

「What's New in Adobe AIR 3 Quickstart Guide for Desktop and Mobile Development」

本文は <http://shop.oreilly.com/product/0636920021681.do> ページから無料で入手できる「What's New in Adobe AIR 3 Quickstart Guide for Desktop and Mobile Development」をヒム・カンパニー 永井勝則が自主的に日本語に訳したものです。

<http://www.himco.jp/>

[knagai@himco.jp](mailto:knagai@himco.jp)

(2012/1)

## Adobe AIR 3 の新機能

### (注意)

原書では Flash Builder 4.6 が使用されていますが、本日本語訳ドキュメントでは、主に Flash CS5.5 を使って同じ作業を行っています。本書のサンプルコードを実行するには、Flash Builder 4.6 が必要ですが、Flash CS5.5 でも、「Flash Professional CS5.5 で AIR SDK を使用する 方法」([http://kb2.adobe.com/jp/cps/910/cpsid\\_91088.html](http://kb2.adobe.com/jp/cps/910/cpsid_91088.html))で公開されている設定を行うことで、AIR 3 の新機能を試すことができます。また Android アプリを作成するときには、Android SDK をダウンロード (<http://developer.android.com/intl/ja/sdk/index.html>)し、システムに必要なパスを設定しておく と便利 です。

サンプルコードは <http://shop.oreilly.com/product/0636920021681.do> からダウンロードできます。

---

*Quickstart Guide for Desktop and Mobile Development*

*What's New in*

# Adobe AIR 3



O'REILLY®



Adobe  
Developer  
Library

*Joseph Labrecque*

## 1章: MovieClip と描画 API の向上

Adobe AIR は Adobe Flash Player ランタイムとそのコアの機能の多くを共有しています。Flash Player は 1990 年代中頃、Web ベースのメディアアニメーションと表示テクノロジーとして登場しました。その歴史の大部分において、グラフィックとして強烈かつ高機能で美しいイメージのレンダリングと操作性によって大きな信頼を得てきました。AIR 3 では、Flash Player においても中核で AIR にも受け継がれているグラフィックとベクター描画テクノロジーが、さらに便利な方法で拡張され向上しています。

### 三次ベジェ曲線

AIR のこのリリースでは、新しい曲線を描画したいときに毎回、自分でたくさんの複雑な式を記述することなく、簡単な三次ベジェ曲線が作成できるグラフィック描画 API が追加されました。新しい `cubicCurveTo()` メソッドは、これを正確に行うために6つの引数を取ります。引数は1つめのコントロールポイント用の  $x$  と  $y$  座標のセットと、2つめのコントロールポイントの同様のセット、アンカーポイント用の座標のセットです。



ベジェ曲線は、コンピュータグラフィックスで4つの別々の点を使った滑らかな曲線を描くために幅広く使用されています。4つの点には開始点と終了点、それに描画する曲線に曲がりの向きを知らせ曲線を引き出す2つのコントロールポイントがあります。

この曲線は、線が現在ある場所ならどこからでも開始でき、これまでのグラフィック API 呼び出しと同じように、`moveTo()` メソッドを使って開始点を正確に配置することができます。2つのコントロールポイントは線の曲がりに影響を与える点で、アンカーポイントは描画される曲線の端点です。図 1-1 はこれを示しています。

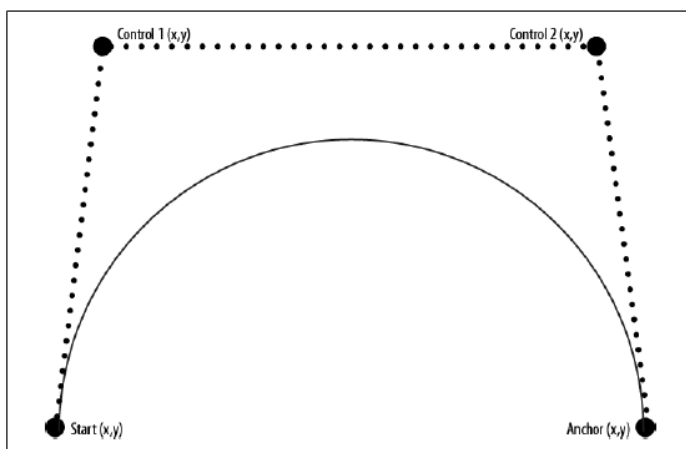


図 1-1: 三次ベジェ曲線の動作

次のサンプルでは、Sprite を作成して新しい cubicCurveTo()メソッドを呼び出し、ステージ全体にわたって三次ベジェ曲線を描画しています。

```
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);
        }
    }
}
```

これにより、図 1-2 に示すようなアプリケーションウィンドウがレンダリングされます。

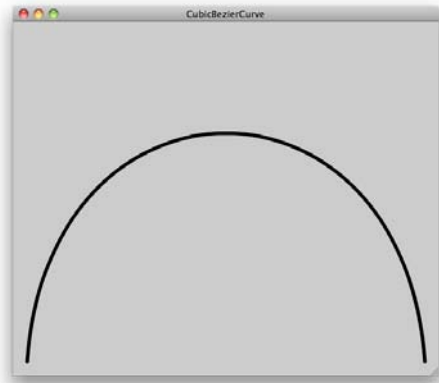


図 1-2: 三次ベジエ曲線

`DisplayObjectContainer.removeChilden()`

コンテナオブジェクトからすべての子を削除したい場合、AIR 3 以前では、まず `DisplayObjectContainer.numChildren` を使って存在する子の数を求めてから、子オブジェクトを走査して1度に1つずつ削除する必要がありました。

しかし `DisplayObjectContainer.removeChilden()` メソッドを使用すると、簡単な1つの命令で、親コンテナのすべての子を削除し、それらをすべてガベージコレクションの対象にすることができます。



ただし `removeChildren()` を呼び出すときにはその前に、イベントリスナーの削除やこれらの子への参照を必ず削除するようにします。そうしないと、ガベージコレクタはこれらのオブジェクトに割り当てられたメモリすべてを解放できません。

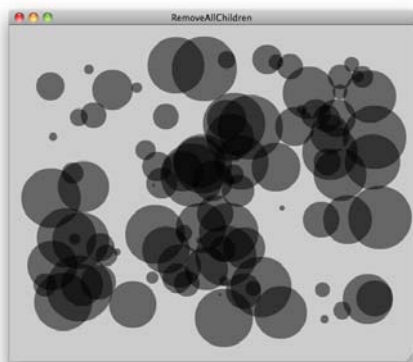


図 1-3: 子の削除

次の例では、ステージ上にたくさんの MovieClip をダイナミックに生成し、ステージに単純な MouseEvent.CLICK イベントを監視するイベントリスナーを追加しています。するとステージのクリックによって、stage.removeChildren()が呼び出され、すべての MovieClip が削除されます。

```
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

実はいささか驚いてしまうことなのですが、以前の Flash Player と AIR のバージョンにはこのプロパティがありませんでした。MovieClip インスタンスは、メインのタイムラインから独立した自分自身のタイムラインを持つという点が特徴的です。通常デベロッパーは、特定の MovieClip インスタンスが実際に再生中であるかどうかを知りたい場合が多くあり、これまでは MovieClip の現在のフレームを経時的に監視してそれが変化しているかどうかを調べる必要がありました。

この新しい機能の使用は実に直接的で、MovieClip.isPlaying はすべての MovieClip インスタンスが持つただのプロパティです。これは再生中には true の、停止中には false の Boolean 値を返します。次の例では、MovieClip を作成しそれを DisplayList に追加して、その isPlaying プロパティを TextField に書き込んでいます。

```
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 = 0xFFFFFFFF;

    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;
}
}

```

}



図 1-4: MovieClip.isPlaying

図 1-4 はこのコードの実行結果です。MovieClip をクリックすると再生がトグルし、isPlaying プロパティの Boolean 値が調べられて、画面上に表示されます。

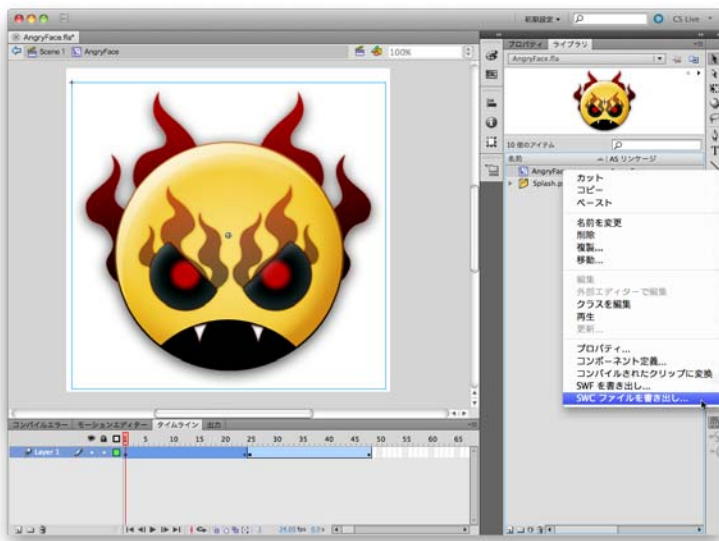


図 1-5: Flash Professional からの SWC の書き出し



この例では、Flash Professional CS5.5 でアニメーションさせた MovieClip オブジェクトを使って SWC の一部として書き出し、Flash Builder 4.5 にリンクさせています。同じような方法はほかにもありますが、これは Flash Professional を使っていない場合には、非常に分かりやすい直接的な方法です。

## 2章: 外部イメージへの適応

AIR と Flash Player との密接な関係性を考慮し、Flash Player のベクター描画オブジェクトを処理する機能を踏まえると、Flash Platform では埋め込んだり外部に置いたイメージファイルのビットマップデータも利用できるという機能を見落としがちになります。使用するのが PNG であれ JPG、GIF または新しい JPEG-XR ファイルタイプであれ、この新しいイメージ処理技術はかなり劇的な方法で拡張され大きく向上しました。

### 拡張された高解像度ビットマップサポート

ロードする BitmapData オブジェクトはこれまで、幅または高さが 8,191 ピクセル、合計のピクセル数が 16,777,215 ピクセルに制限されており、高解像度のイメージを処理するには十分ではありませんでした。一般的なデジタルカメラの解像度はここ 10 年でメガピクセルを超え、さらに高い解像度が求められていることは明白です。AIR 3 ではこの制限がなくなり、さまざまなプロジェクトで高解像度のビットマップが使用できるようになりました。



1 メガピクセルは 1,000,000 ピクセルです。AIR 2 は 16,777 メガピクセルまでをサポートしますが、AIR 3 ではこの制限がなくなりました。

実際にはこの振る舞いのサポートは AIR ランタイム自体に組み込まれているので、何かを行うときこれを有効化する必要はありません。次の例では Loader クラスを使って高解像度イメージを AIR プロジェクトに持ちこんでいます。

```
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 = 0xFFFFFFFF;

        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;
    }
}
}

```

図 2-1 はこのコードの実行結果です。

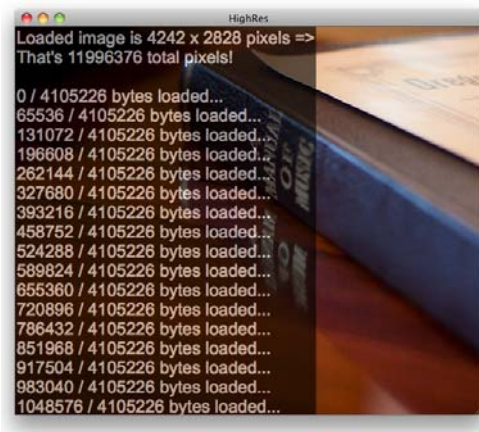


図 2-1: 高解像度ビットマップ

## JPEG-XR サポート

AIR 3 ではほかにも拡張されたイメージファイル形式のサポートが含まれています。以前の AIR でサポートされたのは GIF、JPEG、PNG のイメージファイル形式で、これら以外のファイルはそれを解釈する外部コードのライブラリに依存していました。新たに追加された JPEG-XR(国際規格 ISO/IEC 29199-2)は、JPG にまさる効率的な圧縮を、不可逆、可逆両方のオプションとともに AIR にもたらします。JPEG-XR はまた、PNG 形式のように、完全なアルファチャンネルも持っています。



```

        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 = 0xFFFFFFFF;

        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...\r\n");
    }

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

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

```

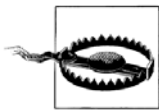
図 2-2 はこのコードの実行結果です。



図 2-2: JPEG-XR のサポート

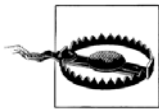
### 3章: Stage3D: 高性能な視覚表現

AIR 3 の機能について最も取り上げられることの多いのは、Stage3D(コード名 Molehill)によって高速化したグラフィックレンダリングエンジンです。この高度なレンダリングアーキテクチャは、そのAPIを直接使用するか、または API の最上位で構築されたエンジンやフレームワークのいずれかを実装した AIR 内の 2D と 3D ビジュアルオブジェクト両方のレンダリングに使用できます。



AIR で Stage3D を使用するには、アプリケーション記述ファイルで `<renderMode>direct</renderMode>`を設定する必要があります。

Stage3D API を使用する大きなメリットは、サポートされるシステム構成で Stage3D によってレンダリングされるものはすべて、そのシステムの GPU(Graphics Processing Unit)を通して直接レンダリングされる、ということです。これにより GPU が複雑なレンダリング作業に関する全体的な責任を負い、CPU(Central Processing Unit)はそのままそのほかの機能で利用できるようになります。



特定のシステムでハードウェアによる Stage3D のレンダリングが行えない場合には、Stage3D は代替としてソフトウェアを使ってレンダリングを行います。

#### Stage3D の高速なグラフィックレンダリング

新しい `flash.display.Stage3D` クラスは、AIR 内での表示オブジェクトとしての振る舞いにおいて、`flash.media.StageVideo` と似ています。Stage3D も StageVideo のように AIR の `DisplayList` に追加せず、オブジェクトのスタックから離れて存在します。`DisplayList` は視覚的なスタック順で、StageVideo を使用する場合のように、Stage3D の上に来ます。



Stage3D は現在、デスクトップ上でのみサポートされています。モバイルの Stage3D は今後の AIR リリースでサポートされます。

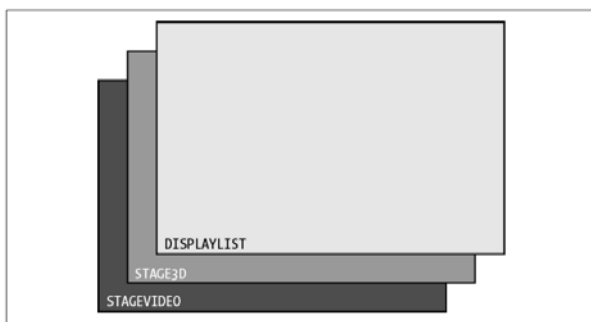


図 3-1: Stage3D は StageVideo と従来の DisplayList の間に存在する

### Stage3D の要素

前述したように、Stage3D 自体はかなり低いレベルに移植されているので、ほとんどの ActionScript デベロッパーにとってその使用は決して容易ではありません。これまで 3D プログラミング環境で作業したことがない方には、そのために必要な用語やオブジェクトの多くは通常のワークフローとはかけ離れたもののように思えるかもしれません。



これらの生の API の使用方法に関するサンプルを見たい場合には、Thibault Imbert の Web サイト (<http://www.bytearray.org/>) をおすすめします。ここには多くの Stage3D サンプルが掲出されており、本書で以降見ていくよりもかなり突っ込んだ多くの情報を得ることができます。

AIR アプリケーションで Stage3D を設定しレンダリングするには、次のコアクラスをインポートする必要があります。

```
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;
```

Stage3D のビューの上に何かをレンダリングするには、頂点とフラグメントシェーダを使用する必要があります。これらの用語に不慣れな方のために言うと、シェーダは、システムの GPU でのエフェクトのレンダリングの計算に使用される低レベルのソフトウェア命令で、実際、グラフィックレンダリングパイプラインや GPU を直接プログラムするために使用されます。頂点シェーダは視覚要素の直接の外見に影響し、フラグメントシェーダは要素のサーフェスの詳細を管理します。



Adobe Pixel Bender 3D を使用すると、頂点とフラグメントシェーダの成果物を作成し、出力イメージを 3D ハードウェアで実行することができます。これらのカーネルは 3D オブジェクトに作用し、その外見に影響を与えます。

シェーダを実際に作成しレンダリングするには、AGAL (Adobe Graphics Assembly Language) という新しい言語を使う必要があります。AGAL は極めて低レベルで、相当に意欲にあふれた人以外には向いていません。従来の ActionScript デベロッパーはおそらく AGAL に四苦八苦するでしょうが、OpenGL や一般的なアセンブリ言語などの環境になじんだ方にはお手のものかもしれません。そのどちらの場合でも、Stage3D の使用に関しては、より高レベルのフレームワークを使用することが推奨されます。



Stage3D は、Flash Player で複雑な 3D グラフィックを作成しレンダリングするために使用される非常に多くの 3D フレームワークを備えていますが、実際にはサーフェスのレンダリングは、高速な視覚体験の有効化するために使用される 3D や 2D コンテンツに使用できます。

ActionScript プロジェクトで Stage3D を機能させるために必要な基本設定は、次のアクションを実行することです。

- Context3D オブジェクトを、stage.stage3D 配列を通して要求します。
- Context3D オブジェクトが準備できたら、IndexBuffer3D や VertexBuffer3D オブジェクトなど、仕様に関して Context3D を設定します。
- AGAL を使って、Program3D オブジェクト内で使用するさまざまなシェーダを作成します。
- 最後に、レンダーラープ (Event.ENTER\_FRAME) を使ってこのすべてを処理し、Context3D、Program3D と Matrix3D オブジェクトコントロールのセットを使って、Stage3D オブジェクトにレンダリングします。

こう言うと、複雑そうに聞こえますが、実際その通りです。ここに述べた大まかな処理とそれに関する複雑性は、実際には、Stage3D 基盤の上に独自のフレームワークやエンジンを構築しようとする人々にのみ意味があります。次節では、3D フレームワークを実際を使って Flash Player 内にコンテンツをレンダリングする方法を見ていきます。

訳注:

原書のサンプルには、上記の複雑な基本設定の例として SimpleStage3D クラスが用意されています。これを実行するには、<http://www.bytearray.org/wp-content/projects/agalassembler/com.zip> をダウンロードし、AGALMiniassembler クラスをインポートする必要があります。



github には、Stage3D 向け AGAL の作成を簡素化することを目的とした EasyAGAL という名前のプロジェクトがあります (<https://github.com/Barliesque/EasyAGAL>)。

#### Away3D を使った Stage3D サンプル

ありがたいことに、Stage3D の API や AGAL は、実際に使用したいと思わない限り、直接扱う必要はありません。AIR Stage3D API の高レベルの代替として使用できる、堅牢で完成されたフレームワークはいくつか存在します。次の例は Away3D を使った簡単な実装の例で、Stage3D を使ってアニメーションするプリミティブをレンダリングします。

このような基本的なレンダリングの実行に関心のあるみなさんはまず、直接的な API を使った方法を試し (SimpleStage3D クラスのような)、つづいて Away3D 実装の方法を試して2つを比べてみてください。その違いは、Away3D などのフレームワークでは API が非常に使いやすい形式に変えられている、という点にあります。



下記のサンプルを実行するにはその前に、<http://away3d.com/> から適切な Away3D フレームワークのコードをダウンロードしておく必要があります。

訳注:

実際には <http://away3d.com/download/> から Away3D 4.0 のソースファイルをダウンロードします。これは ZIP 形式に圧縮されているので展開し、その src フォルダをプロジェクトのソースパスに追加します。

下記のコードからも分かるように、必要なことはただ View3D インスタンスを作成し、プリミティブの WireframeCube などのオブジェクトを生成して、それらを View3D.scene プロパティに追加するだけです。ただし View3D のレンダリングを忘れてはいけません。これは通常、Event.ENTER\_FRAME を使い、このイベントから呼び出されるリスナー関数内で View3D.render() を実行するレンダーループから作成されます。レンダーループの繰り返し内では、オブジェクトのプロパティを変更することができます。次の例では WireframeCube プリミティブの rotationX と rotationY プロパティを変更して 3D アニメーションを作成しています。

```
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();
    }
}
}
}

```

このコードを実行すると、x と y 軸周りにゆっくり回転するワイヤーフレームのキューブが表示されます。Away3D には、3D コンテンツのレンダリングに使用できる多くの異なるプリミティブやマテリアルがパッケージされて備わっています。このサンプルは、広範囲にわたるフレームワークのただ1つの例を示したにすぎません。

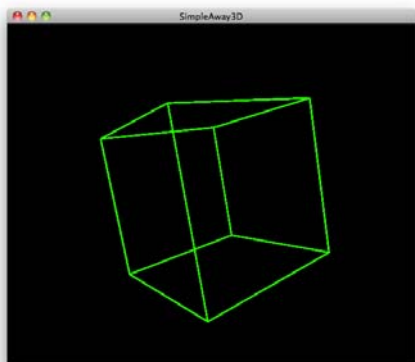


図 3-2: Away3D を使ってレンダリングした WireframeCube プリミティブ



Stage3D フレームワークとライブラリのリストは本書の付録に掲載しています。

#### Starling を使った Stage3D サンプル

Starling ( <http://starling-framework.org/> ) は Adobe と Sparrow Framework

(<http://www.sparrow-framework.org/>)によって始められたオープンソースの労作で、Flash Platform のデベロッパーが使用する従来の DisplayList をエミュレートする、Stage3D 用の 2D フレームワークを作成します。デベロッパーは実際に、Sprite や MovieClip、TextField といったなじみのある概念を、ほとんど同じような方法で使用することができます。



Starling は、Flash の DisplayList API を真似た、iOS 用の Sparrow フレームワークの直接的な移植です。

Starling フレームワークは <https://github.com/PrimaryFeather/Starling-Framework> から無償で入手できます。わずか 80K で非常に軽量です。またオープンソースプロジェクトなので、みなさんはコミュニティに貢献し、フレームワークの成長を支援することができます。

次の簡単な例では、Quad を作成し時計回りに回転させます。そのためにはまず、メインアプリケーションクラスで Starling クラスを設定する必要があります。ここで重要なことは、starling.core.Starling の新しいインスタンスを作成し、それに残りのコードを記述した Game クラスと、現在のステージへの参照を渡していることです。最後、物事を次に進めるには Starling.start() を呼び出す必要があります。

```
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();
    }
}
}
}

```

Starling が設定できたら、初期化に使用する Game クラスがいます。この例では、レンダリングするものは全部、メインアプリケーションクラスと同じパッケージにある Game クラスに存在しています。

Game クラスではまず、これがステージに追加され、表示機能を実行する準備が整ったことを知る必要があります。そのためには Event.ADDED\_TO\_STAGE タイプのイベントリスナーを追加します。このイベントが発射されたら、Starling クラスを使ったビジュアルオブジェクトの描画が安全に行えます。



Sprite や Event といったなじみのあるクラスを使用することもできますが、ここではコアの Flash クラスではなく、これらのクラスの Starling バージョンを使用しています。

ここでは Quad を設定しています (onStageReady()内)。これは基本的に、2つの三角形を合わせて矩形平面にしたものです。これは変換点 (ピボットポイント) をセンターに設定した Quad をステージのセンターに置く方法で設定しています。Quad はこうすることで、デフォルトの左上隅でなく、そのセンターを中心に回転するようになります。Quad.setVertexColor() を使って、緑の異なるシェードをグラデーション点として設定しています。

最後、Event.ENTER\_FRAME から呼び出されるレンダーループを設定しています。ここは時間の経過とともに変化が発生する場所で、今の場合では、Quad が時計回りに回転します。

```
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;
    }
}

```

このコードをコンパイルしデスクトップで実行すると、Stage3D によって高速化された 2D グラフィックの使用が、この素晴らしいフレームワークのおかげで実に簡単に実現できることが分かります。

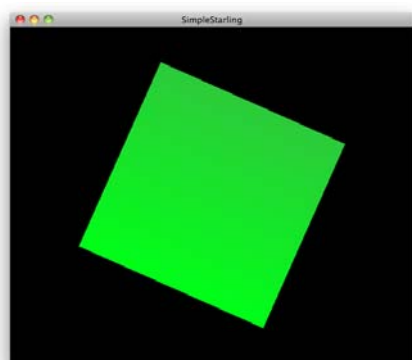


図 3-3: Starling を使った簡単な Quad のレンダリングと回転



Starling フレームワークの詳細については、Thibault Imbert の電子書籍「Introducing Starling」で読むことができます (<http://www.bytearray.org/>)。この書籍は Starling と同様、無料です。

#### Stage3D をサポートするツール

Stage3D は多くの 3D フレームワークでサポートされているだけでなく、さまざまな製品でも新しい機能として取り入れられています。中でも Unity 開発環境と Flare3D Studio が有名です。

#### Unity

Unity には Stage3D が組み込まれており、サポートされる機能に応じて、Unity への書き出しとほとんど同じようにコンパイル済みの SWF を直接書き出すこともできます。サポートされる機能にはフィジックス、ライトマッピング、オクルージョン、カリング、カスタムシェーダ、ライトプロープ、パーティクルシステム、ナビゲーションメッシュなどがあります。これは、Flash と AIR のゲームに関係する実にすばらしい開発です。Unity はこのようにゲームの偉大なエンジンと編集環境なので、その使用はさまざまなプラットフォームをターゲットにする多くのゲームデベロッパーによってすでに始まっています。



Flash Player 向けの Unity コンテンツをレンダリングすると、そのコンテンツはさらに大きな Flash Player や AIR プロジェクト内でビルドできるようになります。その用途の1つには、ゲーム用の堅牢なメニューシステムの作成があります。

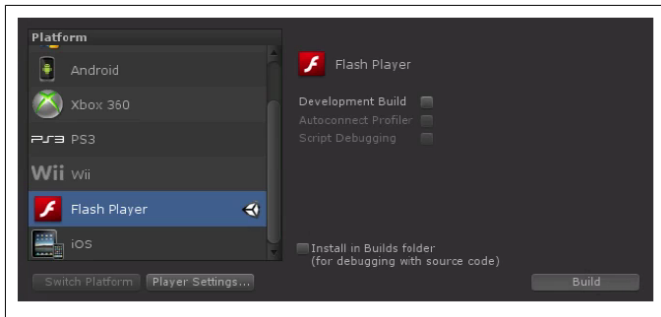


図 3-4: Unity3D のビルド設定

## Flare3D Studio

Flare3D Studio は Flash Platform によって構築され、AIR ランタイムによって配布される 3D デザイン環境です。

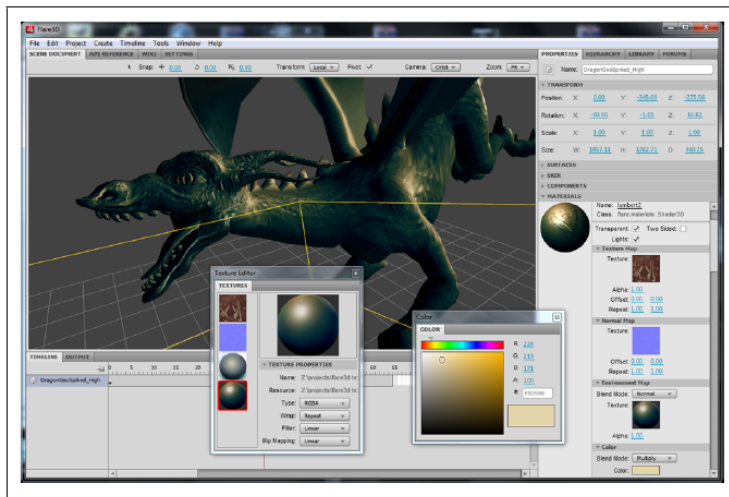


図 3-5: Flare3D Studio

#### 4章: モバイルでの向上: StageText と StageVideo

AIR が、iOS や Android といったモバイル OS に向けた開発の本格的な選択肢となるにしたいが、システムのハードウェアアクセラレーションやリッチなテキスト入力フレームワークといった構成要素のメリットを生かすために必要なプラットフォーム機能を備える重要性が増しています。AIR 3 ではランタイムとこれらの基盤システムの機能性との統合が図られています。

##### ネイティブテキスト入力 UI の StageText (モバイル)

AIR 3 の新しい `flash.text.StageText` クラスを使用すると、オートコレクト(自動訂正)やオートキャピタライズ(文頭の自動大文字化)、ユーザーに提示する仮想キーボード特有のタイプなどの機能を公開し、OS のテキスト入力メカニズムに直接入力できるようになります。

StageText も、StageWebView や StageVideo と同様、従来の `DisplayList` には含まれず、独自の方法で処理する必要があります。StageText で作成されたネイティブのテキスト入力フィールドはつねに `DisplayList` の上にレンダリングされ、ステージにレンダリングされるほかのコンテンツの上に表示されます。



StageText には背景や境界の装飾は含まれません。これは以下に示すように、`DisplayList` を通して提供する必要があります。

StageText インスタンスを作成するにはまず、必要な背景要素を描画し、`DisplayList` に追加します。次の例ではグラフィック API を使って簡単なボックスを描画しています。これはユーザーに対し、タップして StageText オブジェクトオブジェクトにフォーカスを与えるインジケータの役割を果たします。次いで新しい `StageTextInitOptions` インスタンスを定義し、`multiline` パラメータ(複数行のテキストを表示するかどうか)に `true` か `false` を指定しています。

これらの設定が済んだら、StageText オブジェクトをインスタンス化し、プロパティの定義を行っています。プロパティでは、テキストをオートキャピタライズするかどうかやオートコレクトできるようにするかどうか、また Return キーに表示するラベルの設定のほか、この特定のフィールドで使用されるソフトキーボードの外見なども指定できます。



ソフトキーボードの外見を指定する `flash.text.SoftKeyboardType` には次のオプションがあります。

`SoftKeyboardType.CONTACT` (個人の名前や電話番号を入力するためのキーパッド)

`SoftKeyboardType.DEFAULT` (現在の入力方式のためのデフォルトキーボード)

`SoftKeyboardType.EMAIL` (メールアドレスを指定するために最適化されたキーボード)

`SoftKeyboardType.NUMBER` (PIN 入力のためのテンキー)

`SoftKeyboardType.PUNCTUATION` (URL 入りに最適化されたキーボード)

`SoftKeyboardType.URL` (URL キーボード)

最後に、現在のステージを `StageText.stage` プロパティに割り当て、`StageText.viewport` プロパティに `Rectangle` を割り当てて、位置とサイズを決めています。

```
package {
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.geom.Rectangle;
    import flash.text.ReturnKeyLabel;
    import flash.text.SoftKeyboardType;
    import flash.text.StageText;
    import flash.text.StageTextInitOptions;

    [SWF(backgroundColor="#CCCCCC")]

    public class MobileStageText extends Sprite {

        private var decoration:Sprite;
        private var stageText:StageText;
        private var stageTextInitOptions:StageTextInitOptions;

        public function MobileStageText() {
            super();
            stage.scaleMode = StageScaleMode.NO_SCALE;
```

```

        stage.align = StageAlign.TOP_LEFT;

        generateDisplayObjects();
    }

    protected function generateDisplayObjects():void {
        decoration = new Sprite();
        decoration.graphics.lineStyle(4, 0x000000, 1);
        decoration.graphics.beginFill(0xFFFFFFFF, 1);
        decoration.graphics.drawRect(6, 44, stage.stageWidth-12, 70);
        this.addChild(decoration);

        stageTextInitOptions = new StageTextInitOptions(false);
        stageText = new StageText(stageTextInitOptions);
        stageText.softKeyboardType = SoftKeyboardType.DEFAULT;
        stageText.returnKeyLabel = ReturnKeyLabel.DONE;
        stageText.autoCorrect = true;
        stageText.fontSize = 40;
        stageText.color = 0x440000;
        stageText.fontWeight = "bold";
        stageText.stage = this.stage;
        stageText.viewPort = new Rectangle(10, 50, stage.stageWidth-20, 70);
    }
}
}

```



このサンプルでは、本書のほかのモバイル AIR サンプルと異なり、ソフトウェアキーボードが StageText インスタンスと画面上に共存できるように、アプリケーション記述ファイルで向きを“portrait”に設定しています。

```
<aspectRatio>portrait</aspectRatio>
```

このコードをモバイルデバイスにコンパイルすると、図 4-1 に示すような結果が得られます。StageText は Android と iOS 両方でサポートされています。



図 4-1 : Android のネイティブテキスト



StageText に関連する複雑性を回避するため、Christian Cantrell は、これらの API の使用時に遭遇する多くの困難を支援する NativeText ラッパーを開発しています。NativeText は <https://github.com/cantrell/StageTextExample> からダウンロードできます。

#### StageVideo ハードウェアアクセラレーション(モバイル)

StageVideo はこれまで、Flash Player(と AIR for TV)で使用できた機能の1つですが、その領域はほかの Flash Platform にも広がり始めています。サポートされるデバイスで StageVideo クラスを使用すると、AIR3 を使ってビデオメディアを表示するとき、デバイスのハードウェアアクセラレーション機能を活用してパフォーマンスをアップさせ、CPU の使用量を減らし、バッテリーの使用をさらに効率化することができます。



StageVideo は Android 3.1 と BlackBerry Tablet OS、iOS でサポートされます。

flash.media.StageVideo クラスは、その目的と機能において flash.media.Video と同じように動作します。実際 StageVideo がサポートするコーデックとフォーマットは Video オブジェクトとまったく同じです。



StageVideo は従来の Video オブジェクトをおとしめたり阻害するものではありません。StageVideo は、DisplayList の下にある、Video のハードウェアアクセラレーション版と見なすことができます。

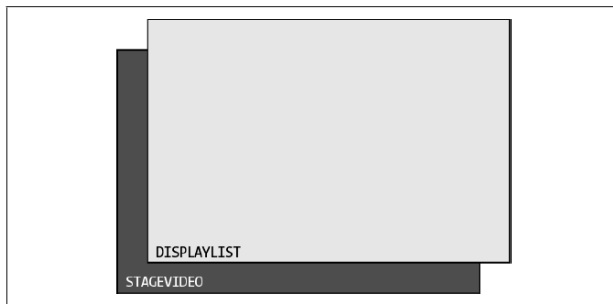


図 4-2: StageVideo は従来の DisplayList の直下に存在する

モバイル AIR プロジェクトで StageVideo を使用するには、`flash.media.StageVideo` クラスをインポートする必要があります。ただし実際にはこのクラスのインスタンスは作成しません。StageVideoAvailabilityEvent.STAGE\_VIDEO\_AVAILABILITY イベントをステージで監視し、特定のデバイスで StageVideo が使用できるようになったことが確認できたら、次へ進みます。

StageVideo オブジェクトは、ステージの `stageVideos` プロパティの位置に、配列のシンタックス (`stage.stageVideos[0]`) を使って割り当てます。StageVideo クラスを使って実際にビデオを再生するときには、標準的な Video クラスを使ったときとまったく同じ方法で行います。StageVideo が使用できない場合、これは容易な代替手段となるので、うまい方法だと言えます。



`stage.stageVideos` 配列の長さはデバイスの性能によって変わりますが、StageVideoAvailabilityEvent.STAGE\_VIDEO\_AVAILABILITY によって、StageVideo が使用できるようになったことが報告されたら、少なくとも1つ、ビデオ再生に使用できる位置が確保できます。

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

```
import flash.display.StageScaleMode;
import flash.events.NetStatusEvent;
import flash.events.StageVideoAvailabilityEvent;
import flash.media.StageVideo;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.text.TextField;
import flash.text.TextFormat;
import flash.geom.Rectangle;
```

```
[SWF(backgroundColor="#000000")]
```

```
public class StageVideoMobile extends Sprite {

    private var traceField:TextField;
    private var stageVideo:StageVideo;
    private var connection:NetConnection;
    private var stream:NetStream;
    private var streamClient:Object;

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

        generateDisplayObjects();
        performOperations();
    }

    protected function generateDisplayObjects():void {
        var defaultFormat:TextFormat = new TextFormat();
        defaultFormat.font = "Arial";
        defaultFormat.size = 30;
        defaultFormat.color = 0xFFFFFFFF;

        traceField = new TextField();
```

```

        traceField.backgroundColor = 0x000000;
        traceField.alpha = 0.7;
        traceField.width = stage.stageWidth;
        traceField.background = true;
        traceField.defaultTextFormat = defaultFormat;
        addChild(traceField);
    }

    protected function performOperations():void {
        stage.addEventListener(StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY,
availabilityChanged);
    }

    protected function availabilityChanged(e:StageVideoAvailabilityEvent):void {
        if(e.availability == "available" && stageVideo == null){
            stageVideo = stage.stageVideos[0];
            createConnection();
        }
        traceField.text = "StageVideo => " + e.availability + "\n";
    }

    protected function createConnection():void {
        streamClient = new Object();
        streamClient.onBWDone = onBWDone;
        streamClient.onMetaData = onMetaData;
        streamClient.onXMPData = onXMPData;
        streamClient.onPlayStatus = onPlayStatus;

        connection = new NetConnection();
        connection.client = streamClient;
        connection.addEventListener(NetStatusEvent.NET_STATUS, monitorStatus);
        connection.connect(null);

        beginStreaming();
    }

```

```

protected function monitorStatus(e:NetStatusEvent):void {
    if(e.info.code == "NetStream.Buffer.Full"){
        stageVideo.viewPort = new Rectangle(0, 0, stage.stageWidth,
stage.stageHeight);
    }
}

protected function beginStreaming():void {
    stream = new NetStream(connection);
    stream.client = streamClient;
    stream.addEventListener(NetStatusEvent.NET_STATUS, monitorStatus);
    stageVideo.attachNetStream(stream);
    stream.play("assets/FlashRuntimes.mp4");
}

public function onBWDone():void {}
public function onMetaData(e:*):void {}
public function onXMPData(e:*):void {}
public function onPlayStatus(e:*):void {}

}
}

```

このクラスは、StageVideoをサポートするモバイル OS 向けにコンパイルする限り、ビデオは完全なハードウェアアクセラレーションを使って、図 4-3 に示すように再生されます。なおアプリケーション記述ファイルの <renderMode>は direct です。

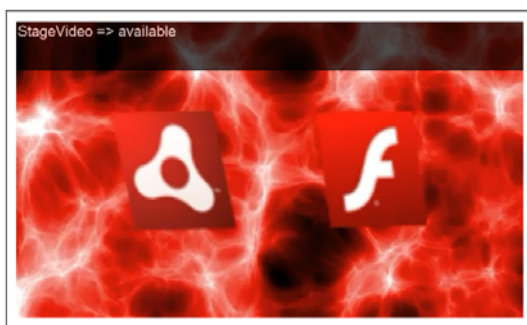


図 4-3: タブレット OS での StageVideo の再生

## 5章:ビデオとオーディオの強化

Adobe AIR はこれまで、VP6 と Spark に加え、業界標準の H.264 のデコードと再生をサポートしてきました。AIR 3 では、キャプチャしたビデオをランタイム自体で H.264 にエンコードすることができます。これはビデオをキャプチャし配信するデスクトップ向けアプリケーションのさらなる可能性を開く機能です。H.264 のエンコーディングに加え、G.711 形式で動作するオーディオコーデックも利用できます。

### H.264/AVC ソフトウェアエンコーディング

AIR 3 では、ランタイム内で H.264 ビデオストリームをエンコードすることができます。AIR の前バージョンでは H.264 のデコードが行えたので、エンコードに関する機能を追加することで、業界標準形式を使った録画と配信を行うアプリケーションを構築することができます。



H.264/AVC ソフトウェアエンコーディングはデスクトップ環境でのみ可能です。モバイルデバイスでは、ストリームのエンコードに割く CPU 量の関係で、この機能は利用できません。

次のサンプルをコンパイルし効率的に実行するには、Flash Media Server のインストールを推奨します。



Flash Media Server の商用使用権を持っていない方のために、Adobe はこのサンプルのテストに使用できるフリーの開発者版を提供しています (<http://www.adobe.com/jp/downloads/>)。FMS には Windows 版と Linux 版があります。

### AIR 3 内での H.264 エンコーディング

AIR 3 で H.264 を使ってビデオストリームをエンコードするには、新しい `H264VideoStreamSettings` クラスと関連オブジェクトを使用します。`H264VideoStreamSettings` インスタンスを構成するときには、エンコーディングの具体的なプロファイルとレベルが設定できます。これは、ベースラインエンコーディングのみをサポートする特定のデバイスのターゲット指定に有用です。`H264VideoStreamSettings` インスタンスが構成できたら、それを `NetStream` インスタンスの `videoStreamSettings` プロパティに割り当て、後は通常の処理を行います。

```

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 = 0xFFFFFFFF;

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 {}

}
}

```

ローカルマシンに Flash Media Server をインストールしそこで実行している場合、ストリームがパブリッシュされ、ビデオのエンコードに使用するコーデックは H.264 であるとする状態メッセージが、カメラフィードのプレビューとともに表示されます。この例では生のカメラ出力を表示するだけで、エンコードされたストリームは表示していません。



図 5-1 : AIR での H.264 エンコーディング

**訳注:**

connection.connect("rtmp://localhost/live");は、AIR を実行するマシンと同じマシンで稼働する FMS の live アプリケーションに接続する、という意味です。FMS のアプリケーションは、FMS をインストールしたフォルダ内の applications フォルダ内にフォルダを作成することで作成されます (C:\Program

Files¥Adobe¥Flash Media Server 4.5¥applications¥live など)。この例で使われている live アプリケーションはデフォルトで applications フォルダに作成されるので、アプリケーションフォルダを作成しなくてもそのまま使用できます。

また、LAN 内の FMS に接続するには、connection.connect(“rtmp://IP アドレス/live”);のようにします。Windows 7 などでは OS のファイアーウォールによって接続が拒絶されるので、テスト中には一時的にファイアーウォールをオフしておきます。

### AIR 3 での H.264 ストリームの読み取り

H.264 のデコードは AIR の最初のリリースから可能でした。H.264 にエンコードされたストリームやファイルを AIR 3 で再生するには、これまで通りの手順を使用するだけです。まず NetConnection を作成し Flash Media Server への接続を確立します。サーバーが同じマシンにある場合には、rtmp://localhost/live を使用します。次に NetStream オブジェクトを使ってパブリッシュ済みのストリームを再生する要求を出します。

```
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 = 0xFFFFFFFF;  
  
    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 {}
}
}

```

このクラスを AIR にコンパイルします。前のコードサンプルによってエンコードされるストリームの Flash Media Server へのパブリッシュが成功している場合には、ネイティブにエンコードされた H.264 ストリームが図 5-2 の右に示すように Video オブジェクトにレンダリングされます。

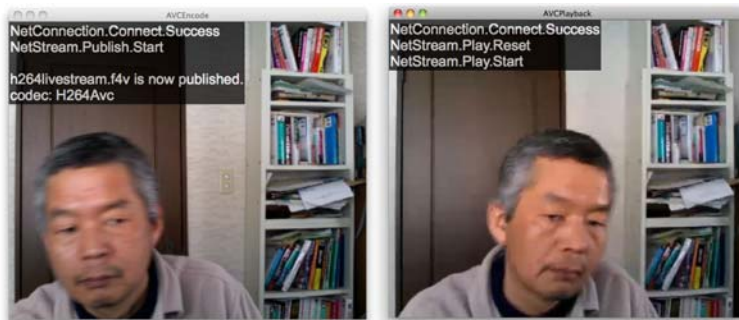


図 5-2: 左がライブでパブリッシュ中の AIR アプリ。右がその H.264 ビデオを再生する AIR アプリ。右のアプリは当然のことながら左よりも遅れて再生される。

#### 電話通信向けの G.711 オーディオ圧縮

AIR 3 にはオーディオ用の G.711 コーデックのサポートも含まれています。G.711 は実は古く、1972 年に”Pulse Code Modulation”(PCM、パルス符号変調、アナログ信号をデジタルに変換する方式)として公式に規格化されています。G.711 の圧縮アルゴリズムは2種類あり、両方とも AIR で使用できます。

- SoundCodec.PCMA は G.711 A-law コーデックを指定します(ヨーロッパやそのほかの国で使用)。
- SoundCodec.PCMU は G.711  $\mu$ -law コーデックを指定します(北アメリカと日本で使用)。



G.711 は主に、電話通信やアプリケーションベースの SIP (Session Initiation Protocol、セッション確立プロトコル) で使用されます。特に通話に適しており、数多くのシステムでサポートされています。

これは、オーディオプロジェクトで `flash.media.SoundCodec` を使うことで実現できます。Microphone オブジェクトの構成では、`codec` プロパティに `SoundCodec.PCMU` か `SoundCodec.PCMA` 定数が指定できます。これにより音源からのオーディオが確実に G.711 として処理されるようになります。

```
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 = 0xda0000;  
  
        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 = 0xFFFFFFFF;  
  
    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);
    }
}
}

```

図 5-3 はこのクラスの実行結果です。

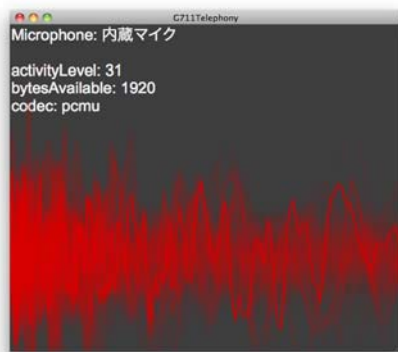


図 5-3: G.711  $\mu$ -law コーデックでエンコードされたオーディオデータ

## 6章: 追加されたモバイルデバイスのハードウェアサポート

モバイルデバイス向けの開発で魅力的なもの1つに、小さな個体に組み込まれた数多くの入出力ハードウェアがあります。アプリケーションでプログラミングからデバイスのカメラやスピーカー、マイクに接触できることは、希望する機能というよりも、多くのモバイルアプリケーションにとってはそれが期待され必要とされる機能です。AIR 3 では前バージョンと比べて、これらのハードウェアを制御する機能がはるかに多く提供されています。

### CameraPosition API (モバイル)

デバイスのフロントカメラへのアクセスはこれまでも iOS と TabletOS プラットフォームでは可能でしたが、AIR 3 では Android にも拡張されました。

デバイスの複数のカメラを区別するには、`flash.media.Camera` クラスの新しい `position` プロパティを調べます。また定数のセットを指定する新しい `flash.media.CameraPosition` クラスを使用すると、現在のカメラの位置が特定できます。



`CameraPosition.FRONT` = 前向きデバイスカメラ(自分撮り)

`CameraPosition.BACK` = 後ろ向きデバイスカメラ

`CameraPosition.UNKNOWN`

次の例では、デバイスのカメラの数を調べて各 `Camera` オブジェクトを走査し、`position` プロパティを調べて、前向きカメラを特定しています。希望するカメラが配置できたら、その映像を画面に表示しています。

```
package {  
    import flash.display.Sprite;  
    import flash.display.StageAlign;  
    import flash.display.StageScaleMode;  
    import flash.media.Camera;  
    import flash.media.CameraPosition;  
    import flash.media.Video;  
  
    [SWF(backgroundcolor="#000000")]
```

```

public class FrontCamera extends Sprite {

    private var video:Video;
    private var camera:Camera;

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

        if(Camera.isSupported){
            setupCamera();
        }
    }

    private function setupCamera():void {
        for(var i:int=0; i<Camera.names.length; ++i) {
            camera = Camera.getCamera(String(i));
            if (camera.position == CameraPosition.FRONT){
                camera.setMode(stage.stageWidth, stage.stageHeight, 30);
                camera.setQuality(60000, 80);
                setupVideo();
                break;
            }
        }
    }

    private function setupVideo():void {
        video = new Video(stage.stageWidth, stage.stageHeight);
        video.attachCamera(camera);
        addChild(video);
    }
}

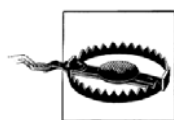
```

前向きカメラはほとんどの場合で、図 6-1 に示すようにユーザーの顔のキャプチャに使用されます。



図 6-1 : Android の前向きカメラ

これは、ビデオチャットアプリケーションのような、AIR アプリケーションで特定のカメラが必要な場合に便利です。



Android をターゲットにするときには、アプリケーション記述ファイルでカメラに関する適切な権限を与える必要があります。

```
<uses-permission android:name="android.permission.CAMERA" />
```

#### デバイススピーカーの制御 (モバイル)

モバイルデバイスには一般的に、オーディオを伝送するハードウェアメカニズムが2つ別個に存在します。1つは音声データの伝送に使用されるスピーカーで、通話などに使用されます。もう1つは、ゲームやアプリケーション、音楽、ビデオ再生などほとんどの事柄に使用されるスピーカーです。

AIR デベロッパーは、個々のアプリケーションの必要性に応じて、デバイスの特定のハードウェアスピーカーをターゲットとすることができます。これは、`flash.media.SoundMixer` クラスの `audioPlaybackMode` プロパティを、`flash.media.AudioPlaybackMode` クラスの `AudioPlaybackMode.MEDIA` か `AudioPlaybackMode.VOICE` 定数に設定することで行います。



もう1つ、`SoundMixer.useSpeakerphoneForVoice` プロパティを使用すると、デフォルトの音声の振る舞いをオーバーライドできるという面白い機能も追加されています。

```

package {
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;
    import flash.media.AudioPlaybackMode;
    import flash.media.Sound;
    import flash.media.SoundMixer;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFormat;

    [SWF(backgroundColor="#000000")]

    public class SoundSpeaker extends Sprite {

        private var traceField:TextField;
        private var sound:Sound;
        private var id3:Boolean;

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

            generateDisplayObjects();
            setupSoundMixer();
        }

        protected function generateDisplayObjects():void {
            var defaultFormat:TextFormat = new TextFormat();
            defaultFormat.font = "Arial";
            defaultFormat.size = 40;
            defaultFormat.color = 0xFFFFFFFF;

            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);
    }

    private function setupSoundMixer():void {
        SoundMixer.audioPlaybackMode = AudioPlaybackMode.MEDIA;
        // SoundMixer.audioPlaybackMode = AudioPlaybackMode.VOICE;
        SoundMixer.useSpeakerphoneForVoice = false;

        setupSoundandLoad();
    }

    private function setupSoundandLoad():void {
        sound = new Sound(new URLRequest("assets/drowning.mp3"));
        sound.addEventListener(Event.ID3, id3Loaded);
        sound.play();
    }

    protected function id3Loaded(event:Event):void {
        if(!id3){
            traceField.appendText("Playing: " + sound.id3.songName + "\n");
            traceField.appendText("From the album: " + sound.id3.album + "\n");
            traceField.appendText("By the artist: " + sound.id3.artist + "\n");
            traceField.appendText("Released in: " + sound.id3.year + "\n");
            traceField.appendText("Audio      Playback      Mode:      "      +
SoundMixer.audioPlaybackMode);
            id3 = true;
        }
    }
}

```

```
}  
}
```

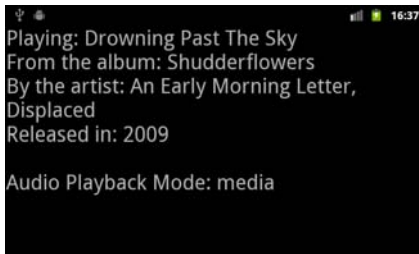


図 6-2: デバイスピーカーの制御

### iOS でのバックグラウンドオーディオ再生のサポート

AIR 3 では iOS に対し、アプリケーションにフォーカスがないとき、アプリケーションのオーディオ再生をつづけるべきかどうかを知らせることができます。これは、AIR のアプリケーション記述ファイルから行います。

アプリケーション記述ファイルの iPhone ノード内の infoAdditions ノードに移動し、新しい<key>を "UIBackgroundModes" の値で挿入します。この<key>のすぐ下に、ネストした<string>ノードを持つ<array>ノードを記述します。<string>ノードの値は "audio" です。

```
<!-- iOS specific capabilities -->
```

```
<iPhone>
```

```
  <InfoAdditions>
```

```
    <![CDATA[
```

```
      <key>UIDeviceFamily</key>
```

```
      <array>
```

```
        <string>1</string>
```

```
        <string>2</string>
```

```
      </array>
```

```
      <key>UIStatusBarStyle</key>
```

```
      <string>UIStatusBarStyleBlackOpaque</string>
```

```
      <key>UIRequiresPersistentWiFi</key>
```

```
      <string>YES</string>
```

```
<key>UIBackgroundModes</key>
<array>
  <string>audio</string>
</array>
]]>
</InfoAdditions>
<requestedDisplayResolution>high</requestedDisplayResolution>
</iPhone>
```

これにより、AIR for iOS モバイルアプリケーションのバックグラウンドでの実行中、どのアプリケーションのオーディオも再生をつづけることができるようになります。

## 7章: 追加されたデータ転送機能

堅牢なアプリケーションの開発プラットフォームには、堅牢なデータ転送オプションが必要です。ルートレベルの XML サポートは Adobe AIR の最初のリリースで導入されていたので、アプリケーションでは XML をベースにした形式を利用することができました。AIR 3 では、ネイティブの JSON ハンドラのサポートが追加され、さまざまなシステムとのやりとりにソケットを使用する際の大きな強化が図られています。

### ネイティブでの JSON サポート

JavaScript Object Notation (JSON) は構造化されたデータセットの転送に使用される人気の高い方法で、AIR アプリケーションへのデータの入出力に使用できます。ActionScript 3.0 が登場したときから、プロジェクトで JSON が容易に使用できるサードパーティ製のライブラリがありましたが、AIR 自体のコアの機能でなかったため、パフォーマンスの面で少し問題がありました。



JSON は、ActionScript の XML や Array クラスと同様に、トップレベルのクラスなので、アプリケーションで使用する際、わざわざインポートする必要はありません。

次の JSON オブジェクトは、一連の名前と値のペアを使って人物を表しています。JSON オブジェクトはネストでき、このシンタックスでは配列構造も含むことができます。これは実に柔軟な方法です。

```
{
  "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()

次のコード例ではこの JSON ファイルを使っています。ロードした JSON オブジェクトから JSON.parse()を使って値を解析し、AIR アプリケーションの基本的な TextField にその値を表示しています。

```
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 = 0xFFFFFFFF;

    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");
    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");
    }
}
}
}
}

```

ご覧のように、JSON の処理は ActionScript の XML の処理とよく似ています。読み込んだ JSON を解析し、そのデータを TextField に出力してレンダリングすると、図 7-1 のようになります。

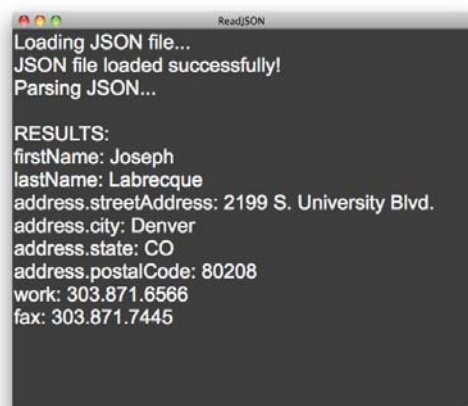


図 7-1: 解析した JSON の出力

JSON.stringify()

AIR プロジェクトで実際に JSON を記述する (ActionScript のオブジェクトを JSON にする) には、次のコード例で示すように、JSON.stringify() メソッドを使用します。

```

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 = 0xFFFFFFFF;
```

```
        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());
    }
}
}

```

JSON.stringify()メソッドは、集めて作った ActionScript オブジェクトを完全な JSON シンタックスに変換します。ただしオブジェクト内のデータ型はすべて、JSON に変換できる型である必要があります。



有効なデータ型には、Array、String、Number、Boolean、null があります。

このメソッドに渡している1つめの引数は、変換したい実際のオブジェクトです。2つめの引数は、変換するオブジェクトのキー/値のペアの変換やフィルタリングに使用できる、オプションの関数か配列です。これはたとえば、あるキー/値のペアを実際の出力から除外したい場合に使用できます。

最後の引数では、判読性を高めるために、データの各ピースの前後に挿入するスペース量が指定できます。この例では数値 4 を渡して、各エントリの前に4つの空白文字を加えるように指定しています。これにより図 7-2 に示すように、空気が生まれ、出力結果が読みやすくなります。

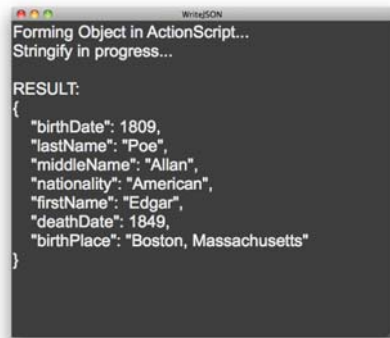


図 7-2: JSON.stringify()の結果

### ソケットの ProgressEvent

flash.net.Socket クラスは、Adobe AIR の最初のリリースから使用できました。ActionScript のソケットを使用すると、flash.events.ProgressEvent を使って、つねにイベントにアクセスし、ソケットを通して入ってくる入力データの状況を監視することができます。しかし今までは、ソケット接続を通して送られる出力データの監視はほぼ可能でした。

AIR 3 では、flash.events.OutputProgressEvent クラスにアクセスでき、ソケット接続上を送信中のバイトと合計バイト両方を監視することができます。これは、ユーザーに進行状況を示すインジケータや、アプリケーション内での連続するイベントの表示、または単にソケットでデータ処理が完了したことの検証などに使用できます。

次の例では、基本的なソケット接続を使用して adobe.com にソケット接続し、flash.events.ProgressEvent と flash.events.OutputProgressEvent 両方を監視します。

```
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 = 0xFFFFFFFF;
```

```
        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());
}

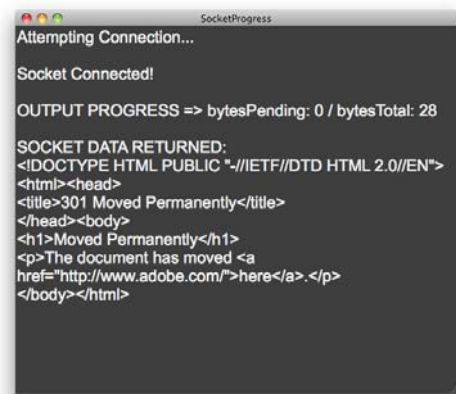
protected function socketOutputProgress(e:OutputProgressEvent):void {
    traceField.appendText("OUTPUT PROGRESS => bytesPending: " + e.bytesPending
+ " / bytesTotal: " + e.bytesTotal + "¥n¥n");
}

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");
}
}
}

```

図 7-3 はこのコードの実行結果です。ご覧のように、AIR 3 では、接続を確立したソケットから返されるバイトの監視と、このトランザクションから期待される合計バイトにアクセスすることができます。



```
SocketProgress
Attempting Connection...
Socket Connected!
OUTPUT PROGRESS => bytesPending: 0 / bytesTotal: 28
SOCKET DATA RETURNED:
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a
href="http://www.adobe.com/">here</a>.</p>
</body></html>
```

図 7-3: ソケットの出力進行状況

## 8章:ランタイムの強化

AIR 3 で強化された言語とランタイムの中には、さまざまな新しいクラスやメソッド、プロパティ、アーキテクチャがありますが、その目的は、ランタイムとその使用に関し、物事がより容易により拡張的に、より速く行えるようにすることです。本章では、言語とランタイムのさまざまな改良点と、プロジェクトで容易に構築できる一般的な実装例を見ていきます。

### ActionScript Native Extension

AIR 2 リリースでは、ホスト OS のネイティブプロセスとやりとりできるオプションがデスクトップ AIR に与えられていました。これはすばらしい機能で、アプリケーションと実行プロセス間でメッセージをやりとりすることによって、AIR アプリケーションが拡張できるということを意味していました。しかし理想的なものではなく、またモバイル AIR プロジェクトでは利用できませんでした。

AIR 3 では、これが ActionScript Native Extension (ANE) として、かなり強力になっています。ANE を使用すると、アプリケーションの大部分は ActionScript で記述しつつ、別のある部分はネイティブ OS の言語で提供して AIR ランタイムを拡張し、これまで触れることのできなかつた可能性を広げることができます。



コードの大部分は ActionScript 以外の言語で記述することになります。

たとえば、AIR ランタイムは iOS や Android で加速度メータなどのセンサーをネイティブでサポートしますが、デバイスに備わるジャイロスコープや気圧計、バイブレーションといった構成要素を利用する方法は提供していません。同様に、デバイスのコンタクトリスト(連絡先のリスト)や通知の送信といった OS のデータや機能性も AIR から完全に隠されたままです。ANE を使用すると、AIR の機能性を拡張するコードを記述して、アプリケーション内のこういった事柄にアクセスすることができます。



本章では、モバイル向けの ActionScript Native Extension の使用にフォーカスしますが、ANE はデスクトップでも使用できます。

次の例では、Android 用に開発された Vibrator ANE を使用します。ANE は SWC と似て、なじみのある方法でプロジェクトに含めます。次の例では Flash Builder 4.6 を使って、AIR for Android プロジェクトに

Vibrator ANE をインクルードします。ActionScript モバイルプロジェクトを作成し、[パッケージエクスプローラ]でプロジェクト名をクリックして[ファイル]→[プロパティ]を選びます。[プロパティ]ダイアログが開くので、左の枠で[ActionScript ビルドパス]を選択し、[ネイティブエクステンション]タブをクリックします。ここで[ANE を追加]ボタンをクリックすると、プロジェクトに組み込みたい ANE ファイルが指定できます。



追加した ANE パッケージ要素の左の三角マークをクリックして展開すると、ターゲットと ANE の制限が表示されます。

訳注:

ANE ファイルは Adobe の「Vibration native extension sample」ページ (<http://www.adobe.com/devnet/air/native-extensions-for-air/extensions/vibration.html>)にあるサンプルファイル (Vibration.zip) に含まれているものを使用します (Vibration/VibrationNEDeliverables/ReadyToUseExtension/com.adobe.extensions.Vibration.ane)

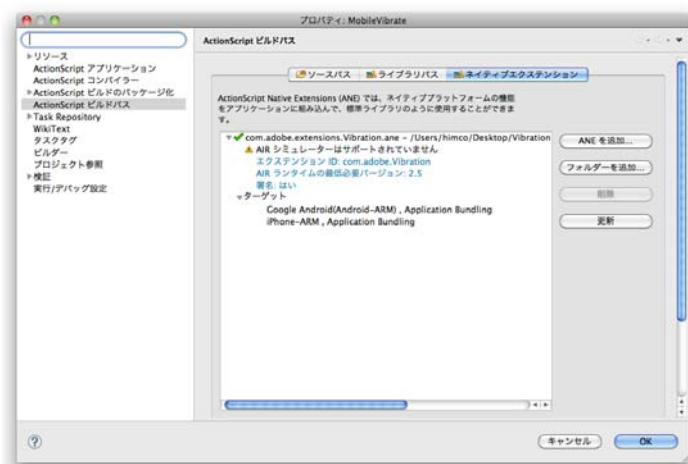


図 8-1: Vibrator ANE をモバイルプロジェクトに追加した

つづいて、ANE をビルド設定にパッケージングする必要があります。前の[プロパティ]ウィンドウで、左の[ActionScript ビルドのパッケージ化]を選択し、左の三角マークをクリックして展開して、[Google Android]を選択します。[ネイティブエクステンション]タブをクリックします。ここでは各プラットフォーム(今の場合は Android)向けのビルドに含めたい ANE ファイルを選択します。ANE と SWC には、ANE はこの方法で別個に含める方法しかない、という大きな違いがあります。

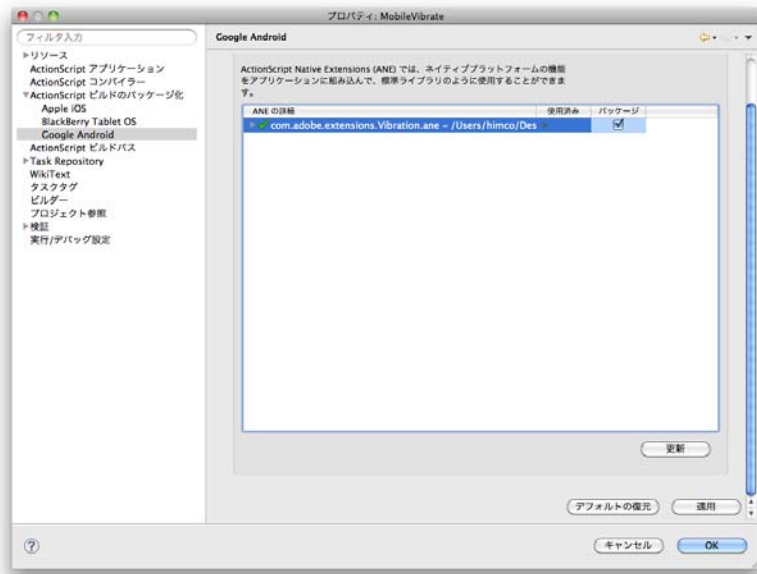


図 8-2: ANE をビルドパッケージにインクルードする

Android で ANE を使用するには、アプリケーション記述ファイルの<android>ノードにいくつかの権限を追加する必要があります。この例ではバイブレーションの権限を追加します。

<android>

```
<manifestAdditions><![CDATA[
    <manifest android:installLocation="auto">
        <uses-permission android:name="android.permission.INTERNET"/>
        <uses-permission android:name="android.permission.VIBRATE"/>
    </manifest>
]]></manifestAdditions>
```

</android>

プロジェクトに追加するエクステンションについては、次に示すようにアプリケーション記述ファイルのルートで宣言する必要があります。希望する場合、Flash Builder 4.6 はこの記述を自動的に追加します。

<extensions>

```
<extensionID>com.adobe.Vibrator</extensionID>
```

</extensions>



準備が多すぎるように思えますか？ しかしこれはプロジェクトで1回行えばよいだけです。

ActionScript Native Extension によってもたらされるパワーは、この小さな面倒をこなすだけの十分に価値のあるパワーです。

プロジェクトでこの Vibrator ANE を有効に使用するには、Vibrator クラスをインポートする必要があります。次いで新しい Vibrator オブジェクトをインスタンス化し、デバイスのバイブレーションを引き起こしたいタイミングで `Vibrator.vibrate()` を呼び出します。引数にミリ秒の数値を渡すと、デバイスに対してその時間の長さ分だけ振動するように命令できます。次のコード例では、ユーザーがステージにタッチしたらバイスを振動させ、`flash.utils.getTimer` クラスからの時間とともにメッセージを表示しています。

```
package {
    import com.adobe.nativeExtensions.Vibration;

    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.MouseEvent;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.getTimer;

    [SWF(backgroundColor="#000000")]

    public class MobileVibrate extends Sprite {

        private var traceField:TextField;
        private var vibrator:Vibration;

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

```
        generateDisplayObjects();
        performOperations();
    }
```

```
protected function generateDisplayObjects():void {
    var defaultFormat:TextFormat = new TextFormat();
    defaultFormat.font = "Arial";
    defaultFormat.size = 36;
    defaultFormat.color = 0xFFFFFFFF;

    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 {
    if(Vibration.isSupported){
        vibrator = new Vibration();
        stage.addEventListener(MouseEvent.CLICK, performVibration);
        traceField.appendText("Click anywhere to vibrate device...¥n¥n");
    }else{
        traceField.appendText("Vibrator not supported!!!");
    }
}
```

```
protected function performVibration(e:MouseEvent):void {
    vibrator.vibrate(1000);
    traceField.appendText("VIBRATE! [" + getTimer() + "]¥n");
}
```

```
}  
}
```

**訳注:**

原書では Vibrator クラスを使う記述が書かれていますが、ダウンロードしたファイルのクラスは Vibration だったので、ここでは Vibration を使っています。原書のサンプルには Vibrator クラスも ANE ファイルも含まれていません。

このコードを APK にコンパイルし Android で実行して画面をタップすると、図 8-3 に示すような結果が得られます。

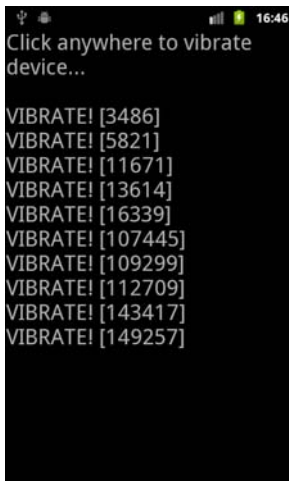


図 8-3: Android の Vibration ANE

**訳注:**

ここでは Flash Builder 4.6 を使った例が紹介されていますが、Flash CS5.5 でもやってやれないことはありません。次はわたしがとった手順です。

- 1) 通常通り、FLA ファイル (MobileVibrate fla) とドキュメントクラス (MobileVibrate.as) を作成します。
- 2) ダウンロードしたサンプルファイル (Vibration.zip) に含まれている Vibration/VibrationNEDeliverables/ReadyToUseExtension/com.adobe.extensions.Vibration.ane をコピーし、com.adobe.extensions.Vibration.swc という拡張子に変更します。
- 3) FLA ファイルの [ActionScript3.0 の詳細設定] で [ソースパス] に、ダウンロードしたサンプルファイル ( Vibration.zip ) に含まれている Vibration/VibrationNEDeliverables/VibrationActionScriptLibrary/src フォルダを (ここに Vibration.as が含まれています) 追加します。[ライブラリパス] に、2) で作成した com.adobe.extensions.Vibration.swc を追加します。
- 4) 表示される SWC ファイルの左の三角マークをクリックして展開し、[リンクの種類] をダブルクリックして開

き、[ライブラリパスアイテムオプション]の[リンクの種類]で[コードにマージ]を[外部]に変更します。

5) [AIR for Android 設定]の[権限]タブで[アプリケーション記述ファイルへの権限およびマニフェストの追加を手動で管理します]チェックボックスを選択して有効化します。

6) アプリケーション記述ファイルを開き、前述したパーミッション設定とエクステンション設定を追加します。

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

```
<extensions>
```

```
<extensionID>com.adobe.Vibrator</extensionID>
```

```
</extensions>
```

7) Flash CS5.5 でプレビューし、SWF ファイル(MobileVibrate.swf)を作成します。

8) AIR 3 SDK フォルダにある bin フォルダを開きます。ここに AIR のパッケージングに使っている.p12 ファイル、7)で作成した SWF ファイルとアプリケーション記述ファイルをコピーします。

9) さらに bin フォルダ内に libs フォルダを作成し、ここに com.adobe.extensions.Vibration.ane をコピーします。

10) コマンドラインを開き、bin フォルダに移動して(cd コマンド)、次の adt コマンドを実行します(コマンドは実際には1行です)。これにより MobileVibrate.apk ファイルがこの bin フォルダに作成されます。

```
./adt -package -target apk -storetype pkcs12 -keystore mycert.p12 MobileVibrate.apk  
MobileVibrate-app.xml MobileVibrate.swf -extdir libs
```

11) Android デバイスを接続し、次のコマンドを実行します。1つめのコマンドでデバイスが認識され、2つめのコマンドで MobileVibrate.apk ファイルをデバイスにインストールします。

```
adb devices
```

```
adb install -r MobileVibrate.apk
```

12) デバイスでアプリケーションを起動します。Flash Builder で作成したアプリケーションと同じように振動します。bin フォルダにコピーしたり作成したファイルは全部削除しておきます。



ActionScript Native Extensions (ANE)を使用するときにはまた、Android マーケットライセンス ( <https://support.google.com/androidmarket/developer/bin/answer.py?hl=ja&answer=186113> ) などを利用して、Android マーケットを通して公開するアプリケーションの使用許可を合理的に適用することができます。

Adobe と Flash Platform コミュニティによる多くの ANE ファイルがすでに公開されており、デベロッパーはこれをアプリケーションで自由に使うことができます。ネイティブコードの記述に習熟されている方には、ANE の作成とパッケージ化の処理に関して詳しく書かれた Oliver Goldman の記事 (<http://www.adobe.com/devnet/air/articles/extending-air.html>) が参考になります。

#### Captive Runtime のサポート

これまで AIR に最も頻繁に求められてきた機能があるとすると、これがおそらくそれに当たるでしょう(今はそうでなくても、いずれそうなります)。Captive Runtime は、ユーザーのシステムに AIR ランタイムを別にインストールする必要のない方法で、アプリケーションをパッケージ化します。これは概念的に言うと、AIR ランタイムがアプリケーションといっしょにパッケージ化され完全なパッケージになる iOS での AIR の働きに似ています。



Captive Runtime は現在、Windows と OSX、Android でサポートされています。AIR for iOS をパッケージ化するときには、iOS の外部ランタイムに対する制限により、AIR ランタイムがつねにバンドルされます。

AIR ランタイムをアプリケーションとともに埋め込むメリットの1つは明白です。デスクトップやデバイスに AIR がインストールされていない場合でも、ユーザーはそれを追加的にダウンロードしインストールする面倒がなくなります。また、厳しく制御されたネットワークにつながり、ランタイムがインストールできない会社のマシンのユーザーにもメリットがあります。とはいえこの方法にも、ランタイムの埋め込みにより配布するパッケージのファイルサイズが大きくなるというデメリットがあります。この増分はプラットフォームによっては5から8メガバイトほどになると思われます。したがってデベロッパーはその配布方法を決めるときには、それぞれの良い面と悪い面両方を検討する必要があります。しかしこのような選択肢があることは、選択肢がまったくないよりもましです。

Flash Builder 4.6 を使ってデスクトップ向けアプリケーションにこの機能を使用するには、[プロジェクト]→[リリースビルドのエクスポート]を選択します。[署名済みキャプティブランタイムアプリケーション]を選択し、[次へ]をクリックします(図 8-4 参照)。

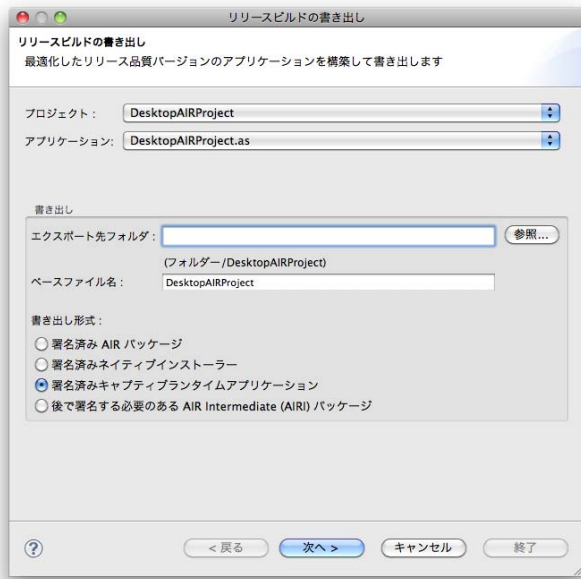


図 8-4: AIR のデスクトップ用キャプティブルランタイム

Flash Builder 4.6 を使って Android アプリケーション内に AIR ランタイムを埋め込むには、[プロジェクト]→[リリースビルドのエクスポート]を選択します。[各ターゲットプラットフォーム用の署名済みパッケージ]を選択し、[次へ]をクリックします(図 8-5 参照)。

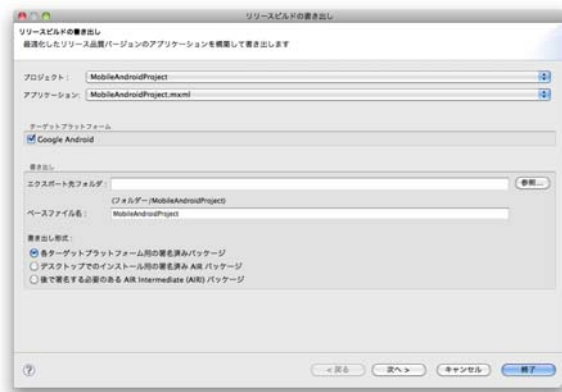


図 8-5: Android プロジェクトの書き出し

Android 用の[パッケージ化設定]が表示されるので、[デプロイメント]タブをクリックし、[書き出しオプション]から[キャプティブルランタイムアプリケーションを書き出し]を選択します(図 8-6 参照)。

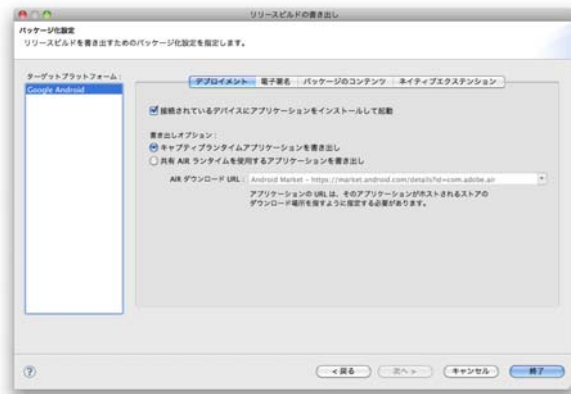


図 8-6: Android でのランタイムの埋め込み



Android やデスクトップ向けアプリケーションのパッケージ化はまた、埋め込みランタイムオプションを使ったコマンドラインからでも、手動で行うことができます。

#### Android のカラー深度設定 (モバイル)

AIR 3 for Android をターゲットにするときには、アプリケーションを 16 ビットか 32 ビットカラーモードのどちらかでコンパイルしたいかを指定することができます。そのためにはアプリケーション記述ファイルを修正します。



これは `<renderMode>` が `"cpu"` か `"auto"` に設定されている場合のみ機能します。 `"gpu"` モードでは 32 ビットカラーモードが必要なので、この設定は効果的ではありません。

カラー深度を指定するには、`<initialWindow>` に移動し、ネストされた `<renderMode>` を探します。値は必ず `"auto"` か `"cpu"` にします。 `<colorDepth>` には 16bit か 32bit を指定します。

```
<colorDepth>32bit</colorDepth>
```



指定しない場合にはデフォルトで、AIR 2.7 以前では 16 ビットカラーが、AIR3 以降では 32

ビットカラーが使用されます。

## ガベージコレクションへのアドバイス

flash.system.System クラスには、pauseForGCIfCollectionImminent()という名前の新しいメソッドがあります。このメソッドは、希望する緊急度の値を決める引数を1つ取ります。これはオプションで、0 から 1 までの Number です。低い値ほどガベージコレクションによる一掃の必要性が低いことを表します。このメソッドを使用すると、ガベージコレクタに対し、一掃が適切になったタイミングでその作業を開始するようにアドバイスすることができます。

次の例では、2つの状態を持った小さな”ゲーム”を作成します。1つめの状態はユーザーがインタラクティブ操作を行うゲームレベルで、時間切れがあります。2つめの状態はレベル間の一時停止している状態を表します。ここでユーザーは、自分の成績を反映したり、次のゲームレベルの準備をする機会が与えられます。適切なタイミングで System.pauseForGCIfCollectionImminent()を呼び出すことで、ガベージコレクタに対して、最良のタイミングでアクションを実行するようにアドバイスすることができます。

```
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 = 0xFFFFFFFF;  
  
    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¥n ユーザーはプレイ中なので、今は GC にアドバイスして混乱を起こさせるべき  
ではない。。。";  
    msg += "¥n¥n ユーザーのゲーム中、もし GC を呼び出すと、CPU インテンシブな多くのプロ  
セスが動作している場合、ゲームは一時的にフリーズする可能性がある。";  
    msg += "¥n¥n もしそうなると、ユーザーの操作は失敗し、視覚的にも混乱することになる。  
";  
  
    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 = "現在の状態:" + stateName;
        traceField.appendText(msg);
    }

    protected function timeUp(e:TimerEvent):void {
        stateName = "LevelComplete";
        System.pauseForGCIfCollectionImminent();
        msg = "¥n¥nSystem.pauseForGCIfCollectionImminent()が呼び出された!";
        msg += "¥n¥nこれは、GCの実行に起因する異常な表示など、おかしな振る舞いを回避
        するために、レベル間で行う ";
        msg += "¥n¥n ユーザー操作やゲームエンジン処理がないときに GC にアドバイスすることで、
        混乱する危険性がかなり現象する。";
        msg += "¥n¥n 実に便利!";
    }
}
}
}

```

ゲーム中にはガベージコレクタを起動させるようなことはしたくありません。なぜなら処理が完了するまで、何秒かゲームがフリーズする可能性があるからです。ガベージコレクタは、ゲームで進行している事柄がなく、レベルの終了時や画面のロード時など、ユーザーが画面に注目していない、ゲームのほかのタイミングで実行する方がずっと良いに決まっています。重要なことは、ユーザー体験を分断しないということです。

今の例では、Timer を使って時間を計り、アクティブなゲーム状態からレベル間の休止状態に移動するタイミングを決めています。この休止状態は System.pauseForGCIfCollectionImminent()を使って、ガベージコレクタにその職務を実行するようアドバイスする絶好のタイミングです。

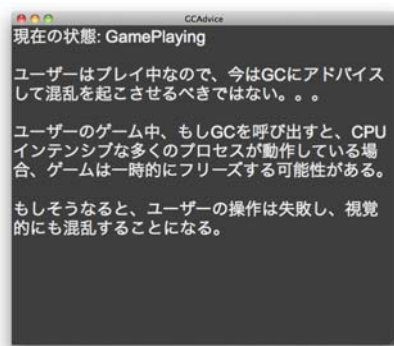


図 8-7: "GamePlaying" 状態中の画面

Timer が時間切れを告げると、ゲームは“LevelComplete”状態に入り、ユーザーは次のレベルに移る前に、少し休憩することができます。このときユーザーはゲームに積極的でなく、基本的に画面上では何も起きていないので、これはガベージコレクションにうってつけのタイミングです。たとえガベージコレクタによるフリーズや同種の視覚的分断が発生したとしても、ユーザーがそれに気がつくことはまずありません。

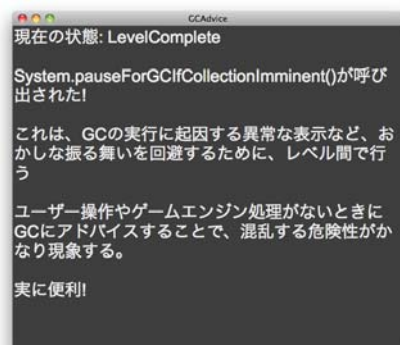


図 8-8: “LevelComplete”状態中の画面



System.pauseForGCIfCollectionImminent()を呼び出すことで、ゲームやアプリケーションでガベージコレクタの実行をアドバイスするタイミングを選ぶことはできますが、実際にそのタイミングで発射されるという保証はありません。

## 9章: Adobe AIR のセキュリティ

セキュリティはすべての環境においてつねに重要な関心事です。プラットフォームが拡大し、全体的な技術の展望が開けるにつれて、注意や追加的なメカニズムを必要とする新しい問題が生じ、そのためにプラットフォームの防御を強固にする必要性が生まれます。AIR 3 では、セキュアなデータ生成メカニズムに関する新しい API や、デスクトップとモバイルでの防御されたビデオのストリーミングに関する拡張されたランタイムサポートなど、多くの新しいまたは更新されたセキュリティメカニズムが備わりました。

### 暗号化されたローカルストレージ(モバイル)

暗号化されたローカルストレージ(ELS)は、デスクトップでは AIR の最初のリリースからサポートされています。モバイルプラットフォームでは使用できませんでしたが、AIR 3 になってモバイルデバイスでも `flash.data.EncryptedLocalStore` クラスが利用できるようになりました。

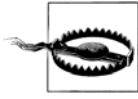


Android では、データは Android のユーザー ID にもとづくファイルシステムセキュリティによって守られます。

モバイルデバイスで `EncryptedLocalStore` を使用するにはまず、`EncryptedLocalStore.isSupported` プロパティを調べて、`EncryptedLocalStore` がサポートされているかどうかチェックします。これが `true` を返す場合には、アプリケーションで `EncryptedLocalStore.setItem()` と `EncryptedLocalStore.getItem()` コマンドが実行できます。

`EncryptedLocalStore.setItem()` を使ってアイテムを ELS に設定するには、少なくともデータを識別する名前とデータ自体の2つの引数を渡す必要があります。データは、ELS に設定する前、`ByteArray` にエンコードしておく必要があります。データを取得するには、`EncryptedLocalStore.getItem()` に識別する名前を渡すだけです。

ELS からあるアイテムを削除するには、`EncryptedLocalStore.removeItem()` に、識別子として使用する任意の名前を渡して呼び出します。`EncryptedLocalStore.reset()` を呼び出すと、すべてのデータがクリアされ、システムから完全になくなります(暗号化されたローカルストア全体がなくなります)。



アプリケーションを削除するとき、アンインストールは暗号化されたローカルストアに保存したデータを消去しません。

```
package {  
    import flash.data.EncryptedLocalStore;  
    import flash.display.Sprite;  
    import flash.display.StageAlign;  
    import flash.display.StageScaleMode;  
    import flash.text.TextField;  
    import flash.text.TextFormat;  
    import flash.utils.ByteArray;  
  
    [SWF(backgroundColor="#000000")]  
  
    public class MobileEncryptedStore extends Sprite {  
  
        private var traceField:TextField;  
  
        public function MobileEncryptedStore() {  
            super();  
            stage.scaleMode = StageScaleMode.NO_SCALE;  
            stage.align = StageAlign.TOP_LEFT;  
  
            generateDisplayObjects();  
            performOperations();  
        }  
  
        protected function generateDisplayObjects():void {  
            var defaultFormat:TextFormat = new TextFormat();  
            defaultFormat.font = "Arial";  
            defaultFormat.size = 24;  
            defaultFormat.color = 0xFFFFFFFF;  
  
            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 = "EncryptedLocalStore.isSupported => " +
EncryptedLocalStore.isSupported + "\n\n";
        if(EncryptedLocalStore.isSupported){
            writeEncryptedLocalStore();
        }
    }

    protected function writeEncryptedLocalStore():void {
        var userName:String = "Joseph.Labrecque@du.edu";
        var passWord:String = "Adobe_AIR_Rocks!!!";

        var userBytes:ByteArray = new ByteArray();
        userBytes.writeUTFBytes(userName);
        EncryptedLocalStore.setItem("username", userBytes);

        var passBytes:ByteArray = new ByteArray();
        passBytes.writeUTFBytes(passWord);
        EncryptedLocalStore.setItem("password", passBytes);

        traceField.appendText("データは EncryptedLocalStore に書き込まれた!\n");
        traceField.appendText("ユーザー名: " + userName + "\n");
        traceField.appendText("パスワード: " + passWord + "\n\n");
        readEncryptedLocalStore();
    }

```

```

protected function readEncryptedLocalStore():void {
    var userData:ByteArray = EncryptedLocalStore.getItem("username");
    var passData:ByteArray = EncryptedLocalStore.getItem("password");

    traceField.appendText("データを EncryptedLocalStore から取得した!¥n");
    traceField.appendText("¥t ユーザー名: " + userData.readUTFBytes(userData.length)
+ "¥n");
    traceField.appendText("¥t パスワード: " + passData.readUTFBytes(passData.length)
+ "¥n¥n");

    clearEncryptedLocalStore();
}

protected function clearEncryptedLocalStore():void {
    EncryptedLocalStore.reset();
    traceField.appendText("EncryptedLocalStore はリセットされた¥n");
    traceField.appendText("機密的なデータはすべてクリアされた!");
}
}
}

```

図 9-1 はこのクラスをデバイスで実行したときの出力結果を示しています。



図 9-1: 暗号化されたローカルストア

プロテクトされた HTTP ダイナミックストリーミングと Flash Access コンテンツ保護サポート(モバイル)

Flash Media Server 4 で導入された HTTP ダイナミックストリーミング(HDS)では、ストリーミングプロトコル

の Real Time Message Protocol (RTMP) ファミリーではなく、Hyper Text Transfer Protocol (HTTP) 上でのライブまたはオンデマンドのメディアストリーミングが行えます。これは、ネットワークで RTMP が使用するポート (通常は 1935) がブロックされている場合、非常に便利です。この技術はこれまで、AIR のデスクトップ版でのみ利用できましたが、AIR 3 ではモバイル版にも拡張されました。



とは言うものの、RTMP 上でのビデオの配信は依然として、最もセキュアで堅牢なストリーミングの方法です。HDS は単に、追加的なオプションとしてコンテンツプロバイダを提供しているだけです。

Adobe Flash Access は、Flash Media Server (FMS) を使って RTMP か HTTP 上で AIR にコンテンツをデプロイするときに使用できるデジタルライツマネジメント (DRM) システムです。AIR の前バージョンでは Flash Access をデスクトップランタイムでのみサポートしていましたが、AIR 3 ではモバイルデバイスでも同様のセキュリティレベルが提供されます。



Adobe Flash Access に関する情報は、<http://www.adobe.com/jp/products/flashaccess/>を参照してください。

モバイルデバイスの AIR をターゲットとするアプリケーションを開発するときでも、使用するコードは、デスクトップ版 AIR をターゲットにしていたときのコードとまったく変わりません。現在、Flash Access 2.0 を通して DRM を処理するように設定されているアプリケーションはどれも、AIR 3 をサポートするモバイルデバイスに対しても、まったく同じように動作します。



Flash Access を Open Source Media Framework (OSMF) とともに使用すると、その併用を活かすために設けられたイベントやプロパティが利用できます。これに関する詳細は [http://help.adobe.com/ja\\_JP/FlashPlatform/reference/actionscript/3/org/osmf/events/DRMEvent.html](http://help.adobe.com/ja_JP/FlashPlatform/reference/actionscript/3/org/osmf/events/DRMEvent.html) で読むことができます。

セキュアな乱数ジェネレータ

新しい `flash.crypto.generateRandomBytes()` メソッドを使用すると、非常にセキュアでランダムなバイトのセットが生成できます。これは暗号化キーを必要とするアプリケーションに使用でき、銀行業務や会計処理、さらには高度なセキュリティが求められるアプリケーション一般に適用可能な、セキュアなセッションIDの生成に使用することができます。この暗号化されたセキュアなバイトを生成する実際の機能は、AIR 自体のものではなく、OS によるものです。

次の例ではこの新しいメソッドを使って、ランダムに生成され、暗号化された 1024 のセキュアなバイトを含む `ByteArray` オブジェクトを生成します。

```
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 = 0xFFFFFFFF;

            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 = "セキュアに生成されたランダムなバイト!¥n¥n";
        randBytes = generateRandomBytes(1024);
        traceField.appendText(randBytes.toString());
    }
}
}
}

```

コード内にブレークポイントを設定すると、生成された `ByteArray` の詳細がデバッガで表示できます。図 9-2 では、`ByteArray` には 1024 バイト含まれていることが分かります。これは、サンプルコードの `flash.crypto.generateRandomBytes(1024)` の呼び出しで要求したバイト数です。

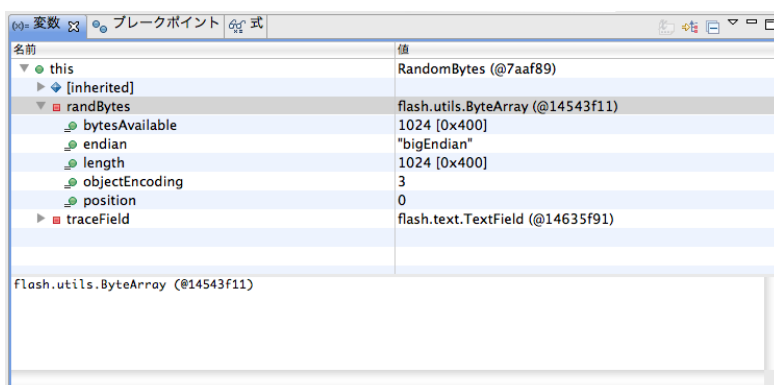


図 9-2: Flash Builder デバッガでランダムなバイトを調べる

**訳注:**

図 9-2 では Flash Builder を使っていますが、この情報は `ByteArray` クラスのプロパティを使用することで、

Flash IDE でも得られます。

```
trace(randBytes.bytesAvailable);  
trace(randBytes.endian);  
trace(randBytes.length);  
trace(randBytes.objectEncoding);  
trace(randBytes.position);
```



この新しいメソッドには、1 から 1024 までのバイト数が要求できます。

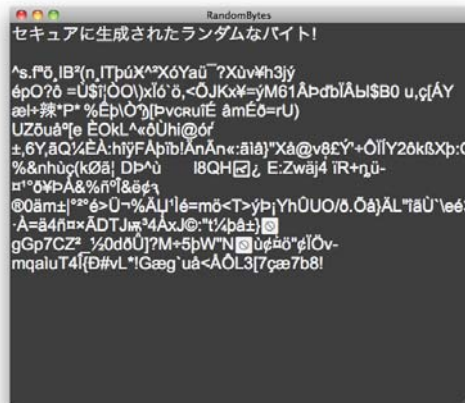


図 9-3: ランダムなバイトのストリング表現

## APPENDIX: 追加的なリソース

われわれは本書が、Adobe AIR 3 で可能になった新機能の理解と使用のスタートに役立つことを願います。さらに深く調べたい方には、次のリソースをおすすめします。

### What's New in Flash Player 11

AIR 3 には、Flash Player 11 という名前の、デスクトップとモバイルで実行するブラウザベースのアプリケーションランタイムがあります。Flash Player 11 の新機能については、「What's New in Flash Player 11」(<http://shop.oreilly.com/product/0636920021698.do>)で学ぶことができます。

### Stage3D フレームワークの使用

Adobe は、Stage3D が業界やコミュニティを通し、多くのプロジェクトでサポートされるように、多数のフレームワークやゲームエンジンと密接に連携しています。以下はその一部です。

#### 3D フレームワーク

- ・ 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](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>

## 記事とリソース

以下は Web 上で読むことのできる追加的なリソースです。

- ・ 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/>

- ・ How to Overlay the AIR SDK for Use With the Flex SDK

[http://kb2.adobe.com/cps/495/cpsid\\_49532.html](http://kb2.adobe.com/cps/495/cpsid_49532.html)

- ・ Native extensions for Adobe AIR

<http://www.adobe.com/devnet/air/native-extensions-for-air.html>

- ・ Setting Up Flash Builder 4.5 for Flash 11 and AIR 3 Apps

<http://www.fmsguru.com/showtutorial.cfm?tutorialID=59>

- ・ Overlay AIR SDK | Flash Professional CS5.5

[http://kb2.adobe.com/cps/908/cpsid\\_90810.html](http://kb2.adobe.com/cps/908/cpsid_90810.html)