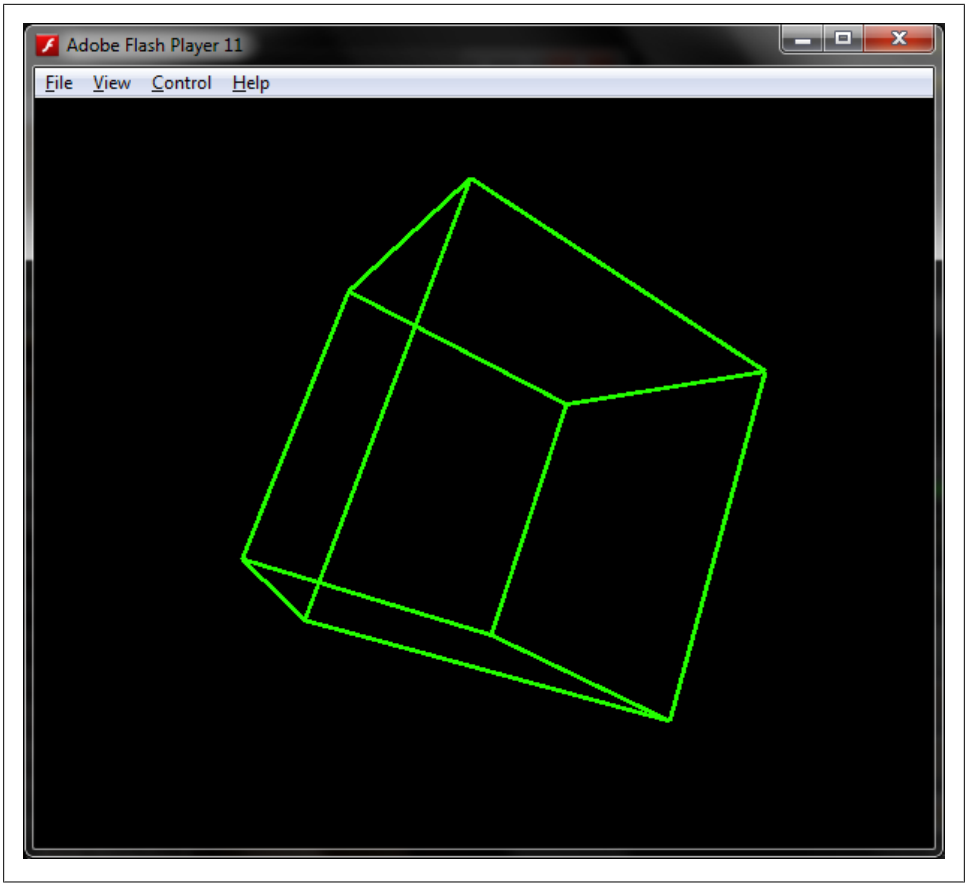


Figure 3-2. WireFrameCube primitive rendered using Away3D

`Away3D` is just one of many ActionScript frameworks which utilize `Stage3D`. These frameworks are meant to provide high-level access to powerful display technology, and each has its strengths and weaknesses. Experiment with a number of these frameworks to discover what will work best in your particular project.

A list of `Stage3D` frameworks and libraries is included in Appendix of this book.

Stage3D Example Using Starling

Starling (*http://starling-framework.org/*) is an open source effort begun by **Adobe** and the **Sparrow Framework** (*http://www.sparrow-framework.org/*) to create a 2D frame-

work for `Stage3D` which emulates the traditional `DisplayList` that Flash Platform developers are so used to. In fact, developers can use concepts that they are familiar with such as `Sprite`, `MovieClip`, and `TextField` in a very similar way to how these objects would be used with native Flash and AIR classes.

Starling is a direct port of the Sparrow framework for iOS which mimics the Flash `DisplayList` APIs.

The Starling framework can be freely acquired from *http://github.com/PrimaryFeather/Starling---Framework/* and weighs in at only 80k – very lightweight. Since it is an open source project, the community can contribute and help grow the framework.

In this quick example, we will create a simple *Quad* and cause it to continuously rotate clockwise. First, we must set up our Starling classes through the main application class. The important thing here is that we create a new instance of `starling.core.Starling` and pass in a class called `Game` which will contain the remainder of our code. We also pass in a reference to the current `Stage`. The final step is to invoke `Starling.start()` to get things going.

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

    import starling.core.Starling;

    [SWF(width="600", height="500", backgroundColor="#000000")]

    public class SimpleStarling extends Sprite {

        private var starlingBase:Starling;

        public function SimpleStarling() {
            super();
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;

            performOperations();
        }

        protected function performOperations():void {
            starlingBase = new Starling(Game, this.stage);
            starlingBase.antiAliasing = 2;
            starlingBase.start();
        }

    }

}
```

Now that we have set up Starling, we have to create the `Game` class which it uses upon initialization. All of our rendering will live inside of this `Game.as` class included within the same package as our main application class in this example.

Initially, we want to be sure that our class is added to the `Stage` and ready to perform display functions for us. To do this, we add an event listener of type `Event.ADDED_TO_STAGE`. Once this event fires, we are safe to begin drawing out our visual objects using Starling classes.

Note that even though we are using familiar classes like `Sprite` and `Event`, we are using the Starling versions of these classes — not the core Flash classes.

Here, we now set up our *Quad*. A quad is basically two triangles which link together to form a square plane. We will set this up in such a way that its position is at the center of the `Stage` with a transform point (pivot) at its center. This will allow us to rotate around the center point instead of the upper left which is default. Using `Quad.setVertexColor()`, we set different shades of green as gradient points.

Finally, we set up the render loop which is invoked through `Event.ENTER_FRAME`. This is where any change over time should occur, and in this case it does a simple clockwise rotation of the *Quad*.

```
package {
    import starling.display.Sprite;
    import starling.display.Quad;
    import starling.events.Event;

    public class Game extends Sprite {

        private var quad:Quad;

        public function Game() {
            this.addEventListener(Event.ADDED_TO_STAGE, onStageReady);
        }

        protected function onStageReady(e:Event):void {
            quad = new Quad(300, 300);
            quad.pivotX = 150;
            quad.pivotY = 150;
            quad.setVertexColor(0, 0x00ff18);
            quad.setVertexColor(1, 0x2dcb3b);
            quad.setVertexColor(2, 0x00ff18);
            quad.setVertexColor(3, 0x2dcb3b);
            quad.x = (stage.stageWidth/2);
            quad.y = (stage.stageHeight/2);
            this.addChild(quad);
            this.addEventListener(Event.ENTER_FRAME, renderLoop);
        }
```

```
            protected function renderLoop(e:Event):void {
                quad.rotation += 0.02;
            }

        }

    }
```

When we compile and run this code on the desktop, we can see how simple using accelerated 2D graphics with `Stage3D` can be thanks to this fabulous framework.

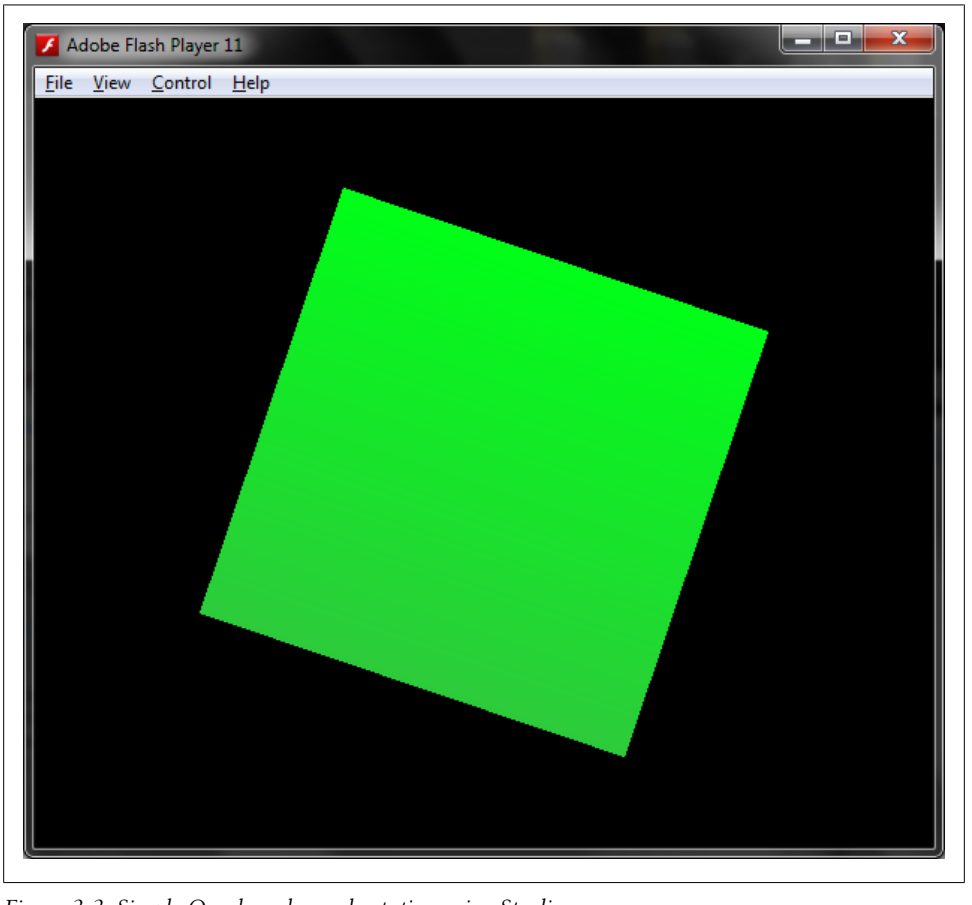


Figure 3-3. Simple Quad render and rotation using Starling

Read all about the Starling framework in Thibault Imbert's book: **Introducing Starling** [*http://byearray.org/*] – just like Starling itself, this book is free!

Tooling Support for Stage3D

Not only does `Stage3D` have the support of many 3D frameworks, but a variety of tooling products have also embraced this new functionality. Most notable of these, is the *Unity* development environment.

Unity

Unity has built-in support for `Stage3D`, going so far as to export directly to a compiled *SWF* which can be nearly identical to an export to the Unity, depending upon supported features. These features include *physics*, *lightmapping*, *occlusion culling*, *custom shaders*, *lightprobes*, *particle systems*, *navigation meshes*, and more! This is truly an incredible development where Flash and AIR gaming is concerned, as Unity is such a great gaming engine and editor environment, already in use by many game developers targeting a variety of diverse platforms.

After rendering Unity content for Flash Player, developers should be able to build upon that content within larger Flash Player projects. One use for this would be to create a robust menuing system for a game.

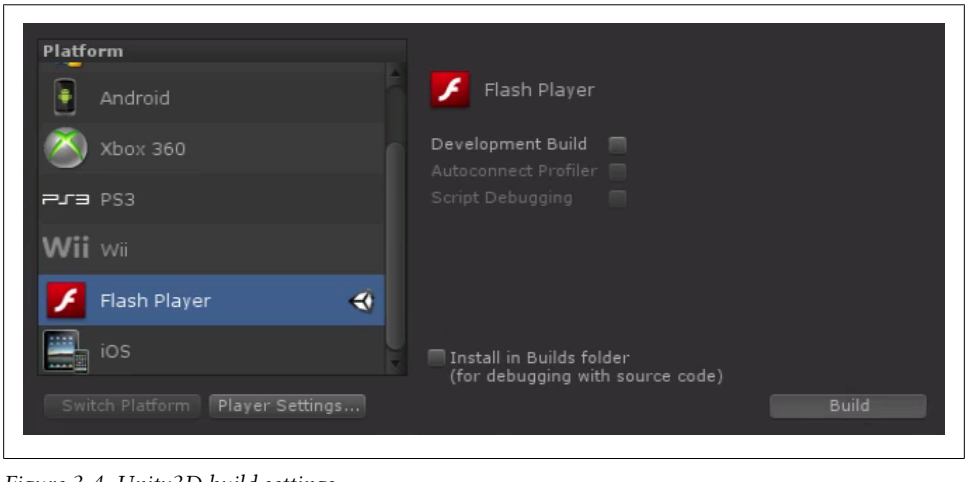


Figure 3-4. Unity3D build settings

Flare3D Studio

Also of note is Flare3D Studio – a 3D design environment build using Flash Platform tooling and distributed using the AIR runtime! It is excellent to see such excitement and collaboration in the industry around `Stage3D` from all of these different players.

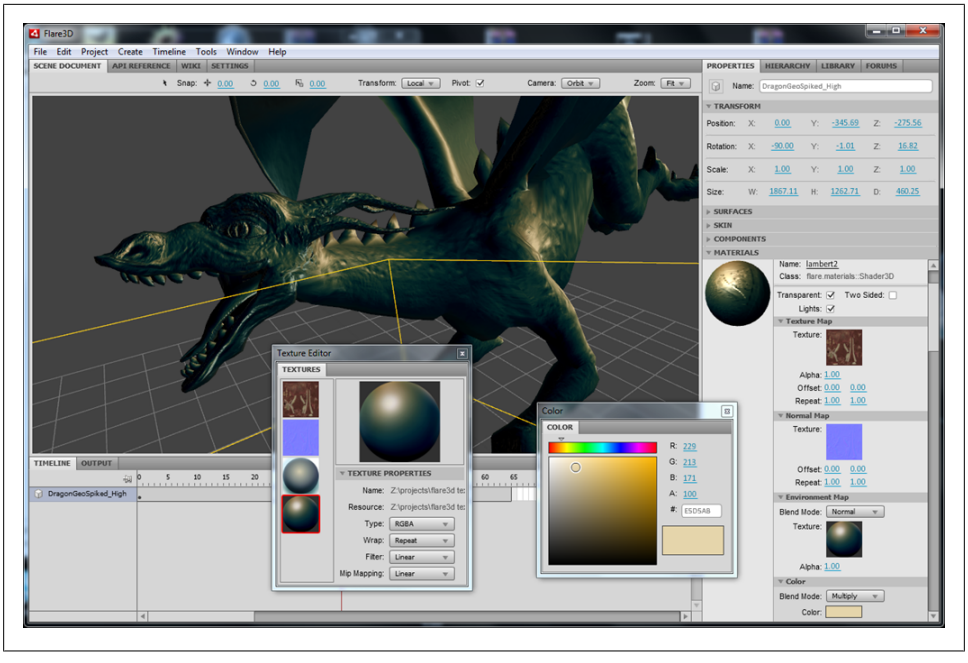


Figure 3-5. Flare3D Studio

I gather that we have much to look forward to in terms of improved tooling from Adobe, Unity, and perhaps other parties as well.

CHAPTER 4
Audio and Video Enhancements

Since the *YouTube* video revolution of 2006, Flash has been the premiere method of video delivery through web browsers, avoiding the fragmentation that otherwise persisted as a result of warring codecs and players. The `VP6` Flash Video (`FLV`) format standardized most video playback over the Web in alignment with Flash Player. Over the years, different container formats and codecs have been added to the Flash Player to keep up with industry trends, including `H.264` decoding with Flash Player 9. With Flash Player 11, we have a set of new enhancements to take advantage of.

H.264/AVC Software Encoding for Cameras

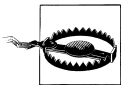
With Flash Player 11, developers now have the ability to encode `H.264` video streams within the Flash Player itself. Prior versions of the runtime were able to decode `H.264` and with the additional encoding functionality, a whole other set of applications can be built to record and broadcast in this industry standard format.

H.264/AVC software encoding is only available on the desktop. Mobile devices cannot utilize this feature due to the amount of CPU it takes to encode the streams.

To compile and run the examples included below effectively, it is recommended that you install Flash Media Server.

Note that if you do not have access to a commercial license for Flash Media Server, Adobe does offer a free developer edition with which you can test this and other examples. When doing any serious work through FMS, it is encouraged that you begin with a local development instance such as this.

Flash Media Server developer edition can be acquired from Adobe via *http://www.adobe.com/products/flashmediaserver/*



Flash Media Server is available for either Windows or Linux.

Encoding H.264 within Flash Player 11

To encode a video stream using `H.264` within Flash Player 11, we must employ the new `H264VideoStreamSettings` class and associated objects. When constructing a `H264VideoStreamSettings` instance for use in our project, we can set the particular profile and level of the encoding. This is useful for targeting certain specific devices which may only support something like baseline encoding. Once we have configured our `H264VideoStreamSettings` instance, we assign it to the `videoStreamSettings` property of a `NetStream` instance and then process the stream as normal.

```
package {
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.media.Camera;
    import flash.media.H264Level;
    import flash.media.H264Profile;
    import flash.media.H264VideoStreamSettings;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.text.TextField;
    import flash.text.TextFormat;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class AVCEncode extends Sprite {

        private var traceField:TextField;
        private var video:Video;
        private var camera:Camera;
        private var connection:NetConnection;
        private var stream:NetStream;
        private var streamClient:Object;

        public function AVCEncode() {
            generateDisplayObjects();
            performOperations();
        }

        protected function generateDisplayObjects():void {
            video = new Video(stage.stageWidth, stage.stageHeight);
            addChild(video);

            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 24;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
```

```
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.autoSize = "left";
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            camera = Camera.getCamera();
            camera.setMode(stage.stageWidth, stage.stageHeight, 30);
            camera.setQuality(60000, 80);
            video.attachCamera(camera);

            streamClient = new Object();
            streamClient.onBWDone = onBWDone;

            connection = new NetConnection();
            connection.client = streamClient;
            connection.addEventListener(NetStatusEvent.NET_STATUS, monitorStatus);
            connection.connect("rtmp://localhost/live");
        }

        protected function monitorStatus(e:NetStatusEvent):void {
            traceField.appendText(e.info.code + "\n");
            if(e.info.code == "NetConnection.Connect.Success"){
                beginStreaming();
            }else if(e.info.code == "NetStream.Publish.Start"){
                traceField.appendText("\n" + e.info.description + "\n");
                traceField.appendText("codec: " + stream.videoStreamSettings.codec);
            }
        }

        protected function beginStreaming():void {
            var h264Settings:H264VideoStreamSettings = new H264VideoStreamSettings();
            h264Settings.setProfileLevel(H264Profile.BASELINE, H264Level.LEVEL_2);

            stream = new NetStream(connection);
            stream.addEventListener(NetStatusEvent.NET_STATUS, monitorStatus);
            stream.videoStreamSettings = h264Settings;
            stream.attachCamera(camera);
            stream.publish("mp4:h264livestream.f4v", "live");
        }

        public function onBWDone():void {}

    }
}
```

So long as we have Flash Media Server installed and running on our local machine, we will receive a message stating that the stream is being published and that the codec being used to encode the video is `H.264`, along with a preview of the camera feed. In this example, we are simply viewing the raw camera output and not the encoded stream.

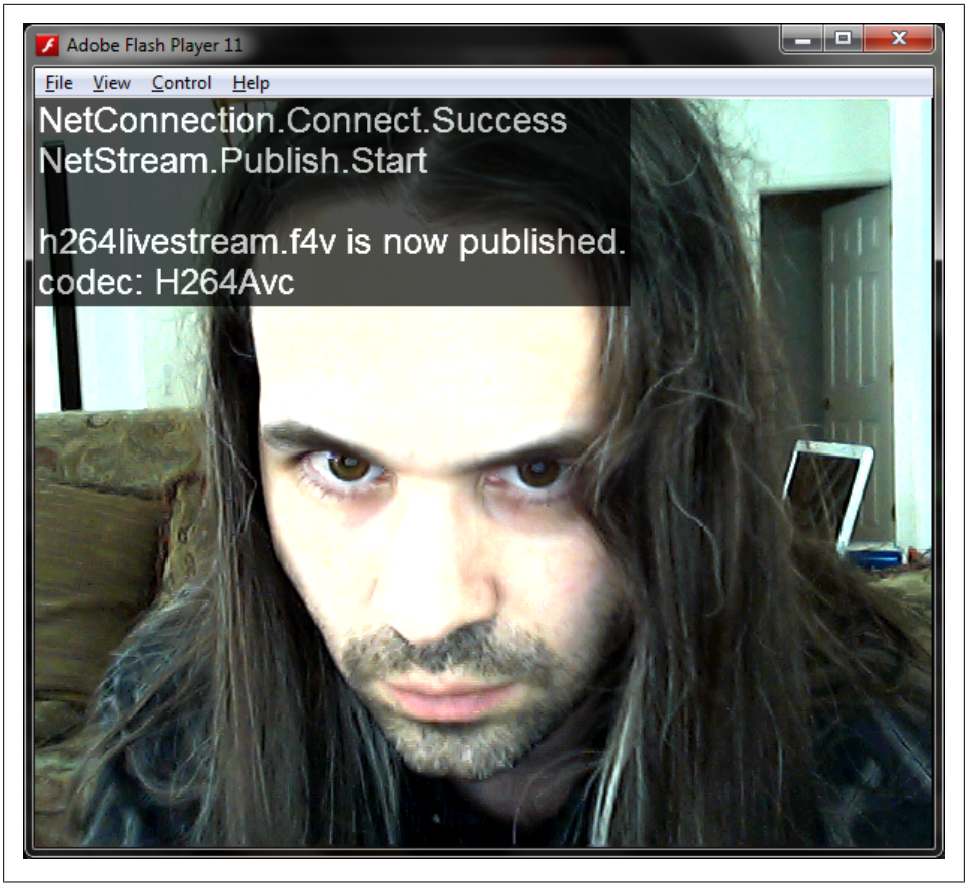


Figure 4-1. H.264 Encoding within Flash Player

Reading an H.264 Stream into Flash Player 11

The ability to decode `H.264` has been available since Flash Player 9. To play back a stream or file encoded to `H.264` with Flash Player 11 is the same procedure we would normally use. First, create a `NetConnection` and establish a connection to Flash Media Server. In this case, we use *rtmp://localhost/live*, as the server exists on the same machine. We then hook in a `NetStream` object and request to play the previously published stream.

```
package {
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.text.TextField;
```

```
import flash.text.TextFormat;

[SWF(width="600", height="500", backgroundColor="#CCCCCC")]

public class AVCPlayback extends Sprite {

    private var traceField:TextField;
    private var video:Video;
    private var connection:NetConnection;
    private var stream:NetStream;
    private var streamClient:Object;

    public function AVCPlayback() {
        generateDisplayObjects();
        performOperations();
    }

    protected function generateDisplayObjects():void {
        video = new Video(stage.stageWidth, stage.stageHeight);
        addChild(video);

        var defaultFormat:TextFormat = new TextFormat();
        defaultFormat.font = "Arial";
        defaultFormat.size = 24;
        defaultFormat.color = 0xFFFFFF;

        traceField = new TextField();
        traceField.backgroundColor = 0x000000;
        traceField.alpha = 0.7;
        traceField.autoSize = "left";
        traceField.background = true;
        traceField.defaultTextFormat = defaultFormat;
        addChild(traceField);
    }

    protected function performOperations():void {
        streamClient = new Object();
        streamClient.onBWDone = onBWDone;

        connection = new NetConnection();
        connection.client = streamClient;
        connection.addEventListener(NetStatusEvent.NET_STATUS, monitorStatus);
        connection.connect("rtmp://localhost/live");
    }

    protected function monitorStatus(e:NetStatusEvent):void {
        traceField.appendText(e.info.code + "\n");
        if(e.info.code == "NetConnection.Connect.Success"){
            beginStreaming();
        }
    }

    protected function beginStreaming():void {
        stream = new NetStream(connection);
        stream.addEventListener(NetStatusEvent.NET_STATUS, monitorStatus);
```

```
            stream.play("mp4:h264livestream.f4v");
            video.attachNetStream(stream);
        }

        public function onBWDone():void {}
    }
}
```

When this class is compiled to `SWF`, so long as we have the encoded stream successfully published to Flash Media Server through the previous code example, we will see the published, natively encoded `H.264` stream render through a `Video` object similar to Figure 4-2.

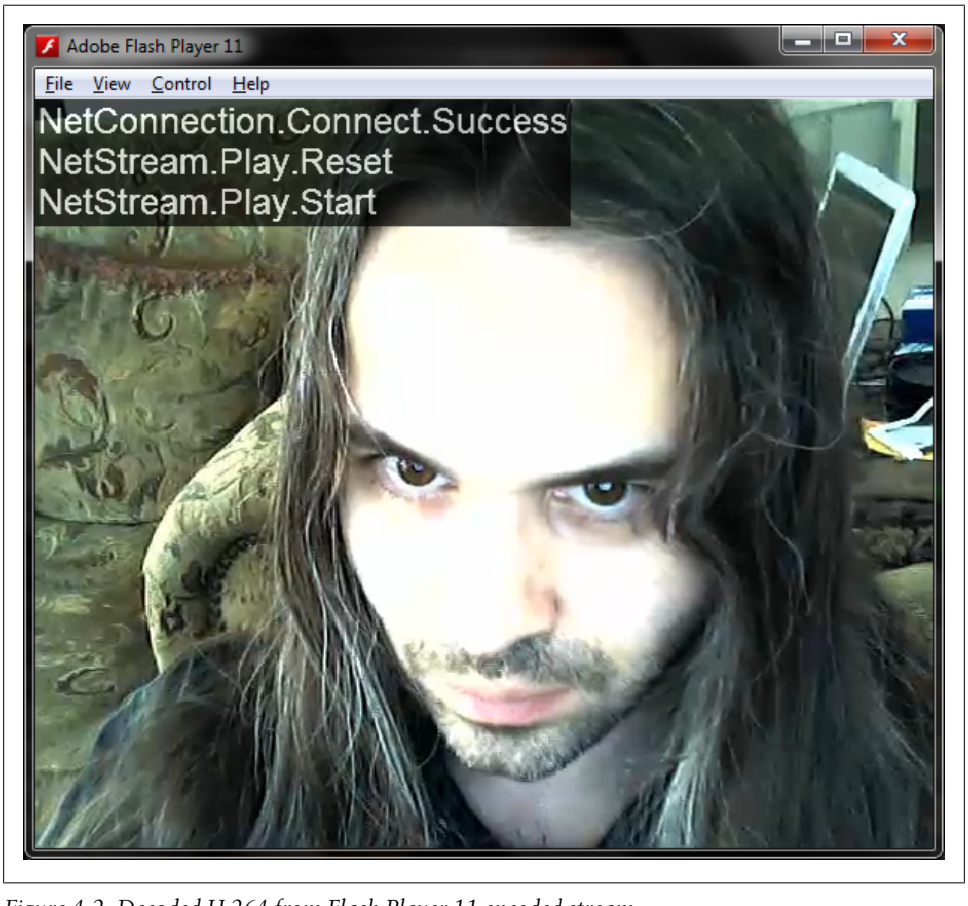


Figure 4-2. Decoded H.264 from Flash Player 11 encoded stream

G.711 Audio Compression for Telephony

Flash Player 11 includes support for the `G.711` codec for audio. `G.711` is actually a rather old – actually formally standardized in 1972 as "Pulse Code Modulation (*PCM*) of voice frequencies". There are two compression algorithm variants of `G.711`, both of which can be used in Flash Player:

`SoundCodec.PCMA`
: Specifies the G.711 A-law codec (used in Europe and the rest of the world).

`SoundCodec.PCMU`
: Specifies the G.711 μ-law codec (used in North America and Japan).

`G.711` is primarily used in telephony and `SIP` (Session Initiation Protocol) based applications. It is tailored specifically for voice communications and supported on innumerable systems.

We can accomplish this through use of the `flash.media.SoundCodec` class within our audio project. Upon configuration of our `Microphone` object, we can assign the codec property to the `SoundCodec.PCMU` or `SoundCodec.PCMA` constant. This will ensure that any audio from that source is processed as `G.711`.

```
package {
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.events.SampleDataEvent;
    import flash.filters.BlurFilter;
    import flash.geom.Point;
    import flash.media.Microphone;
    import flash.media.SoundCodec;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.ByteArray;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class G711Telephony extends Sprite {

        private const SPECTRUM_COLOR:uint = 0x24ff00;

        private var traceField:TextField;
        private var microphone:Microphone;
        private var audioBlur:BlurFilter;
        private var audioBitmapData:BitmapData;
        private var audioBitmap:Bitmap;
        private var soundDisplay:Sprite;
        private var soundActivity:Sprite;

        public function G711Telephony() {
```

```
            generateTextFields();
            spectrumSetup();
            performOperations();
        }

        protected function generateTextFields():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 24;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.multiline = true;
            traceField.wordWrap = true;
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function spectrumSetup():void {
           audioBitmapData = new BitmapData(stage.stageWidth, stage.stageWidth, true,
0x000000);
            audioBitmap = new Bitmap(audioBitmapData);

            audioBlur = new BlurFilter(2, 12, 2);

            soundActivity = new Sprite();
            soundDisplay = new Sprite();
            soundActivity.addChild(soundDisplay);
            soundActivity.addChild(audioBitmap);
            addChild(soundActivity);
        }

        protected function performOperations():void {
            if(Microphone.isSupported){
                microphone = Microphone.getMicrophone(0);
                microphone.codec = SoundCodec.PCMU;
                microphone.addEventListener(SampleDataEvent.SAMPLE_DATA,
microphoneDataSample);
            }
        }

        protected function microphoneDataSample(e:SampleDataEvent):void {
            var soundBytes:ByteArray = new ByteArray();
            soundBytes = e.data;

            traceField.text = "Microphone: " + e.target.name + "\n\n";
            traceField.appendText("activityLevel: " + e.target.activityLevel + "\n");
            traceField.appendText("bytesAvailable: " + soundBytes.bytesAvailable +
"\n");
            traceField.appendText("codec: " + e.target.codec);
```

```
            drawSpectrum(e.data);
        }

        protected function drawSpectrum(d:ByteArray):void {
            var ba:ByteArray = new ByteArray();
            ba = d;
            var a:Number = 0;
            var n:Number = 0;
            var i:uint = 0;
            soundDisplay.graphics.clear();
            soundDisplay.graphics.lineStyle(2, SPECTRUM_COLOR, 0.8, false);
            soundDisplay.graphics.moveTo(0, (n/2)+(stage.stageHeight/2+100));
            for(i=0; i<=ba.bytesAvailable; i++) {
                a = ba.readFloat();
                n = a*soundActivity.height;
                soundDisplay.graphics.lineTo(i*(stage.stageWidth/ba.bytesAvailable),
(n/2)+(stage.stageHeight/2+100));
            }
            soundDisplay.graphics.endFill();
            audioBitmapData.draw(soundDisplay);
            audioBitmapData.applyFilter(audioBitmapData, audioBitmapData.rect, new
Point(0,0), audioBlur);
        }
    }
}
```

The result of running this class can be seen in Figure 4-3.

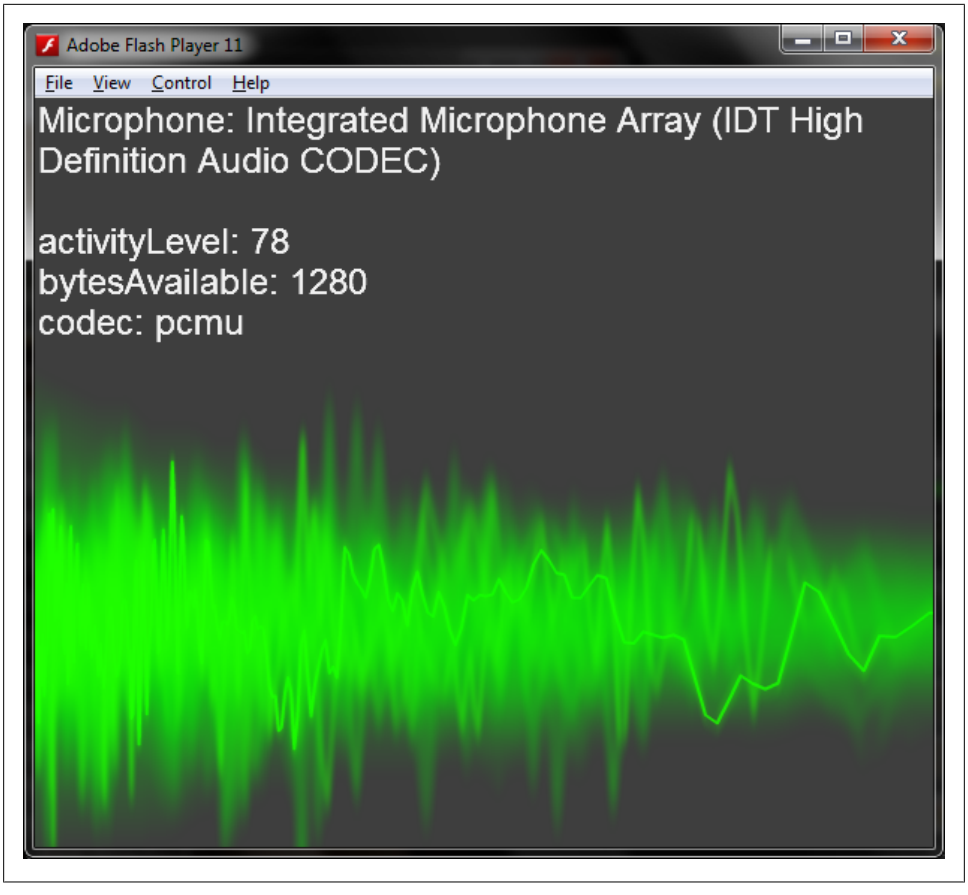


Figure 4-3. G.711 μ-law (PCMU) encoded audio data

CHAPTER 5
Data Transfer Additions

A robust application development platform needs a robust set of data transfer options. With the introduction of root level `XML` support in Flash Player 9, developers could take advantage of any `XML`-based format within their applications. Flash Player 11 adds support for a native `JSON` handler and some great enhancements when working with sockets in communicating with various systems.

Native JSON (JavaScript Object Notation) Support

JavaScript Object Notation (*JSON*) is a hugely popular way of transporting structured data sets into and out of applications that run within Flash Player. Ever since ActionScript 3.0 was introduced, there have been third-party support libraries which allowed developers to use JSON in their projects quite easily; however, this is costly in terms of performance, since it was never a core function of Flash Player itself.

`JSON` is a top level class, similar to the `XML` or `Array` classes present in ActionScript. As such, they do not need to be imported in order to be used within an application.

The following `JSON` object describes a person through a series of name value pairs. Notice that objects can be nested and that this syntax can even include array structures. It is incredibly flexible.

```
{
     "firstName": "Joseph",
     "lastName": "Labrecque",
     "address":
     {
         "streetAddress": "2199 S. University Blvd.",
         "city": "Denver",
         "state": "CO",
         "postalCode": "80208"
     },
```

```
        "phoneNumber":
        [
            {
              "type": "work",
              "number": "303.871.6566"
            },
            {
              "type": "fax",
              "number": "303.871.7445"
            }
        ]
}
```

JSON.parse()

We use this JSON file in the following code example to parse the values from the loaded `JSON` object using `JSON.parse()`, and then format the values within a basic `TextField` within our `SWF`.

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFormat;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class ReadJSON extends Sprite {

        private var traceField:TextField;
        private var json:URLLoader;
        private var parsedJSON:Object;

        public function ReadJSON() {
            generateDisplayObjects();
            performOperations();
        }

        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 26;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.wordWrap = true;
            traceField.multiline = true;
            traceField.background = true;
```

```
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            json = new URLLoader();
            json.addEventListener(Event.COMPLETE, parseJSON);
            json.load(new URLRequest("assets/data.json"));

            traceField.appendText("Loading JSON file...\n");
        }

        protected function parseJSON(e:Event):void {
            traceField.appendText("JSON file loaded successfully!\n");
            traceField.appendText("Parsing JSON...\n\n");
            traceField.appendText("RESULTS:\n");

            parsedJSON = JSON.parse(json.data);

            traceField.appendText("firstName: " + parsedJSON.firstName + "\n");
            traceField.appendText("lastName: " + parsedJSON.lastName + "\n");
            traceField.appendText("address.streetAddress: " +
parsedJSON.address.streetAddress + "\n");
            traceField.appendText("address.city: " + parsedJSON.address.city + "\n");
           traceField.appendText("address.state: " + parsedJSON.address.state + "\n");
            traceField.appendText("address.postalCode: " +
parsedJSON.address.postalCode + "\n");
            for(var i:int = 0; i<parsedJSON.phoneNumber.length; i++){
                traceField.appendText(parsedJSON.phoneNumber[i].type + ": " +
parsedJSON.phoneNumber[i].number + "\n");
            }
        }

    }
}
```

As you can see, dealing with `JSON` is very similar to dealing with `XML` within ActionScript. Parsing our imported `JSON` and outputting the data into a `TextField` renders similar to the following figure.

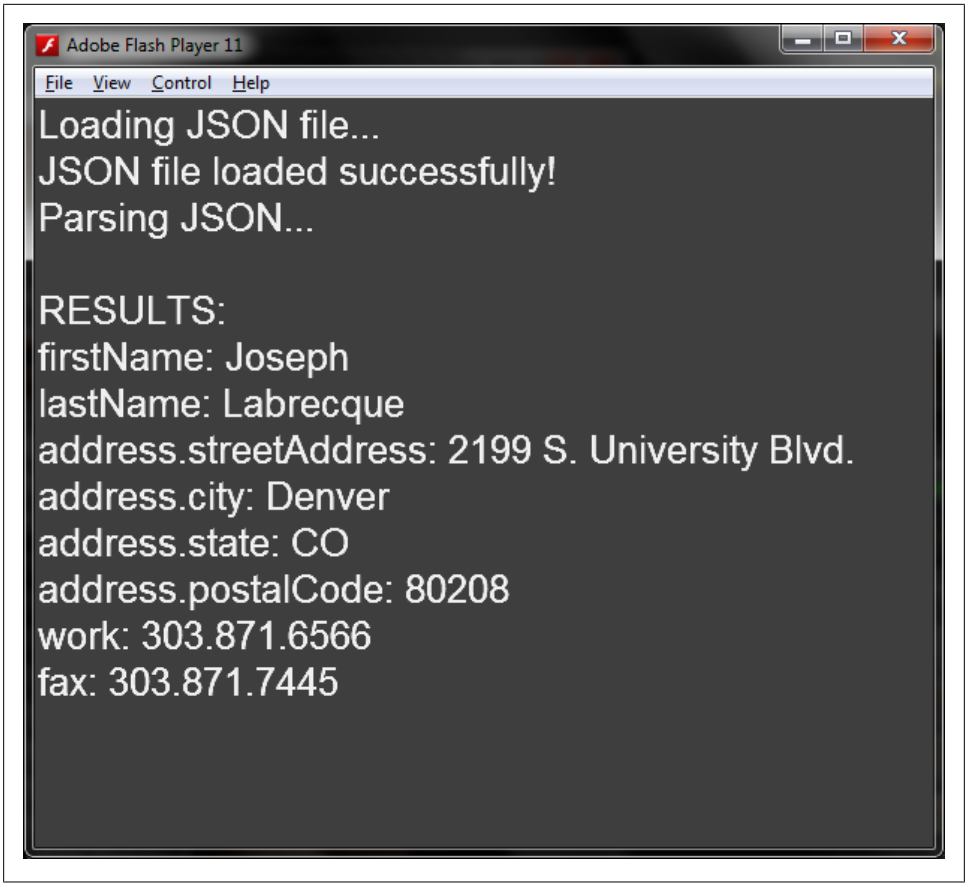


Figure 5-1. Parsed JSON output

JSON.stringify()

To actually write `JSON` from within Flash Player, we need to employ the `JSON.stringify()` method as demonstrated in the following code example.

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

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class WriteJSON extends Sprite {

        private var traceField:TextField;
        private var jsonObject:Object;
```

```
        public function WriteJSON() {
            generateDisplayObjects();
            performOperations();
        }

        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 26;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.wordWrap = true;
            traceField.multiline = true;
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            traceField.appendText("Forming Object in ActionScript...\n");

            jsonObject = new Object();
            jsonObject.firstName = "Edgar";
            jsonObject.middleName = "Allan";
            jsonObject.lastName = "Poe";
            jsonObject.birthDate = 1809;
            jsonObject.deathDate = 1849;
            jsonObject.nationality = "American";
            jsonObject.birthPlace = "Boston, Massachusetts";

            traceField.appendText("Stringify in progress...\n\n");

            var newJSON:Object = JSON.stringify(jsonObject, null, 4);

            traceField.appendText("RESULT:\n");
            traceField.appendText(newJSON.toString());
        }

    }
}
```

The `JSON.stringify()` method will convert the ActionScript object we've assembled into complete `JSON` syntax. This assumes that all data types within the object are of a type that can be converted into `JSON`.

Valid data types include: `Array`, `String`, `Number`, `Boolean`, and `null`.

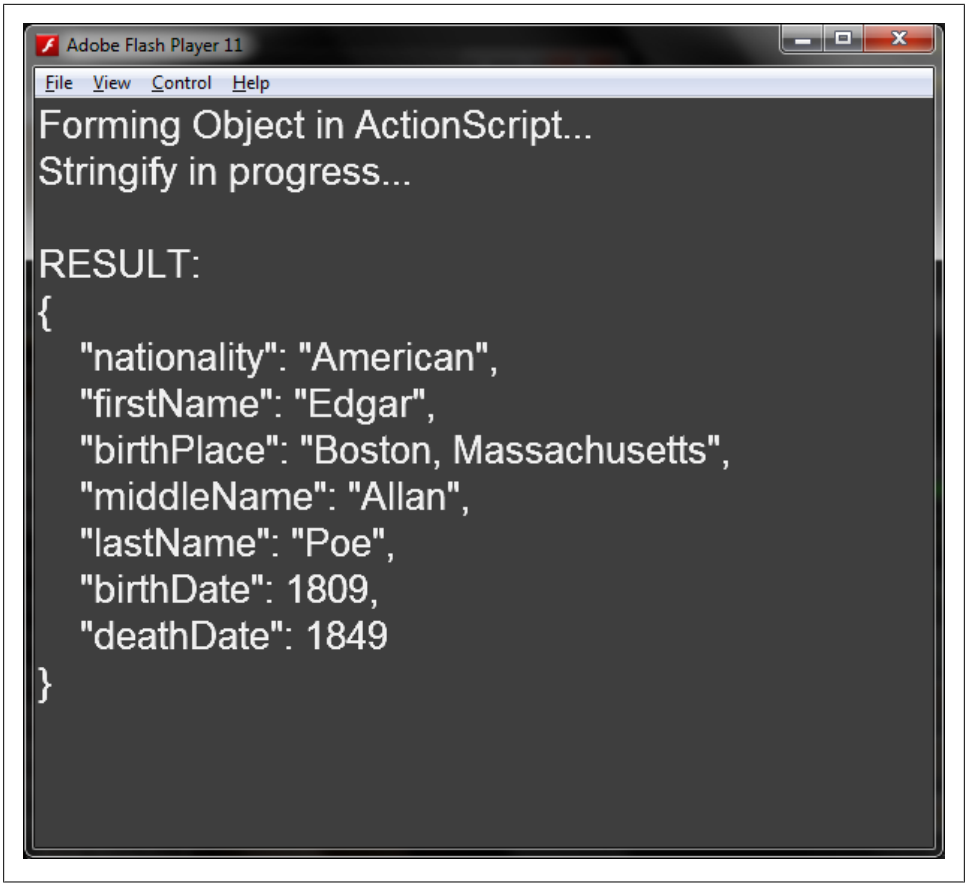


Figure 5-2. Stringified object output

The first argument that we pass into this method is the actual object which we want to convert. The second argument is an optional replacer function (or array) that can be used to transform or filter the key/value pairs in the object to be converted. This could be used in case we want to exclude certain key/value pairs, for instance, from the actual output.

The final argument specifies the amount of spaces to insert before each piece of data in order to make it more human-readable. In this example, we are passing in the number 4, specifying that the method should prepend 4 whitespace characters before each entry. This is how we achieve the spacing and readability in the figure below.

Figure 5-2 demonstrates the stringified output from our example.

Socket Progress Events

The `flash.net.Socket` class has been around for about 5 years now, beginning with Flash Player 9. When using sockets in ActionScript, developers have always had access to an event to monitor the progress of input data coming through the socket by using `flash.events.ProgressEvent`. Up until now, however, monitoring the output data being sent across a socket connection has been nearly impossible.

With Flash Player 11, we now have access to the `flash.events.OutputProgressEvent` class, with which we can easily monitor both the pending and total bytes being send out over a socket connection. This can be used in order to display something like a progress indicator to the user, sequence certain events within an application, or simply verify that the data has been fully processed over the socket.

In the following example, we establish a socket connection and monitor both the `flash.events.ProgressEvent` and `flash.events.OutputProgressEvent` events when connected to adobe.com using a basic socket connection.

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.OutputProgressEvent;
    import flash.events.ProgressEvent;
    import flash.net.Socket;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.ByteArray;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class SocketProgress extends Sprite {

        private var traceField:TextField;
        private var socket:Socket;

        public function SocketProgress() {
            generateDisplayObjects();
            performOperations();
        }

        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 22;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.wordWrap = true;
```

```
            traceField.multiline = true;
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            socket = new Socket();
            socket.addEventListener(Event.CONNECT, socketConnected);
            socket.addEventListener(IOErrorEvent.IO_ERROR, socketError);
            socket.addEventListener(ProgressEvent.SOCKET_DATA, socketProgress);
            socket.addEventListener(OutputProgressEvent.OUTPUT_PROGRESS,
socketOutputProgress);
            socket.connect("adobe.com", 80);
            traceField.text = "Attempting Connection...\n\n";
        }

        protected function socketProgress(e:ProgressEvent):void {
            var byteArray:ByteArray = new ByteArray();
            traceField.appendText("SOCKET DATA RETURNED:\n");
            socket.readBytes(byteArray, 0, socket.bytesAvailable);
            traceField.appendText(byteArray.toString());
        }

        protected function socketOutputProgress(e:OutputProgressEvent):void {
           traceField.appendText("OUTPUT PROGRESS => bytesPending: " + e.bytesPending
+ " / bytesTotal: " + e.bytesTotal + "\n\n");
        }

        protected function socketConnected(e:Event):void {
            traceField.appendText("Socket Connected!\n\n");
            socket.writeUTFBytes("GET/HTTP/1.1\nHost: adobe.com");
        }


        protected function socketError(e:IOErrorEvent):void {
            traceField.appendText("IO Error: " + e.text + "\n\n");
        }

    }
}
```

The result of the above code is shown in Figure 5-3. As we can see, we now have the ability to monitor the bytes being returned over our established socket as well as access the total bytes to expect from this transaction.

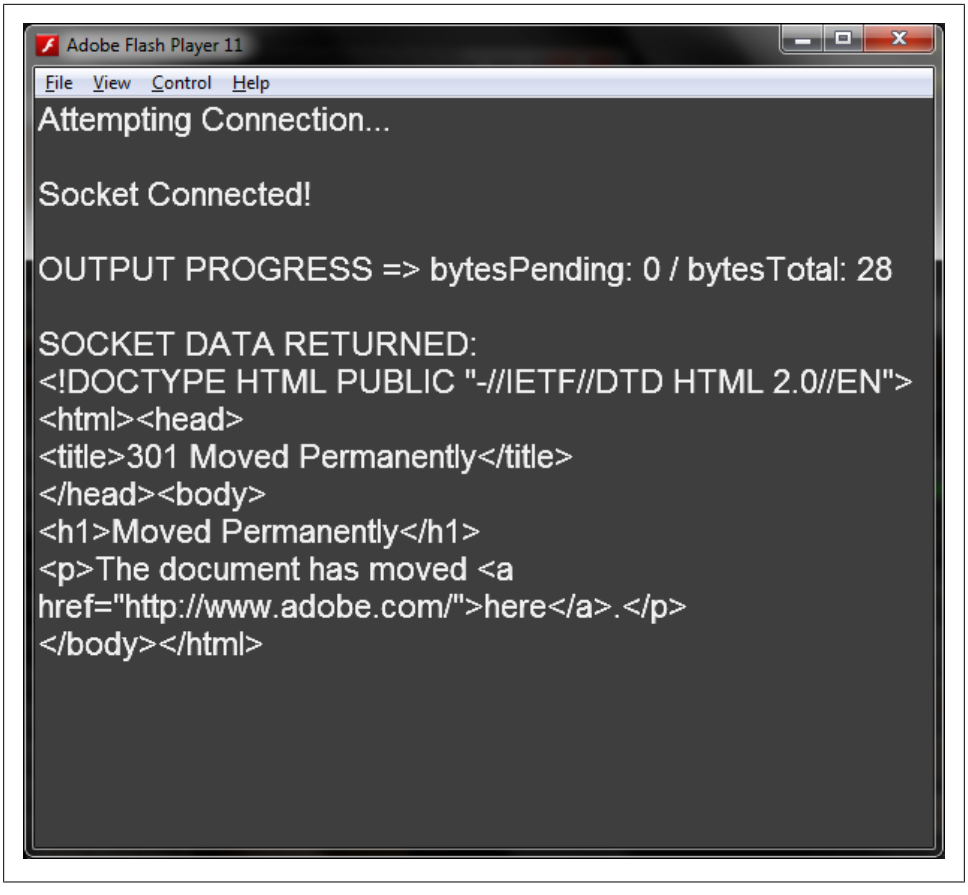


Figure 5-3. Socket output progress

CHAPTER 6
Runtime Enhancements

Among the language and runtime enhancements in Flash Player 11 are a variety of new classes, methods, properties, and architectures whose aim it is to make things easier, smaller, and faster in regard to the runtime and its usage. In this chapter, we will have a look at the variety of language and runtime improvements along with general implementation examples that can be easily built upon for a variety of projects.

Native 64-bit Support

Flash Player comes in a variety of distributions for different desktop and mobile platforms (as always). With Flash Player 11, desktop users of Microsoft Windows, Apple OS X, and a variety of Linux distributions are able to choose between 32-bit or native 64-bit versions of the runtime. There are many advantages to this, mainly having to do with increased compatibility with 64-bit web browsers and operating systems.

With the mobile versions of the Flash Player runtime, the user has no real choice in what is supported, as these versions are tailored to those specific platforms.

There is no action a developer or user must take to enable 64-bit support, as it is integral to the user's choice of browser. Everything should behave the same whether using the 32-bit or 64-bit version of the desktop plugin.

High-Efficiency SWF Compression Support

Previous versions of Flash Player have been able to interpret *.swf* files which use the `zlib` compression mechanism. With Flash Player 11, `LZMA` can also be used to compress a *.swf* for deployment. The need for an additional compression mechanism came about as `Stage3D` content and textures actually benefit greatly from this manner of compres-

sion. In order to be mindful of file size and user bandwidth, Adobe engineers decided to adopt this additional compression mechanism in the new Flash Player.

Keep in mind that even though ActionScript and vector content can be highly compressed with `LZMA`, the compression of bitmap content will be very similar to that seen with `zlib`.

The main benefit of `LZMA` compression is that it is possible to crunch content down to up to 40% of the original size depending upon the content being compressed.

It is important to note that there is not yet tooling support for this new compression mechanism at the time of this writing. If a developer wants to take advantage of this new mechanism, it can be done through other utilities but would require knowledge of something like Python or C++ to get going. For more information on this subject, see Tinic Uro's excellent post at *http://blog.kaourantin.net/?p=124*.

Garbage Collection Advice

Within the `flash.system.System` class is a new method called `pauseForGCIfCollectionImminent()`. The method accepts a single argument that determines the desired imminence value. This optional value must be a `Number` between 0 and 1 where lower values indicate a less intense need for a garbage collection sweep to occur. Using this method, a developer can advise the garbage collector to begin performing its tasks at a time when it is convenient to do so.

In the example below, we create a small "game" with two states. One of these states represents a play level which the user would interact with and is a timed level. The second state represents the paused time between levels, in which the user is provided with a chance to reflect upon their achievements and prepare for another level of play. With `System.pauseForGCIfCollectionImminent()` invoked during the proper time, a developer can advise the garbage collector to perform its actions at the best possible time.

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.TimerEvent;
    import flash.system.System;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.Timer;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class GCAdvice extends Sprite {
```

```
        private var traceField:TextField;
        private var stateName:String;
        private var levelTimer:Timer;
        private var msg:String;

        public function GCAdvice() {
            generateDisplayObjects();
            performOperations();
        }

        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 26;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.wordWrap = true;
            traceField.multiline = true;
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            stateName = "GamePlaying";
           msg = "\n\nUser is playing the game and we do not want to be disruptive by
advising the GC at this time...";
           msg += "\n\nIf we were to try and invoke the GC while a user is interacting
with our game, and while a number of CPU intensive processes are in play, it can
actually freeze the game temporarily.";
            msg += "\n\nThis could cause the user to fail at his task and is visually
disruptive.";

            stage.addEventListener(Event.ENTER_FRAME, monitorGameState);
            levelTimer = new Timer(5000, 1);
            levelTimer.addEventListener(TimerEvent.TIMER_COMPLETE, timeUp);
            levelTimer.start();
        }

        protected function monitorGameState(e:Event):void {
            traceField.text = "Current State: " + stateName;
            traceField.appendText(msg);
        }

        protected function timeUp(e:TimerEvent):void {
            stateName = "LevelComplete";

            System.pauseForGCIfCollectionImminent();

            msg = "\n\nSystem.pauseForGCIfCollectionImminent() invoked!";
```

```
            msg += "\n\nWe do this between levels to avoid any strange behavior such
as display glitches due to the GC running.";
            msg += "\n\nBy advising the GC to fire during a time of little user
interaction and game engine processes - we reduce the risk of disruption
considerably.";
            msg += "\n\nConvenient!";
        }

    }
}
```

During gameplay, we actually do not want to do anything to cause the garbage collector to fire because this could cause the game to freeze up for a few seconds while the process is completed. It is much better to run the garbage collector during some other time in the game lifecycle, preferably when there is little going on and the user is less engaged with the screen, such as in the case of a level complete or level loading screen. The important thing being that there is no disruption to the user experience.

In our demonstration code, we include a `Timer` to decide when to move from a state of active engagement and into a state of rest between levels. This state of rest is the perfect time to advise the garbage collector to perform its duties through `System. pauseForGCIfCollectionImminent()`.

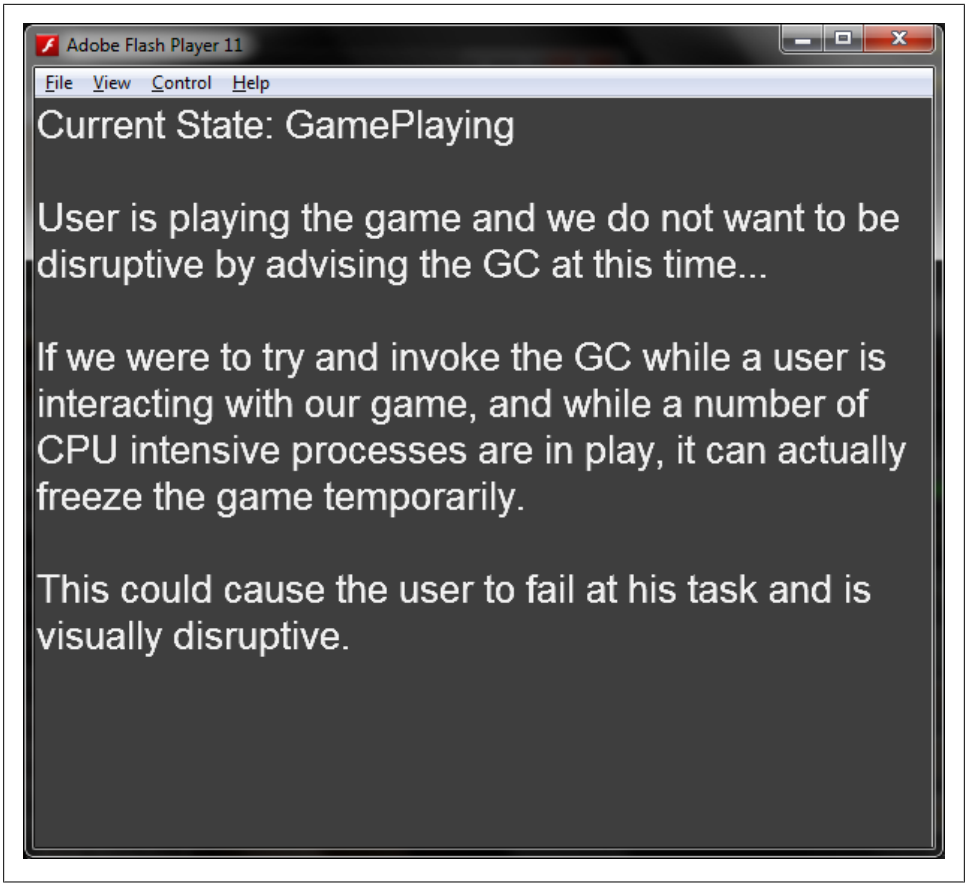


Figure 6-1. Example SWF during the "GamePlaying" state

Once the `Timer` completes for a level, the game will enter a "LevelComplete" state in which the user is able to pause for a moment before continuing on to the next level of game play. This is the perfect time to perform any garbage collection, since the user is no longer actively engaged and there is basically nothing happening on screen. If the garbage collector were to cause any sort of freezing or similar visual disruption, the user would in all likelihood never even notice.

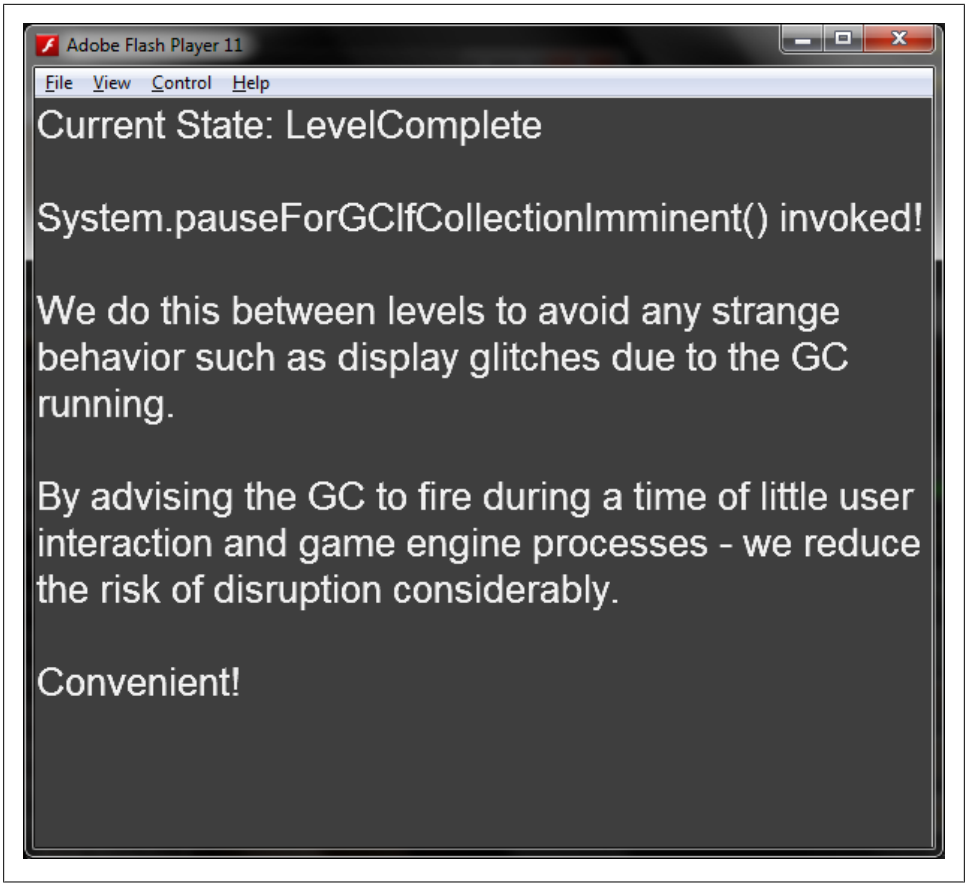
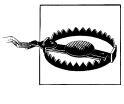


Figure 6-2. Example SWF during the "LevelComplete" state



It's important to note that while a developer can choose when to advise the garbage collector to run within a game or application by invoking `System. pauseForGCIfCollectionImminent()`, there is no guarantee that it will actually fire at that time.

CHAPTER 7
Flash Player Security

Security in any environment is always a just concern. As platforms expand and the general technology landscape shifts, new problems will crop up that require attention, and additional mechanisms are put into place in order to harden platform defenses. Flash Player 11 includes a number of new and updated security mechanisms, including a set of new APIs for secure data generation, SSL and TLS secure sockets, along with expanded runtime support for streaming protected video for desktop and mobile.

Protected HTTP Dynamic Streaming and Flash Access Content Protection Support for Mobile

HTTP Dynamic Streaming (*HDS*) was introduced in Flash Media Server 4 and allows the streaming of live or on-demand media over Hyper Text Transfer Protocol (*HTTP*) instead of the Real Time Message Protocol (*RTMP*) family of streaming protocols. This can be very useful if the port used by *RTMP* (usually 1935) is blocked by a network. This technology has now been extended to mobile versions of the Flash Player runtime, as it was previously only available on desktop versions of the runtime.

It is important to note that delivering video over RTMP is still the most secure, robust method of streaming available. HLS simply provides content providers with additional options.

Adobe Flash Access is a Digital Rights Management (*DRM*) system which can be used when deploying content to Flash Player using Flash Media Server (*FMS*) over RTMP or HTTP. While previous versions of Flash Player have supported Flash Access with the desktop player only, Flash Player 11 provides the same level of security on mobile devices.

For information on Adobe Flash Access 3.0, please refer to the following URL: *http://www.adobe.com/products/flashaccess/*

When developing a client playback targeting Flash Player 11 on mobile devices, the code used is exactly the same as that which has been employed in applications targeting previous versions of Flash Player on the desktop. Any applications which are currently set up to handle DRM through Flash Access 3.0 should now function exactly the same if encountered with a mobile device which supports Flash Player 11.

If using the Open Source Media Framework (*OSMF*) with Flash Access, there are a number of events and properties that have been put into place in order to take advantage of this. For more information, please refer to the content at *http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/org/osmf/events/DRMEvent.html*

Secure Random Number Generator

Using the new `flash.crypto.generateRandomBytes()` method, ActionScript developers can generate a set of highly secure, random bytes for use in applications which require cryptographic keys for use in banking, finance, or even in the creation of general-use secure session ids applicable in just about any application which requires a heightened level of security. The actual functions which generate these cryptographically secure bytes actually are generated by the underlying operating system itself and not Flash Player.

In the example below, we will use this new method to generate a `ByteArray` object containing exactly 1024 randomly generated, cryptographically secure bytes.

```
package {
    import flash.crypto.generateRandomBytes;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.ByteArray;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class RandomBytes extends Sprite {

        private var traceField:TextField;
        private var randBytes:ByteArray;

        public function RandomBytes() {
            generateDisplayObjects();
            performOperations();
        }
```

```
        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 22;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.wordWrap = true;
            traceField.multiline = true;
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            traceField.text = "Here come the securely generated random bytes!\n\n";
            randBytes = generateRandomBytes(1024);
            traceField.appendText(randBytes.toString());
        }

    }
}
```

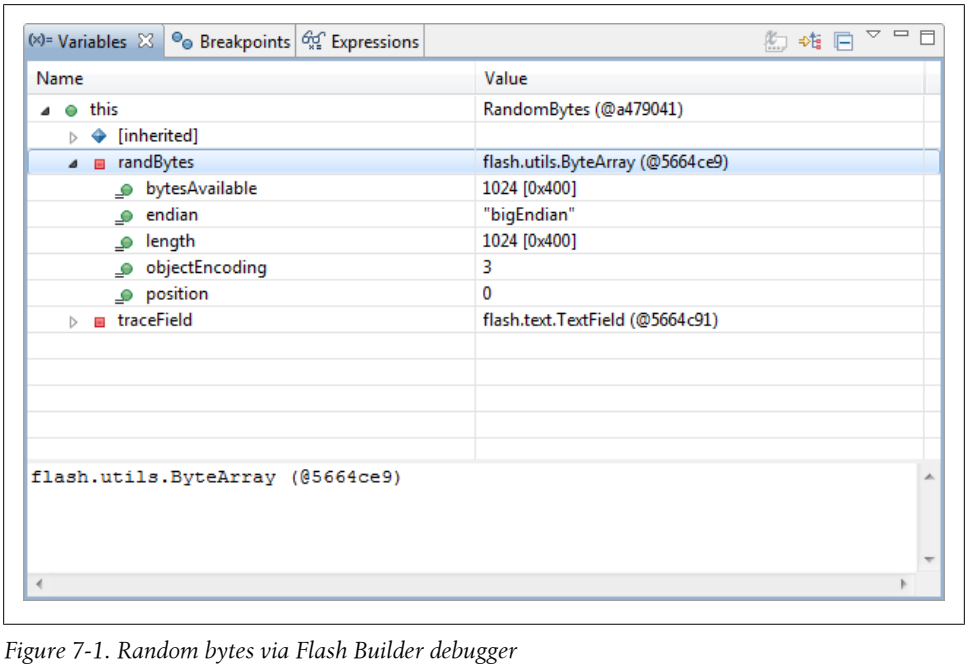


Figure 7-1. Random bytes via Flash Builder debugger

By placing a breakpoint within our code, we can view the generated `ByteArray` details within our debugger. In the figure above, we see that the `ByteArray` contains exactly 1024 bytes. This is precisely the number of bytes requested when invoking `flash.crypto.generateRandomBytes(1024)` in our example code.

We are able to request any number of bytes between 1 and 1024 through this new method.

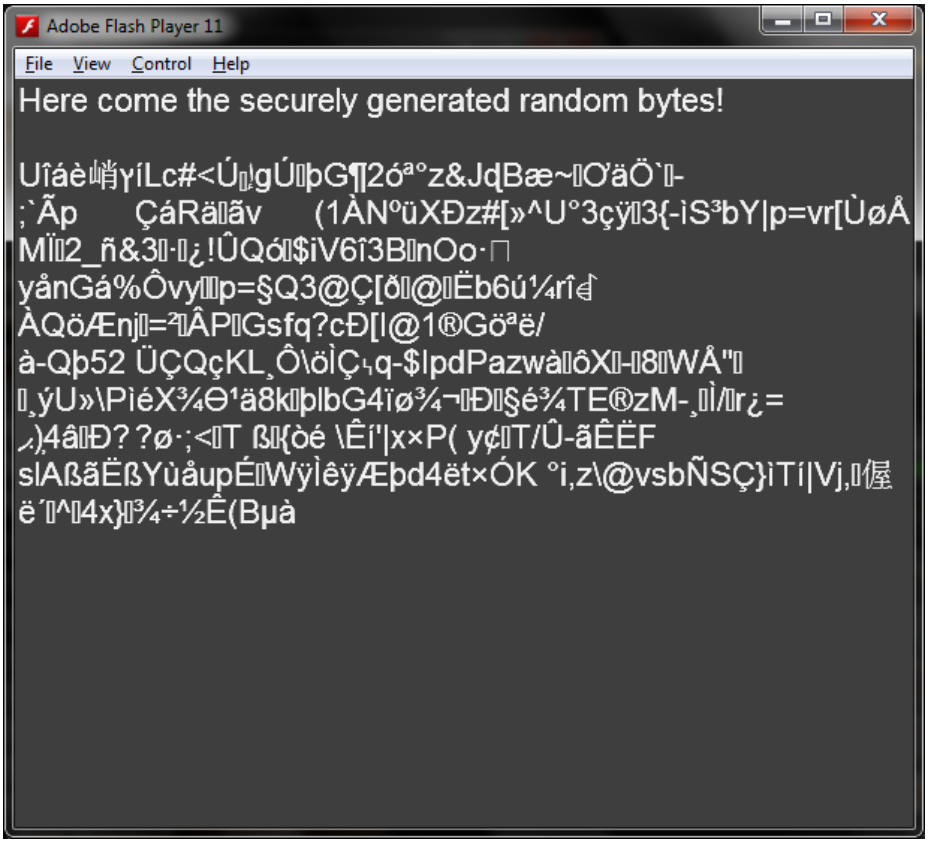


Figure 7-2. Visual string representation of random bytes

Secure Sockets Support

Flash Player 11 has had the ability to establish Transport Control Protocol (*TCP*) socket connections ever since they were introduced in Flash Player 9 with the `flash.net.Socket` class. Sockets allow an application to connect to a server and transmit binary data asynchronously between the client and server. This is often used for gaming, *FTP*, and connections to email servers over *POP3*.

With the addition of the `flash.net.SecureSocket` class, we can now perform the same data transport on secure servers using Secure Sockets Layer (*SSL*) and Transport Layer Security (*TLS*) protocols as well.

When invoking `SecureSocket.connect()`, the method expects two arguments. The first is the IP or full domain name of the server to connect to. The second is the port that is to be used to establish the socket connection. Actually receiving and transmitting data over an established socket connection is done in exactly the same way as if you were using an unsecured socket.

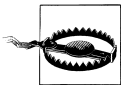
Figure 7-3. SecureSocket results readout

The figure above demonstrates some of the information that can be derived when connecting through a secure socket. In this case, we are making a secure socket connection to adobe.com and then displaying some choice bits of data within a `TextField` on the `Stage`. The code to accomplish this is included below.

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.SecurityErrorEvent;
    import flash.net.SecureSocket;
    import flash.text.TextField;
    import flash.text.TextFormat;

    [SWF(width="600", height="500", backgroundColor="#CCCCCC")]

    public class SocketSecure extends Sprite {
```

```
        private var traceField:TextField;
        private var secureSocket:SecureSocket;

        public function SocketSecure() {
            generateDisplayObjects();
            performOperations();
        }

        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 20;
            defaultFormat.color = 0xFFFFFF;

            traceField = new TextField();
            traceField.backgroundColor = 0x000000;
            traceField.alpha = 0.7;
            traceField.width = stage.stageWidth;
            traceField.height = stage.stageHeight;
            traceField.wordWrap = true;
            traceField.multiline = true;
            traceField.background = true;
            traceField.defaultTextFormat = defaultFormat;
            addChild(traceField);
        }

        protected function performOperations():void {
            secureSocket = new SecureSocket();
            secureSocket.addEventListener(Event.CONNECT, socketConnected);
            secureSocket.addEventListener(IOErrorEvent.IO_ERROR, socketError);
            secureSocket.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
socketSecurity);
            secureSocket.connect("partners.adobe.com", 443);
            traceField.text = "Attempting Secure Connection...\n\n";
        }

        protected function socketConnected(e:Event):void {
            traceField.appendText("Secure Socket Connection => " +
secureSocket.serverCertificateStatus + "\n\n");
           traceField.appendText("Issuer:\n" + secureSocket.serverCertificate.issuer
+ "\n\n");
         traceField.appendText("Subject:\n" + secureSocket.serverCertificate.subject
+ "\n\n");
            traceField.appendText("Valid Not Before:\n" +
secureSocket.serverCertificate.validNotBefore + "\n\n");
            traceField.appendText("Valid Not After:\n" +
secureSocket.serverCertificate.validNotAfter + "\n\n");
        }

        protected function socketSecurity(e:SecurityErrorEvent):void {
            traceField.appendText("Security error => " + e.text + " => " +
secureSocket.serverCertificateStatus + "\n\n");
        }

        protected function socketError(e:IOErrorEvent):void {
```

```
            traceField.appendText("IO Error: " + e.text + " => " +
secureSocket.serverCertificateStatus + "\n\n");
        }

    }
}
```



Note that while this example will work when authoring in Flash Professional or Flash Builder, you will want to keep in mind the Flash player security sandbox restrictions on cross-domain data retrieval when deploying something like this to a server.

APPENDIX

Additional Resources

We hope you have found this book useful in getting a jump start on understanding and using the new features available in Flash Player 11. If you wish to explore further, we recommend the following resources.

What's New in Adobe AIR 3

Flash Player 11 has a companion runtime for standalone applications on desktop and mobile called the Adobe Integrated Runtime (AIR). To learn about the new features in Adobe AIR 3.0, pick up a copy of *What's New in Adobe AIR 3* (O'Reilly).

Using Stage3D Frameworks

Adobe has worked closely with a number of frameworks and gaming engines to make sure `Stage3D` is well supported by a number of projects throughout industry and the community. A sampling follows.

3D Frameworks

- Alternativa Platform
  http://alternativaplatform.com/en/
- Away3D
  http://www.away3d.com/
- Coppercube
  http://www.ambiera.com/coppercube/
- Flare3D
  http://www.flare3d.com/
- Minko

http://aerys.in/minko

- Sophie 3D
  http://www.sophie3d.com/website/index_en.php
- Yogurt3D
  http://www.yogurt3d.com/
- Zest3D
  http://zest3d.digital-glue.com/

2D Frameworks

- Starling
  http://www.starling-framework.org/
- M2D
  https://github.com/egreenfield/M2D
- ND2D
  https://github.com/nulldesign/nd2d

Articles and Resources

Here are some additional resources which are available on the Web.

- Adobe Flash Player Developer Center
  http://www.adobe.com/devnet/flashplayer.html
- Mobile and Devices Developer Center
  http://www.adobe.com/devnet/devices.html
- Flash Platform Game Developer Center
  http://www.adobe.com/devnet/games.html
- Rich Internet application development
  http://www.adobe.com/devnet/ria.html
- Video Technology Center
  http://www.adobe.com/devnet/video.html
- Adobe Evangelists Super Blog
  http://adobeevangelists.com/superblog/
- Adobe Flash Player Support Center
  http://www.adobe.com/support/flashplayer/downloads.html
- Setting Up Flash Builder 4.5 for Flash 11 and AIR 3 Apps
  http://www.fmsguru.com/showtutorial.cfm?tutorialID=59

About the Author

Joseph Labrecque is primarily employed by the University of Denver as senior interactive software engineer specializing in the Adobe Flash Platform, where he produces innovative academic toolsets for both traditional desktop environments and emerging mobile spaces. Alongside this principal role, he often serves as adjunct faculty, communicating upon a variety of Flash Platform solutions and general web design and development subjects.

In addition to his accomplishments in higher education, Joseph is the proprietor of Fractured Vision Media, LLC, a digital media production company, technical consultancy, and distribution vehicle for his creative works. He is founder and sole abiding member of the dark ambient recording project "An Early Morning Letter, Displaced" whose releases have received international award nominations and underground acclaim.

Joseph has contributed to a number of respected community publications as an article writer and video tutorialist and is author of the *Flash Development for Android Cookbook* (2011 Packt Publishing - ISBN: 1849691428), *What's New in Adobe AIR 3* (O'Reilly - ISBN: 9781449311070), *What's New in Flash Player 11* (2011 O'Reilly - ISBN: 9781449311094), and co-author of *Mobile Development with Flash Professional CS5.5 and Flash Builder 4.5: Learn by Video* (2011 Adobe Press - ISBN: 0321788109).

He regularly speaks at user group meetings and industry conferences such as Adobe MAX, FITC, D2W, and a variety of other educational and technical conferences. In 2010, he received an Adobe Impact Award in recognition of his outstanding contribution to the education community. He has served as an Adobe Education Leader since 2008 and is also a 2011 Adobe Community Professional.