



ActionScript コーディングガイド

By Michael Williams

2002年3月

Copyright © 2002 Macromedia, Inc. All right reserved.

以下に記載される内容は、このドキュメントを作成した時点での Macromedia の意向を示したものです。市場状況の変化に対応する必要性から、Macromedia では内容に責任を持つことはできません。従って、ドキュメント作成以降においては、記載情報に正確さを欠く場合もあり、ご了承ください。

このホワイトペーパーは、情報の提供のみを目的としたものです。Macromedia では、このドキュメントにおいて一切の保証、明示、示唆をしておりません。

Macromedia では、このドキュメントの主題に含まれる特許権、特許を受けたアプリケーション、登録商標、著作権、その他の知的財産権を所有します。Macromedia との文書における同意書がない限り、このドキュメントの保有によって、これらの特許権、登録商標、著作権、知的財産権を取得することはできません。

Macromedia ロゴ、ColdFusion、Flash、および Macromedia Flash は、米国または他国における Macromedia, Inc の商標または登録商標です。ドキュメントに記載される実在の会社名および製品名は、各所有者の商標です。

目次

本書について.....	4
スタイル.....	5
命名に関するガイドライン.....	5
コードコメントに関するガイドライン.....	7
例.....	7
タイムラインレイアウトに関するガイドライン.....	8
スコープ.....	8
相対アドレスで変数のスコープを指定する.....	9
_root と _global.....	9
ActionScript のコーディング標準作法.....	10
アクションを一箇所にまとめる.....	10
#include.as ファイルを使用する.....	11
アプリケーションの状態とフレーム.....	11
ムービークリップやボタンにコードを設定しない.....	11
初期化.....	12
ローカル変数に var を使用する.....	13
オブジェクトの生成.....	14
オブジェクトの継承.....	15
パフォーマンス.....	16
プリロード.....	16
SWF のデータタイプとサイズ.....	16
メモリーの使用に関する注意.....	17
最後に.....	17

本書について

Macromedia Flash のアプリケーションは、特定の標準やガイドラインを配慮せずに作成することが可能で、多くの場合、実際にそのような開発が行われてきました。この、Flash 特有の柔軟性により、単一の問題に対して多様な解決策が提供できる一方で、アプリケーションの作成者以外がアプリケーションコードを理解することは難しくなります。また、最悪の場合には、作成者自身が自分で書いたコードを解読できないというケースも発生します。アプリケーションコードを理解できなければ、コードのデバッグや変更、あるいはアプリケーションの再利用は困難となります。

本書は、特に、ActionScript を使ってコーディングし、Macromedia Flash によってアプリケーションを組み立てる開発者のために作成されたもので、Macromedia Flash のアプリケーションのための最も適切なコーディング方法と ActionScript の規約を簡単に解説します。このガイドラインに沿って開発されたアプリケーションは効率的で理解し易くなるとともに、アプリケーションを構成する ActionScript のコードもデバッグや再利用が容易になります。

スタイル

命名に関するガイドライン

最も重要なことは、各アプリケーションにおいて、オブジェクトや変数など、コーディング要素の名前の付け方に一貫性をもたせ、その目的を判断しやすい名前をつけることです。つまり、名前には意味のある語句を使用し、それぞれの要素が持つ基本的な機能や目的が名前から明確にわかるようにすることをお勧めします。ActionScript は動的にタイプ（型）が設定される言語であるため、名前が参照するオブジェクトのタイプを示すサフィックス（接尾辞）も名前に付加します。一般に、"名詞-動詞" または "形容詞-名詞" の語句を指定すると最も自然な名前になります。次に例を示します。

- ムービー名 - my_movie.swf
- URL に追加する要素 - course_list_output
- コンポーネントまたはオブジェクト - ProductInformation
- 変数またはプロパティ - userName

関数名と変数名の最初の文字は小文字にします。オブジェクトやオブジェクトコンストラクタは大文字で開始します。変数名には大文字と小文字の両方を使用することをお勧めしますが、アプリケーション内で統一されている限り他の形式を使用しても構いません。

変数名は、文字と数字およびアンダースコアだけで構成します。ただし、変数名の最初の文字に数字またはアンダースコアを使用することはできません。

不適切な変数名の例:

- `_count = 5;` //最初の文字がアンダースコアです
- `5count = 0;` //最初の文字が数字です
- `foo\bar = true;` //バックスラッシュが含まれています
- `foo bar = false;` //スペースが含まれています

ActionScript の予約語は、名前に使用してはいけません。また、一般的なプログラミングコンストラクトの名称は、現在 Macromedia Flash Player がそのコンストラクトをサポートしていない場合でも変数名として使用しないようにします。これにより Macromedia Flash Player の将来のバージョンアップで問題が発生することを避けることができます。次に不適切な例を示します。

- `var = "foo";`
- `MovieClip = "myMovieClip";`
- `switch = "on";`
- `case = false;`
- `abstract = "bar";`
- `extends = true;`
- `implements = 5;`

ActionScript は ECMAScript に準拠しているため、アプリケーション開発者は現行および将来の ECMA 仕様を参照することで ECMAScript の予約語リストを確認することができます。

Macromedia Flash MX では、常に同じ値を示すコンスタント（定数）変数の表記に仕様上の決まりはありませんが、やはり変数の意図を表す命名ルールを適用してください。たとえば、変数名はすべて小文字、定数名はすべて大文字にします。次に例を示します。

- `course_list_output = "foo"; //変数`
- `courseListOutput = "foo"; //変数`
- `BASEURL = http://www.foo.com; //定数`
- `MAXCOUNTLIMIT = 10; //定数`
- `MyObject = function{}; //コンストラクタ関数`
- `f = new MyObject(); //オブジェクト`

Macromedia Flash MX の ActionScript エディタには、コード補完機能があります。特定の命名ルールで変数名を付けると、この機能を利用できるようになります。デフォルトでは、変数のタイプを示すストリングを変数名にサフィックスとして付加します。次の表に、サポートされているサフィックスのストリングを示します。

表 1: コードコンプリートでサポートされているサフィックスのストリング

オブジェクトタイプ	サフィックス	例
String	<code>_str</code>	<code>myString_str</code>
Array	<code>_array</code>	<code>myArray_array</code>
MovieClip	<code>_mc</code>	<code>myMovieClip_mc</code>
TextField	<code>_txt</code>	<code>myTextField_txt</code>
Date	<code>_date</code>	<code>myDate_date</code>
Sound	<code>_sound</code>	<code>mySound_sound</code>
XML	<code>_xml</code>	<code>myXML_xml</code>
Color	<code>_color</code>	<code>myColor_color</code>

最後に、すべての SWF ファイルには小文字の単語をアンダースコアで区切った名前 (`lower_case.swf` など) を使用します。ActionScript include ファイルの名前のルールについては、次の「#include .as ファイルの使用」を参照してください。

上記のシンタックスの推奨事項はガイドラインにすぎません。最も重要なことは、特定の名前の付け方を選択したら、常にその規則に従うようにすることです。

コードコメントに関するガイドライン

アプリケーションのコードには必ずコメントを書くようにしましょう。コメントは、目的とするコードの動作を作成者が説明するために与えられた手段です。コメントには、アプリケーションの作成過程でなされたすべての決定を記録することができます。つまり、アプリケーションのコーディング方法についてなんらかの選択をしたポイントごとに、選択した方法と選択した理由を説明するコメントを書きます。

アプリケーションの問題を一時的に回避するコードを書いたときは、将来このコードを読む開発者のためにこの問題が何かを明確にするコメントを必ず付加してください。これにより、次に同じ問題が発生したときに対応が容易になります。

次に、変数の簡単なコメントの例を示します。

```
var clicks = 0; // ボタンをクリックした回数を示す変数
```

また、次のように、コメントのテキストが長いときはブロックコメントが便利です。

```
/*
 ボタンをクリックした回数を管理するクリックの変数を初期化します。
*/
```

重要なコメントを示すために、以下のようなコメントがよく使用されます。

```
//:要作業:トピック
ここに現在進行中の作業があることを示します。
//:バグ:<バグ ID>トピック
```

ここに既知の問題があることを示します。問題の説明も記述します。
適用できる場合はバグ ID も表示します。

```
//:要検討:
```

次のコードが洗練されていないこと、または最も適切なコーディングでないことを示します。他の人の注意を喚起して、次回に別の方法でコーディングするヒントを促します。

```
//:要注意:
```

後続のコードが多くのインタラクションを有することを開発者に知らせます。開発者はよく考えてから変更することが必要です。

例

```
/*
:要作業: msw 654321: データベースから大量のデータを表示するときの問題あり。
 リクエストごとにデータを小さい単位に分割することを考えてください。
*/
```

タイムラインレイアウトに関するガイドライン

Flash が自動的につけるレイヤー名 (レイヤー 1、レイヤー 2 など) は、そのままでは何の意味ももたないため、タイムラインのレイヤーには必ず名前を付けます (日本語も使用できます)。また、必要に応じ、フォルダを使用してレイヤーをグループ化します。また、タイムラインの一番上のレイヤーを ActionScript 専用のレイヤーにしておけば、ActionScript コードが記述されている場所をすぐに見つけることができます。

そのほか、現在編集作業の対象になっていないすべてのレイヤーをロックすることもお勧めします。これにより、ドキュメントを誤って変更することが避けられます。

スコープ

Macromedia Flash Player 6 には、ECMA-262 標準で定義された「スコープチェーン」の概念が採用されています。この仕様の採用は、厳密なスコープというものが存在しなかった Macromedia Flash Player 5 の仕様と異なる重要なポイントです。

スコープチェーンとは、ActionScript オブジェクトの検索順のことです。変数名、メソッド名などの識別子を解釈する際、ActionScript はスコープチェーンの最後の要素からさかのぼり、合致する識別子があるかどうかを検索していきます。

標準的な ActionScript のスクリプトでは、スコープチェーンは以下のオブジェクトから構成されます (下から順に上に検索)。

- グローバルオブジェクト
- 包含する MovieClip オブジェクト
- ローカル変数

`with` アクションを使用すると、スコープチェーンの最後に一時的に特定のオブジェクトを追加することができます。`with` アクションの実行が完了すると、一時的なオブジェクトはスコープチェーンから削除されます。

関数を定義すると、関数オブジェクトに現在のスコープチェーンがコピーされて保存されます。関数を呼び出すと、スコープチェーンが関数オブジェクトのスコープチェーンに切り替えられ、新規のローカル変数オブジェクトがチェーンの最後に付加されます。

Macromedia Flash 5 では、ActionScript 関数のスコープは常に以下のようになっていました (下から順に検索)。

- グローバルオブジェクト
- 関数を包含するムービークリップ
- ローカル変数

`with` アクションを使用したときを除き、スコープリストのエントリが 3 つを超えることはありません。この仕様は、ECMA-262 標準とは異なっていました。結果として、メソッドのスコープはメソッドが設定されたムービークリップにあり、ムービークリップのメソッドが定義された場所ではありませんでした。

Macromedia Flash Player 6 では、ムービークリップのメソッドをムービークリップの外部で定義すれば、メソッドのスコープチェーンはその外部オブジェクトを対象とし、ムービークリップオブジェクトに含まれることはありません。このため必要になるのが、this キーワードをメソッド本文で使用するよう習慣づけるということです。

この仕様変更は、Macromedia Flash Player 5 コンテンツとの下位互換性を保持してなされています。したがって、Macromedia Flash 5 のコンテンツには、Macromedia Flash Player 5 で使用されたのと同じ、つまり Macromedia Flash Player 6 と異なったスコープの規則が適用されることに留意してください。

相対アドレスで変数のスコープを指定する

変数を記述する際は、常にスコープを指定するようにします。スコープを指定しない変数は、関数のパラメータとローカル変数だけです。変数のスコープは、可能であればカレントパスを基準に指定します。変数のスコープを `_root` から指定することは、コードの移植性が制限されるため勧められません。代わりにキーワード `_parent` または `this` を使用します。次に例を示します。

```
this.myVar.blah = 100; // このように相対アドレスで変数のスコープを指定する
```

```
_root.myMovieClip.myVar.blah = 100; // このように絶対アドレスでは変数のスコープを指定しない
```

メインタイムラインの絶対アドレスを使用する必要がある場合は、`_root` を使用する代わりにメインタイムラインを参照する変数を作成します。これにより、タイムラインの構造が変化する場合でも、1つのパラメータだけの簡単な変更が可能になります。ムービーのメインタイムラインへの参照を作成するには、メインタイムラインに次のようなコードを記述します。

```
_global.myAppMain = this; // ("myApp" はアプリケーションの名前で置き換えます)
```

このアクションを実行した後は、`_global.myAppMain.someFunction` という形式で簡単にメインタイムラインの関数を参照することができます。これにより、アプリケーションの構造が変化しても、ムービーの関数呼び出しや変数のスコープに影響を与えることはありません。

`_root` と `_global`

`_global` と `_root` の違いを認識しておくことが重要です。`_root` はロードした各ムービーに固有のもので、`_global` は Player 内のすべてのムービーに適用されます。一般に、`_root` によるメインタイムラインへの参照よりも `_global` の使用をお勧めします。

ActionScript のコーディング標準作法

アクションを一箇所にまとめる

できるだけすべてのコードを一箇所にまとめます。これによりコードの検索とデバッグが簡単になります。Macromedia Flash ムービーのデバッグにおける面倒な作業の1つが、コードを見つけることです。大部分のコードが一箇所にあれば、この問題は解消されます。通常、コードの場所として最適なのはフレーム 1 です。

最初のフレームに大量のコードを配置するときは、次のようなコメントで複数のセクションに分けて読みやすくします。

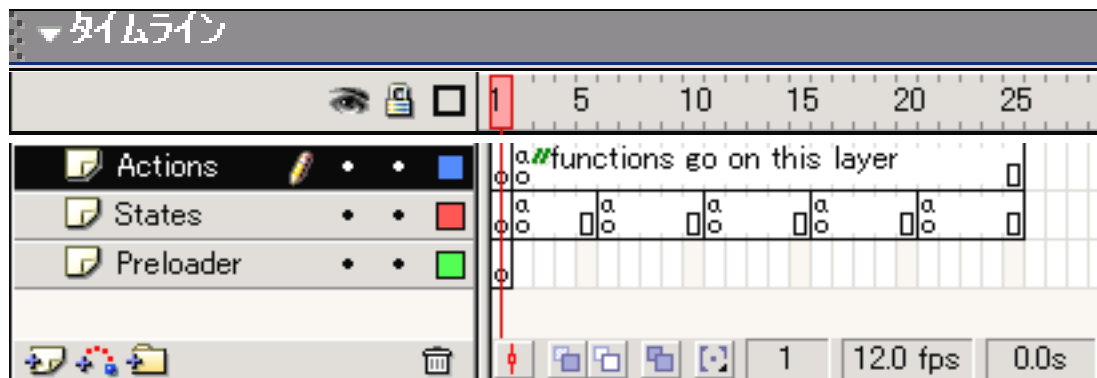
```
/** ボタンと関数のセクション **/  
/** 変数と定数 **/
```

例外として、ムービーをプリロードするときは、すべてのコードをフレーム 1 に配置することができない場合があります。しかし、アプリケーションを作成する場合はなるべく、すべてのコードを 1 つの場所に配置するよう試みてください。

プリローダーを使用するムービーでは、アクションをフレーム 2 で開始します。アクションには 2 つのレイヤーを使用します。一方のレイヤーは関数定義に使用し、もう一方のレイヤーは状態に依存するアクションの定義に使用します。

ActionScript で作成したすべての関数とオブジェクトは、ムービー全体にわたり有効です。一方、ムービークリップはタイムラインの状態に基づいて作成および破棄されます。このため、ムービー全体のための関数またはオブジェクト定義と、ムービークリップおよびシーン特有の関数呼び出しは、2 つのレイヤーに分けて記述することをお勧めします。下図のタイムラインはこのルールを反映して作成されたものです。

図 1: プリローダー付きのタイムラインの例



#include.as ファイルを使用する

アプリケーション内の関連する関数のカスタムオブジェクトやライブラリは、それぞれ外部の ActionScript ファイルに定義して、include ファイルとして使用します。たとえば、アプリケーションで "Foo" というオブジェクトと "Bar" というオブジェクトを使用する場合、各オブジェクトは個別の ActionScript の include ファイルに定義し、そのファイル名を、次のようにオブジェクト名と対応させます。

```
Foo.as  
Bar.as
```

再利用する関数のライブラリも ActionScript の include ファイルに定義します。この場合、include ファイル名には次のように、大文字と小文字の両方を使用します (最初の文字は小文字)。

```
stringUtils.as
```

このように include ファイルを使用することにより、その役割によって分類された ActionScript コードの特定が容易になります。しかし、もっと重要なことは、コードのモジュール化です。開発者は、コードのモジュール化によって ActionScript のオブジェクトや関数のライブラリを作成することができます。アプリケーションに必要なコードだけを外部ファイルのライブラリから読み込むことができます。

また、外部の ActionScript ファイルを使用すれば、開発者はファイルを CVS や SourceSafe などのバージョン管理システムと統合でき、さらにコードがモジュール化され、追跡管理が容易になります。

アプリケーションの状態とフレーム

開発者はしばしば単一フレーム内に記述したコードだけでアプリケーションの状態を定義しますが、これは勧められません。

可読性を考えてコードを整理するには、キーフレームを使用してアプリケーションの状態を記述します。まず、特定の状態に必要なアクションを1つの関数として指定して、次にこの関数の呼び出しをそのキーフレームで実行する唯一のフレームアクションとして記述するようにします。

コードをさらに整理するために、タイムライン上では2つのレイヤーを作成します。大部分のコードは、ムービーの全長にわたり存在するレイヤーに配置します。もう一方のレイヤーには、アプリケーションの状態変化を表すために必要とされるキーフレームのアクションを配置します。各キーフレーム上に存在するコードは、それぞれの特定のアプリケーションの状態に対する関数呼び出しのみとなります。また、attachMovie() アクションによって実行時に付加されたムービークリップを使用して、別の状態 (ダイアログボックスなど) を表すこともできます。

ムービークリップやボタンにコードを設定しない

絶対に必要でない限り、ムービークリップやボタンにコードを設定することは避けます。ムービークリップやボタンにコードを設定する必要があるときは、最小限のコードのみ使用します。次のように、関数呼び出しを使用する方法をお勧めします。

```
myButton.onMouseDown = function() { _parent.doMouseDown(this); }
```

このように関数呼び出しを外部から設定することで、すべての機能をムービークリップのメインタイムラインに配置することができます。

初期化

アプリケーションが起動した直後の状態を設定するためには、初期化処理が必要になります。初期化はアプリケーションの最初の呼び出し関数内で行い、完了するべきです。他の呼び出し関数は、すべてイベントが発生したときに実行されることになります。

```
// フレーム 1
this.init();
function init()
{
    if ( this.inited != undefined )
        return;
    this.inited = true;
    // ここに初期化コード...
}
```

ローカル変数に var を使用する

すべてのローカル変数にキーワード var を使用するように習慣づけます。これにより、変数のグローバルアクセスと不慮の上書きが防止されます。たとえば、次のコードではキーワード var を使用して変数を宣言していないため、別の変数による不慮の上書きが発生する可能性があります。

```
counter = 7;
function loopTest()
{
    trace(counter);
    for(counter = 0; counter < 5; counter++)
    {
        trace(counter);
    }
}
trace(counter);
loopTest();
trace(counter);
```

このコードでは次の値が出力されます。

```
7 7
0
1
2
3
4
5
```

8 このケースでは、メインタイムラインの counter 変数が関数内の counter 変数によって上書きされます。次の修正したコードでは、キーワード var を使用して両方の変数を宣言しています。関数内で var 宣言を使用することにより前のコードの問題点が修正されています。

```
var counter = 7;
function loopTest()
{
    trace(counter);
    for(var counter = 0; counter < 5; counter++)
    {
        trace(counter);
    }
}
trace(counter);
loopTest();
trace(counter);
```

オブジェクトの生成

オブジェクトを生成するときは、オブジェクトのメソッドとプロパティをオブジェクトのプロトタイプに設定し、オブジェクトの全インスタンスで共有します。これにより、メモリ上に存在するメソッドのコピーは1つだけになります。原則として、コンストラクタ内でメソッドを定義することは避けます。コンストラクタ内でメソッドを定義すると、同じメソッドのコピーがオブジェクトの各インスタンスごとに作成され、結果的に不必要にメモリを消費してしまいます。

次の例は、オブジェクトを生成する最も適切な方法です。

```
MyObject = function()
{
}
MyObject.prototype.name = "";
MyObject.prototype.setName = function(name)
{
    this.name = name;
}
MyObject.prototype.getName = function()
{
    return this.name;
}
```

次の例は、オブジェクトを作成する不適切な方法です。

```
MyObject = function()
{
    this.name = "";
    this.setName = function(name)
    {
        this.name = name;
    }
    this.getName = function()
    {
        return this.name;
    }
}
```

最初の例では、MyObject の各インスタンスはオブジェクトのプロトタイプに定義された同一のメソッドとプロパティを参照します。このため、MyObject オブジェクトがいくつ作成されても、メモリ上に存在する getName() メソッドのコピーは1つだけになります。

2 番目の例では、MyObject のインスタンスが作成されるたびにプロパティとメソッドのコピーが作成されます。これらのプロパティとメソッドのコピーは余分なメモリを消費するだけで、ほとんどの場合、なんの利点もありません。

オブジェクトの継承

Macromedia Flash Player 5 では、オブジェクトの継承のために `__proto__` プロパティを設定していました。この方法は Macromedia Flash Player 6 ではサポートされなくなり、勧められません。`__proto__` は、読み取り専用プロパティとして扱う必要があります。正しい継承の方法は、プロトタイプチェーンを作成することです。プロトタイプチェーンを作成するには、次のシンタックスを使用してサブクラスコンストラクタの関数 `prototype` プロパティをスーパークラスのインスタンスに設定します。

```
ChildClass.prototype = new ParentClass();
```

この方法の例を以下に示します：

```
function Shape()
{
}
function Rectangle()
{
}
Rectangle.prototype = new Shape();
```

次の方法は勧められません：

```
Rectangle.prototype.__proto__ = Shape.prototype;
```

この継承の方式ですべてのコンストラクタコードが不必要に呼び出されることが心配な場合は、次のコードでこれを防止できます：

```
_global.SuperClassConstructor = function() {
  if (this._name!=undefined) {
    // ここに実際のコンストラクタのコードを書きます
  }
}
```

上記のコードではインスタンスが未定義のため、コンストラクタのコードは実行されません。このコードは、ムービークリップをベースとするクラスでのみ機能します。

パフォーマンス

プリロード

プリロードに適したコンポーネントを作成するときの主な問題は、最初のフレームの前にロードするアイテム数をどこまで少なくできるかということです。最初のフレームのシンボルがロードされるまで、グラフィックはステージ上に表示されません。つまり、"最初のフレームに書き出し"と設定されているすべてのシンボルがロードされるまで、プリローダーのコンポーネントも表示されません。このため、ユーザーとの対話を処理する ActionScript は、必ずプリローダーより後のフレームに配置するようにならなければなりません。

SWF のデータタイプとサイズ

次のデータを使用してアプリケーションの重要な部分のロード時間を最適化することができます。

表 2: 主な ActionScript のオブジェクト

オブジェクトのタイプ	サイズ
短いストリング (7 文字)	55 バイト
数字	22 バイト
オブジェクト (new Object())	340 バイト
空のムービークリップ (createEmptyMovieClip())	800 バイト
関数オブジェクト (x = function() {})	700 バイト
テキストフィールド (7 文字)	9,500 バイト

表 3: シンプルな SWF ファイル

シンプルな SWF のタイプ	サイズ
空の swf ファイル	250,000 バイト (大部分は定義済みのグローバルな ActionScript オブジェクト)
コンポーネントが定義された空の swf	750,000 バイト (大部分は swf ファイルとコンポーネントの関数オブジェクト)

表 4: コンポーネントのオブジェクト (画面上のインスタンス当たり)

コンポーネントのオブジェクト	サイズ
PushButton	60,000 バイト
CheckBox	55,000 バイト
空の ListBox	490,000 バイト

メモリーの使用に関する注意

テキスト - テキストフィールドの各オブジェクトは最低約 10kb のメモリーを使用します。これには、デバイスフォントを使用する入力テキスト、ダイナミックテキスト、静止テキストといったフィールドがあります。

ムービークリップ - 各ムービークリップは最低約 1kb のメモリーを使用します。この値は、使用するムービークリップの数に比例して増加します。グラフィックシンボルはムービークリップほどメモリーを必要としません。

関数オブジェクト - 各関数オブジェクトは、2 つの ActionScript オブジェクト (最低合計 700 バイト) を生成します。オブジェクトのインスタンスごとに関数オブジェクトを作成すると、大量のメモリーを消費することになります。常にプロトタイプを使用することをお勧めします。プロトタイプは 1 回だけ作成されるため、このようなメモリーのオーバーヘッドを避けられます。

最後に

本書は ActionScript の最適な用法を完全に網羅したものではありませんが、Macromedia 社では Macromedia Flash Player を使用してミッションクリティカルなアプリケーションを開発および導入/展開している内外の方々からのご意見と技術情報提供に基づいて本書を継続的に拡充していきます。また、本書は、ActionScript や Macromedia Flash Player が誕生して以来そうであったように、実際にテクノロジーを利用していただいている方のコミュニティからの貴重なご意見があったからこそ完成したものと弊社では考えております。また、今後も頂戴したご意見を反映させ、以後の著書へ反映させたいと思います。ご協力いただきました皆様へ厚くお礼申し上げます。